Intermediate Lower Bounds and Their Relationship with Complexity Theory

by

Dylan McKay

B.S. Computer Science
B.S. Discrete Mathematics
Georgia Institute of Technology, 2015
M.S. Computer Science
Stanford University, 2016

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
August 28, 2020

Certified by .....
R. Ryan Williams
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Intermediate Lower Bounds and Their Relationship with Complexity Theory

by

Dylan McKay

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

While Complexity Theory has been centered around several major open problems about the relationships between complexity classes, showing resource lower bounds which amount to much weaker versions of these separations still seems to be challenging. We examine some of these lower bounds and techniques for showing them.

We improve the techniques of Beame (1989) and use these results to show time-space lower bounds for various circuit problems such as #SAT and a version of SAT for which we are required to give witnesses to satisfiable formulas.

We reveal a surprising significance of lower bounds of this kind by presenting their relationships with long-standing questions in Complexity Theory, notably by showing that certain weak lower bounds against the Minimum Circuit Size Problem (MCSP) and other compression problems would imply strong complexity class separations such as $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{NP} \not\subset \mathbf{P}/poly$.

We further explore techniques for proving lower bounds as well as the connections between lower bounds and the big picture of Complexity Theory. In doing so, we explore the technique of proving fixed polynomial circuit size lower bounds through improvements to the Karp-Lipton theorem and give surprising evidence that improvements to the Karp-Lipton Theorem are (in some sense) the "only" way to prove fixed polynomial size circuit lower bounds against $\mathbf{P}^{\mathbf{NP}}$.

Thesis Supervisor: R. Ryan Williams
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I wish I could make this section as long as the rest of my thesis. There are so many people I want to show gratitude for their part in my journey. Unfortunately, as I never broke my habit of procrastination, I am left with little time for the thoroughness I want to give. I am sure that I will leave out someone to whom I owe remembrance.

Alec Interrante, Alex Reid, Basil Grant, Courtney Reid, and Evan Vana. You made Southern Poly feel like a new home, and you made it hard to leave. Watching you all grow together after I left was strange and sometimes difficult. You are the first family I found after leaving home, and I'm so grateful that I have managed to find you all in my life again.

AJ Kolenc, Alec Clifford, Casey McArthur, Mark Petell, Ramon Romero, Robert Kretschmar, and Vince Lugli. You were my family at Georgia Tech. You made this scary next step in my life one that I could never regret. I grew so much with all of you, and I had some of the best times of my life with you.

Kat and Nolan Hackett, Charlie Kilpatrick, Rachel Elliott, Andrey and Anna Kurenkov, Charles Wang, Peter Woolfitt, and Alex Bettadapur. I still remember the day Nolan invited me to come watch Primer with him and his friends. Since then, I feel like I have become part of another family. This was a rare gift that I will cherish forever.

Basil Grant, Justin Payne, Mark Fredricks, Chase Wilkins, Chris Hamrick, Chris Nyberg, Joel Clewis, Zach Tenney, Zach Fultz, Evan Vana, Casey Price, Harrison Smith, and Brent Dupont. Thank you for keeping me sane with the countless nights of discord voice chats during the quarantine. Some of you are new friends I did not expect to make during these times, and others are old friends who I am so happy I have gotten the chance to bring back into my life.

My Stanford cohort. The beginning of graduate school was a hard adjustment for me. It was one of the first times in my life where I felt I knew no one around me. I cannot believe how lucky I was to meet all of you, and I cannot imagine how hard things would have been without you. Some of you were role models for me, some of

you had a way of provoking me to think about great interesting things, and some of you became my amazing friends.

My MIT cohort. Boston was a scary city that I did not want to be in when we moved to MIT. It was cold and lonely, but many of you made the Stata center a warm home for me. At the beginning, I came to know a few of you from staying late, goofing off bouncing balls into the fifth floor "first year pit" from the sixth floor. Then there were all the theory jams and music nights. There were the evenings spent playing Super Smash Bros. Melee and others still spent watching lots of you play chess.

Greg Bodwin. Thanks for being such a great role model, friend, and keyboardist. Also, thanks for being so supportive of my drumming.

Rahul Ilango. Thanks for review part of my thesis for me, and thanks for being a great friend. You are a bright star that has shone in my life only briefly so far, but you have already made the theory group and MIT a better place for being there.

Chris Peikert, Dana Randall, István Miklós, Omer Reingold, Tim Roughgarden, and Virginia Williams. You have all been mentors to me throughout my journey, and you have been role models as I envisioned who I wanted to be. Thank you so much for all of your guidance, wisdom, and support.

Ronitt Rubinfeld and Michael Sipser. Thank you so much for serving on my committee and offering revisions to my thesis. Thank you again, Ronitt, for the moments when you talked to me about teaching and presenting. You made me feel like I was where I needed to be when you allowed me to give input for your class.

All of the theory group administrators, especially Rebecca Yadegar. Thank you for all of your help making our lives easier.

Leslie Kolodziejski. Thank you for all of the listening and invaluable advice.

"Venkat" Venkateswaran. I remember sitting in Klaus, and you just started talking to me. I don't remember if I told you my recent interests if you heard about them somehow, but one way or another you knew that I had become enamoured with Gödel's incompleteness theorems. You saw the situation much clearer than I did, and you gave me a good push into the world of complexity theory. You suggested at take

Lance's *Automata and Complexity* class and Dick's *Complexity Theory* graduate class at the same time, and you told me to ask them if I could do research with them. I don't know what all you knew then, but you saw me for what you knew I could be. Most people are not lucky enough to have someone like that, and it is a gift that I remember often and dearly. You also gave me my first big time teaching opportunity. You and Lance somehow got the department to hire me as a research scientist and appoint me as a lecturer for the Summer because you saw how badly I wanted to teach. Thank you so much for all of this. I hope I get to have this kind of impact on someone else someday.

Richard Lipton. Working with you was always fun. I would show up to your office or follow you from class, and you would talk my ear off about whatever amazing ideas had been going through your mind that morning. You taught my first complexity theory class, and I think my view of complexity theory will always carry that. You let me be a teaching assistant for your graduate complexity theory class, and your letter was instrumental for my graduate school applications. No matter how jaded I become with regards to research, I will always remember working with you as evidence that research really can just be about having fun. I will always admire you and be grateful for what you have done for me.

Lance Fortnow. You were my guide and a true mentor. You took me under your wing and showed me a world to explore. You invited me into your home, and you helped me figure out where to go with my life. You helped get me a job lecturing for the Summer between undergrad and graduate school, and you found a way to make the same generous offer when I thought about taking a semester off of graduate school. You have always been there for me not only to teach me about research and complexity theory, but for my career. You are a true role model and one of my personal heroes. I am forever grateful for everything you have done for me.

Anna Kohler, Kim Mancuso, and all of my acting friends. During one of the hardest periods of my life, I was coping with so much loss and so many feelings of purposelessness. I found a family with a small group of actors at MIT. They put me through some of the most intense work I have ever done. They brought me out of my

comfort zone and into a place where I felt connected and cared for. Thank you so much for being with me through these dark times. I'll always cherish the experiences we had.

Music friends. Music continues to be one of the most important parts of my life. Thank you all of my friends who keep pulling me back into music and sharing new beautiful things with me.

Max Madzar and Bennett Kane. You supported me through our shared love of music. You listened to my songs and you made me feel like everything was worth something. These moments were the greatest gifts, and I received them more time than I can remember thanks to you.

Chris Stookey. Over the years, visiting you has been as much a constant in my life as visiting my family. Seeing you always felt like a certainty when I visited home. You're like a brother to me, and you have been there through all of the good times and the bad.

AJ Kolenc. I remember meeting you and hearing about you in high school, but for some reason, I just cannot remember how we became friends, let alone how we became such great friends. You knew me better than anyone through high school and college, and you were not just my closest friend. You were my sanity. You were a constant source of joy and acceptance. I often miss the closeness of our old friendship, but I love seeing who you have become.

Emily Kurtz. You are without a doubt one of my most sincere friends. You are so often there for me when I need you, and I feel I can always count on you for clarity. I have not seen you in years, but the all of our multiple-hour long phone calls throughout grad school have been.

Fred Koehler and Nadine Javier. You two have been an integral part of my life for my last couple of years at MIT. You have been support I badly needed and friends I could count on.

Charles Wang. You are the quiet extrovert opposite my loud introvert. I can still feel you pulling me out the the hole of depression, and I'm so grateful to have someone like you in my life. If one were to figure your future based on our strange

series of co-locations, maybe you will find your way to Ohio in the future. Wherever you go, I wish only the best for you.

Aleksander Makelov. You are my brother from another motherland. I wish we never had to leave that house on Pine St. You are a true friend, and I hope that someday you finally answer the question: "what does it even mean?"

Josh Alman. You stuck with me through my whole graduate school career. You showed me the ropes. You were my first coauthor. I think there are many ways in which we have become each other, and I cherish them. I will always miss taking the long way home and standing on street corners, just to keep talking. I know you are going great places, and I hope that I get to share some of them with you again.

To all of my extended family. Mama, Papa, Grandmommy, and all of my cousins, aunts, and uncles. You have all been so unconditionally supportive. You all give and ask for nothing in return, and you always have a way of making me feel like I am worth something. You always have a way of making me feel loved.

Granddaddy. I think of you and miss you often. At your funeral, there were so many things said about you that I did not know, but were as true of you as they were of me. I am sad that I was blind to so many of these connections when you were still with us, but I have to assume that you are responsible for so many of our similarities. When I was little, you used to make me feel so clever. You used to tell everyone about a joke you would tell to trick people and how I wouldn't be tricked by it. You encouraged me to go to Georgia Tech, and when I didn't get in, you encouraged me to transfer. I like to tell people the last war story I remember you telling me. It was about a scout from the opposition marching a chicken through the jungle trying to bait traps. I think a lot about how many lives you probably saved by having the composure and patience to do nothing and you and your party laid hidden, watching the man and the chicken and then the subsequent hundred-something men go by. I miss you a lot, and I am grateful for everything I have because of you.

Robbie. You were an older brother to me. I still remember our last conversation in which you commented on my receding hairline in your usual way of just being blunt with no ill-intention. When the conversation was over, you said "bye", as if you

somehow knew that this was a moment that called for more than "see ya." I walked back into my parents' house, and you walked back into your dad's house, and that was it. I still look back to a night I sat outside with my guitar, trying to figure out how to sing and play "smells like teen spirit" at the same time. It was a time when I was so used to being told I was a bad singer that I *knew* I was a bad singer, and I was miserably self-conscious teenager. You didn't tell me I was bad. You told me to sing louder, and I'll always be grateful for that. I won't curse here, even though I know you would appreciate it a lot. I miss you a lot, and thanks for everything.

Mom and Dad. You saw me through this journey from beginning to end. There is so much to say here that it is hard to pick anything. You have always tried so hard to give me what you thought I needed in order to be what I wanted to be. This thesis might not exist if Dad did not teach me fractions using paper plates while I took baths, and it might not exist Mom did not buy me my first programming book in high school. I love you both so much. I know how hard you have worked to give me everything I have been given, and you have both given me all the support in the world in my journey through academia. I wish I could give you a much more grandiose entry here. Thank you for everything, and thank you again for all your love.

Emily McKay. I remember the time I wrote you a letter as you were finishing high school. A lot of times when I think about what I want to say to you when we talk, I want share wisdom in a way that I thought I was able to back then. You are not a kid anymore, though. I see who you have become, and I am so proud of you.

Ryan. Where do I begin? Ryan is the adviser for my PhD. He is a dear friend and a mentor. When I first met you, I wanted to be just like you. You shaped my philosophy on complexity theory. You shaped my philosophy on how to do research. At times, you were my confidant, and at others you were my confidence. My helped me find my feet as I grew, and even though I was always full of doubt, you never failed to *truly* convince me that my doubts were unwarranted. You were my inspiration. You *still are* my inspiration. I will miss hearing all of your amazing ideas which connect two things that I never thought two connect. I will miss our music nights. I

will miss all of your concerns about how I treat myself and how I feel about myself. I will miss random arguments about food and music, and I will miss the random moments of solidarity through our southern kinship. I knew you were the adviser for me from the day we met, and I am so grateful we found each other. Thank you for revising my thesis and all our papers. Thank you for pushing me to be who you saw I could be. Thank you helping me become who I am. Thank you for everything.

Contents

Chapter 1

Introduction

The **P** vs **NP** question has dominated Complexity Theory since its conception. It has become so iconic that its influence can be seen not only in other fields but even in pop culture. The question can be asked in many ways, such as: is there a polynomial time algorithm for the Satisfiability Problem (SAT)? Answering the **P** vs **NP** question would mean either showing that there is a polynomial time *upper bound* on the time required to decide SAT, or (often considered the more likely outcome) showing that every polynomial is a *lower bound* on the time required to decide SAT.

Our history in trying to answer this question has become great evidence that this question is hard to answer. Many would call this an understatement, say that it is not much of a stretch to claim this is a *very hard* question. Our difficulty in answering this question has birthed a beautiful web of techniques and proofs as well as new areas of study. It has also birthed a slew of questions that at times feel just as hopeless to answer. We now turn our attention towards not just the mountain that is the **P** vs **NP** question, but some of the questions that seem to lie at the ground level – proving: **intermediate lower bounds**.

Intermediate lower bounds are those that, one way or another, we *must* prove on our way to proving $\mathbf{P} \neq \mathbf{NP}$ or some other long sought-after result we covet as a community. We do not offer a formal definition of intermediate lower bounds. We use this notion simply as a way to capture the space between what we *want* to know and what we *currently* know, and we have no intention of using this notion in a formal

argument.

Our work is centered around intermediate lower bounds with respect to three conjectures or ideas.

SAT is hard for algorithms ($\mathbf{P} \neq \mathbf{NP}$). Starting with the classic **P** vs **NP** question, there is a wealth of intermediate lower bound questions for which we do not have answers. If we cannot yet prove SAT cannot be computed by algorithms running in polynomial time, can we at least prove that SAT cannot be decided in logarithmic space? Or can we prove that SAT cannot be decided in time $O(n^2)$? Can we at least prove that SAT cannot be decided in time logarithmic space when the machine is also only allowed to run in time $O(n^2)$? It turns out that we do not yet know the answers to any of these questions!

SAT is hard for circuits ($\mathbf{NP} \not\subset \mathbf{P}/poly$). A similar line of questions have come from the somewhat ironic[1] idea that circuits are "easy" to analyze – at least compared to algorithms. Perhaps the road to proving $\mathbf{P} \neq \mathbf{NP}$ involves showing $\mathbf{NP} \not\subset \mathbf{P}/poly$. It may very well be the case that we will not know that polynomial time algorithms cannot decide SAT until we can show that polynomial size circuits cannot decide SAT. While it may have once been an approach met with much optimism, it is still open to prove $\mathbf{NEXP} \not\subset \mathbf{P}/poly$ or even that **NP** has a language which cannot be decided by circuits of size $O(n^2)$, both of which are intermediate lower bounds with respect to the desired end result of $\mathbf{NP} \not\subset \mathbf{P}/poly$.

The Minimum Circuit Size Problem (MCSP) is hard. We now widen our view and change our focus from SAT to the Minimum Circuit Size Problem (MCSP) – the problem of finding the smallest circuit computing the function for a given truth table. MCSP can be thought of as the algorithmic task of proving circuit lower bounds. Obviously this is very important to us, not only because we would like to prove or disprove circuit lower bounds, but because this problem has demonstrated an enormous impact on our view of Complexity Theory, especially in regards to randomness, derandomization, and nondeterminism. Lower bounds against MCSP are crucial for many cryptography assumptions, and lower bounds for deciding MCSP

[1] This idea is ironic because our goal is to show that circuit analysis is *hard*.

may be as important as lower bounds for deciding SAT. Just as with SAT, we do not know if MCSP can be decided by algorithms running in time $O(n^2)$ or space $O(\log(n))$.

Some relevant examples of intermediate lower bounds can be found in Kannan [62], Fortnow [43], Williams [90], and Beame [19], among others.

Our contributions. In this thesis, we study **intermediate lower bounds**. Moreover, we study techniques for proving these lower bounds, as well as the relationship between these lower bounds and the big picture of Complexity Theory.

We first show new intermediate lower bounds for various natural circuit problems. We give lower bounds against the product of the length and space of branching programs solving some simple problems building on earlier work of Beame [19]. Our results imply lower bounds on the product of time and space used by any random access machine solving certain problems, and we show that they hold even for randomized machines. We then give reductions from these simple problems to show nearly quadratic lower bounds against the product of time and space for any randomized random access machine evaluating a circuit with many output bits, solving the satisfiability problem under the constraint that a satisfying assignment must be produced, and #SAT (the problem of counting satisfying assignments to a boolean formula). These lower bounds also hold for the problem of printing the truth table of a formula. For some of the aformentioned problems, our lower bounds are curiously tight; the lower bounds can also be "beaten" in slightly stronger models! More details will be given in Section 1 and Chapter 3.

Next, we examine intermediate lower bounds against the Minimum Circuit Size Problem (MCSP) and other compression problems. More specifically, we consider the following family of circuit minimization languages. Let $s : \mathbb{N} \to \mathbb{N}$ in the following.

Definition 1.0.1 (MCSP$[s(n)]$)**.** *Given a truth table T of size $N = 2^n$, decide if there is a circuit C of size at most $s(n)$ whose truth table is T.*

Even for small $s(n)$ such as $s(n) = n^{10}$, there is strong evidence that MCSP$[s(n)]$ cannot be decided in polynomial time [79, 61]. We show somewhat surprisingly that proving even very weak intermediate lower bounds against this language would allow

us to conclude coveted lower bounds. In fact, even showing $\mathsf{MCSP}[n^{10}]$ cannot be computed by an algorithm using $\log^{O(1)}(N)$ space and $N \log^{O(1)}(N)$ time would allow us to conclude $\mathbf{P} \neq \mathbf{NP}$! Thus, in a sense, such intermediate lower bounds are not really 11 intermediate", as they are already as difficult as problems like $\mathbf{P}$ vs $\mathbf{NP}$. More details will be given in Section 1 and Chapter 4.

Finally, we examine a long known method for proving circuit lower bounds for problems in the polynomial hierarchy. Kannan [62] shows that $\mathbf{PH}$ does not have fixed polynomial sized circuits. That is, for every constant k, there is language $L \in \mathbf{PH}$ such that L does not have circuits of size $O(n^k)$. Next, the famous Karp-Lipton theorem shows that, if $\mathbf{NP}$ has polynomial size circuits, then $\Sigma_2^p = \mathbf{PH}$ [63].

This pair of facts yields a very simple argument that Σ_2^p does not have fixed polynomial size circuits! The argument given by Kannan is as follows. First, suppose $\mathbf{NP}$ has polynomial size circuits. Then, $\Sigma_2^p = \mathbf{PH}$. Since $\mathbf{PH}$ does not have fixed polynomial size circuits, we are done. On the other hand, suppose $\mathbf{NP}$ does not have polynomial size circuits. Then Σ_2^p does not have polynomial size circuits, let alone fixed polynomial size circuits!

This "win-win" argument allows for quite a bit of generalization. Any new theorems in the same style as the Karp-Lipton Theorem produce new fixed-polynomial size circuit lower bounds. We explore this technique and ultimately give strong evidence that it is, in some sense, universal. More details will be given in Section 1 and Chapter 5.

Organization of This Thesis

This thesis is divided into six chapters. Chapter 1 is an overview of the main ideas presented in this thesis. Chapter 2 is brief preliminary chapter to direct the reader towards resources and notions which are important to understand for this thesis. Chapter 6 is a short exposition on some open problems motivated by bringing together the results in the body of this thesis.

In the remainder of this section, we describe the body of this thesis and reintroduce

the previously mentioned ideas with some additional concreteness.

Nearly Quadratic Time-Space Lower Bounds Against Natural Problems

In [19], Beame showed that the Unique Elements problem, and consequently the problem of sorting, cannot be computed by any branching program P such that the product of the space and length of P is $o(n^2)$. This implies that in fact these functions cannot be computed by random access machines running in time $t(n)$ and space $s(n)$ if $t(n)s(n) = o(n^2)$.

In **Chapter 3**, which is adapted from [71], we improve upon this idea first by simplifying the lower bound proof, using the same ideas and techniques on a simpler problem. We give an analogous lower bound against the Non-Occurring Elements problem.

Definition 1.0.2 (**Non-Occurring Elements (NOE)**). *Given a list L of n elements each in the set $\{1, ..., n\}$, output a list of each element in the set $\{1, ..., n\}$ which does not occur in L.*

We discuss a stronger model of branching programs that have access to a random oracle, and show matching lower bounds against this model, even in the average case. Lower bounds against this augmented model yield similar results against randomized random access machines, showing that NOE and the above mentioned problems are not computable by randomized random access machines running in time $t(n)$ and space $s(n)$ for any $t(n)$ and $s(n)$ such that $t(n)s(n) = o(n^2)$. In other words, we show that these problems **require a time-space product of at least n^2.**

Definition 1.0.3. *Let Π be a relational problem. We say that* the time-space product of Π is at least $b(n)$ *if for every algorithm running in $t(n)$ time and $s(n)$ space for Π, it must be the case that $t(n) \cdot s(n) \geq \Omega(b(n))$.*

We then give efficient local reductions from NOE to four natural circuit problems, showing analogous lower bounds for each of them. The problems can be described as follows:

- FCircEval: Given a boolean circuit with an arbitrary number of sink nodes, produce the output of each sink node.

- Print-3SAT: Given a 3-CNF boolean formula φ, produce a satisfying assignment to φ or declare that there is not one if one does not exist.

- #2SAT: Given a 2-CNF boolean formula φ, produce the number of satisfying assignments to φ.

- TTPrint: Given a CNF boolean formula φ, produce the truth table of φ. [2]

For each of these problems, we show that there is some constant $k > 0$ such that there is no randomized random access machine computing any of these problems in time $t(n)$ and space $s(n)$ for any $t(n)s(n) \leq O(n^2/\log^k(n))$. For comparison, the best known time lower bound for solving Circuit SAT in $O(\log n)$ space is $n^{2\cos(\pi/7)-o(1)} \geq n^{1.801}$ ([90, 29], building on [44]).

Hardness Magnification for Compression Problems

In Chapter 4, adapted from [70], we show that, not only are "intermediate lower bounds" necessary to show coveted complexity class separations such as $\mathbf{P} \neq \mathbf{NP}$, but some similarly innocuous-looking bounds are *sufficient* for showing such complexity class separations.

We first define a model of streaming algorithms using limited space and update time, which are allowed to query oracles. These are roughly algorithms which may only read the input once and must read their input bits in order. The efficiency of such an algorithm can be measured by their update time – the time taken between reading each bit of the input, as well as their space usage: the number of bits stored by the algorithm. We then give efficient algorithms using oracles in $\mathbf{PH}$ for solving MCSP$[s(n)]$. These algorithms can be viewed as reductions from MCSP$[s(n)]$ to a language $D \in \mathbf{PH}$, and importantly, these reductions have the following property:

[2] If φ has k variables, the output of TTPrint will be length 2^k.

if $D \in \mathbf{P}$, then $\mathsf{MCSP}[s(n)]$ has streaming algorithms with update time and space $\text{poly}(s(n))$. The contrapositive is:

Theorem 1.0.4. *If $\mathsf{MCSP}[s(n)]$ does not have streaming algorithms with update time and space $\text{poly}(s(n))$, then $\mathbf{P} \neq \mathbf{NP}$.*

The real strength of this can be illuminated by the following corollary.

Corollary 1.0.5. *If $\mathsf{MCSP}[n^{10}]$ does not have streaming algorithms with update time and space $\log^{O(1)}(N)$, then $\mathbf{P} \neq \mathbf{NP}$.*

Recalling it is already believed that $\mathsf{MCSP}[n^{10}] \notin \mathbf{P}$, the hypothesis of Corollary 1.0.5 looks extremely plausible. We give similar reductions in the form of oracle circuits from $\mathsf{MCSP}[s(n)]$ and other compression problems to languages in $\mathbf{PH}$ in Theorem 4.1.1. This reduction gives us results similar to the above.

Theorem 1.0.6 (Weakening of Theorem 4.1.4)**.** *Let $s(n) \geq n$.*

- *If there exists an $\varepsilon > 0$ such that for every $c \geq 1$, the problem $\mathsf{MCSP}[2^{\varepsilon n/c}]$ on inputs of length $N = 2^n$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{NP} \not\subset \mathbf{TC}^0$.*

- *If $\mathsf{MCSP}[s(n)]$ on inputs of length $N = 2^n$ does not have circuits of $N \cdot \text{poly}(s(n))$ size and $O(\log N)$ depth, then $\mathbf{NP} \not\subset \mathbf{NC}^1$.*

- *If $\mathsf{MCSP}[s(n)]$ on inputs of length $N = 2^n$ does not have circuits of $N \cdot \text{poly}(s(n))$ size and $\text{poly}(s(n))$ depth, then $\mathbf{NP} \not\subset \mathbf{P}/\text{poly}$.*

Subsequent work has included strengthening some of these results by replacing compression problems (such as MCSP) with arbitrary sparse $\mathbf{NP}$ languages, but this weakening comes at the cost of weaker lower bounds [36]. Informally, a *sparse* language is one which does not have many strings. We examine some of these results in Chapter 5, though the primary purpose of Chapter 5 is to present another relationship between lower bounds.

An Equivalence Between Fixed-Polynomial Circuit Size Lower Bounds and Karp-Lipton Style Theorems

If there is some fixed constant k such that every function f in a class $\mathcal{C}$ can be computed by circuits of size $O(n^k)$, we say $\mathcal{C}$ has *fixed polynomial size* circuits. Likewise, if this is not true of $\mathcal{C}$, we say that $\mathcal{C}$ does not have fixed polynomial size circuits or that $\mathcal{C}$ has *fixed polynomial size circuit lower bounds*.

In [62], Kannan shows that Σ_2^p has fixed polynomial size lower bounds based on the following ideas. **PH** does not have fixed polynomial size circuits, and, known as the Karp-Lipton Theorem,

$$\mathbf{NP} \subset \mathbf{P}/poly \implies \Sigma_2^p = \mathbf{PH}.$$

This is an early example of a proof of intermediate lower bounds, not only with respect to the conjecture that $\mathbf{NP} \not\subset \mathbf{P}/poly$, but also the conjecture that **NP** does not have fixed polynomial size circuits. It also introduces a scheme for proving new fixed polynomial size circuit lower bounds. Letting $\mathcal{C}$ be any class such that $\mathbf{NP} \subseteq \mathcal{C}$, we consider a proposition in the style of the Karp-Lipton Theorem:

$$\mathbf{NP} \subset \mathbf{P}/poly \implies \mathcal{C} = \mathbf{PH}.$$

Showing such a theorem would allow us to conclude in the same way that $\mathcal{C}$ does not have fixed polynomial size circuits! Indeed, lower bounds against $\mathbf{ZPP}^{\mathbf{NP}}$, $\mathbf{S}_2^p$, **PP**, and Promise-**MA** have been demonstrated this way [62, 26, 64, 30, 31, 89, 1, 80, 37].

An interesting question to consider: is this the "only way" to demonstrate fixed polynomial size lower bounds against **NP** or other classes contained in the polynomial hierarchy? With the ultimate goal of showing $\mathbf{NP} \not\subset \mathbf{P}/poly$ in mind, it is a necessary intermediate step to show **NP** does not have fixed polynomial size circuits. We give evidence that the answer to this question is yes!

In Chapter 5, adapted from [37], we show that new fixed polynomial size lower

bounds will actually *require* new theorems in the style of Karp-Lipton and that in fact new fixed polynomial size lower bounds are *equivalent* to new theorems in the style of Karp-Lipton. More precisely, we show that new circuit lower bounds against $\mathbf{P}^{\mathbf{NP}}$ are equivalent to new Karp-Lipton style collapses to $\mathbf{P}^{\mathbf{NP}}$.

Theorem 1.0.7 ($\mathbf{P}^{\mathbf{NP}}$ Circuit Lower Bounds are Equivalent to a Karp-Lipton Collapse to $\mathbf{P}^{\mathbf{NP}}$)**.**
The following two statements are equivalent.

1. *Fixed polynomial size circuit lower bounds against* $\mathbf{P}^{\mathbf{NP}}$*:*

$$\forall k, \mathbf{P}^{\mathbf{NP}} \not\subset \mathbf{SIZE}[n^k]$$

2. *Karp-Lipton Collapse to* $\mathbf{P}^{\mathbf{NP}}$*:*

$$\mathbf{NP} \subset \mathbf{P}/poly \implies \mathbf{PH} \subset i.o.\text{-}\mathbf{P}^{\mathbf{NP}}/n$$

In order to understand Theorem 1.0.7, we need to understand the claim $\mathbf{PH} \subset$ i.o.-$\mathbf{P}^{\mathbf{NP}}/n$. We give the definitions needed here in Chapter 5, but for now, an informal unpacking of this statement should suffice. There are really two important ideas to be considered.

- **Advice:** $\mathbf{P}^{\mathbf{NP}}/n$ denotes the class of languages decidable by *polynomial time machines with an* $\mathbf{NP}$ *oracle and* n *bits of advice.* That is to say that, for each input length n, there is some string a of n bits such that given additional access to this string causes our $\mathbf{P}^{\mathbf{NP}}$ machine to behave correctly. Note that this string a should only depend on the input length n rather than the input.

- **Infinitely Often:** We say a function $f \in$ i.o.-$\mathcal{C}$ if there is function $f' \in \mathcal{C}$ which agrees with f on *infinitely many* input lengths. This means that for each input x of these input lengths, $f(x) = f'(x)$. In the case of $f \in$ i.o.-$\mathbf{P}^{\mathbf{NP}}/n$, this is equivalent to saying that there is some polynomial time machine M with access

to an **NP** oracle and n bits of advice which, on infinitely many input lengths, decides f correctly.

We additionally make progress towards the ultimate goal of showing that "circuit lower bounds against **NP** are equivalent to Karp-Lipton style collapses to **NP**." However, we have not yet reached this goal. In the following, think of "r.o" and "c-r.o." as stronger versions of the notion "infinitely often." We define these notions formally in Section 5.2.

Theorem 1.0.8 (Paraphrased from Theorem 5.1.1)**.** *Assuming* $\mathbf{NP} \subset \mathbf{P}/poly$*, the following are equivalent:*

1. $\mathbf{NP}$ *is not in* $\mathbf{SIZE}[n^k]$ *for all* k*.*

2. $\mathbf{AM}/1$ *is in* c*-r.o.-*$\mathbf{NP}/n^{\varepsilon}$ *for all* $\varepsilon > 0$ *and integers* c*.*[3]
   That is, Arthur-Merlin games with $O(1)$ *rounds and small advice can be simulated "c-robustly often" in* **NP** *with modest advice, for all constants* c*.*

3. **NP** *does not have* n^k*-size witnesses for all* k*.*
   That is, for all k*, there is a language* $L \in \mathbf{NP}$*, a poly-time verifier* V *for* L*, and infinitely many* $x_n \in L$ *such that* $V(x_n, \cdot)$ *has no witness of circuit complexity at most* n^k*.*

4. $\mathbf{NP}$ *is not in* $\mathbf{AMTIME}(n^k)$ *for all* k*.*
   That is, for all k*, there is a language in* **NP** *which cannot be decided by an Arthur-Merlin protocol running in* $O(n^k)$ *time.*

5. $(\mathbf{NP} \cap \mathbf{coNP})/n^{\varepsilon}$ *is not in* $\mathbf{SIZE}[n^k]$ *for all* k *and all* $\varepsilon > 0$*.*

6. $(\mathbf{AM} \cap \mathbf{coAM})/1$ *is in* c*-r.o.-*$(\mathbf{NP} \cap \mathbf{coNP})/n^{\varepsilon}$ *for all* $\varepsilon > 0$ *and all integers* c*.*

The consequence of the following corollary of Theorem 1.0.8 can be thought of as a weak Karp-Lipton style collapse of **AM** to **NP**.

[3] See the Preliminaries for a definition of "c-robustly often". Intuitively, it is a mild strengthening of "infinitely often".

Corollary 1.0.9 (**NP** Circuit Lower Bounds Equivalent to a Karp-Lipton Collapse of **AM** to **NP**). **NP** $\not\subset$ **SIZE**$[n^k]$ *for all* k *if and only if (***NP** $\subset$ **P**$/poly \implies$ **AM** *is in r.o.-c-***NP**$/n^{\varepsilon}$ *for all* c*).*

Finally, we conclude Chapter 5 with additional hardness magnification results similar to those in Chapter 4. These results generalize hardness magnification from *compression problems* to arbitrary *sparse languages in* **NP**.

Definition 1.0.10. *A language* L *is* $s(n)$*-sparse if, for all* n, $|L \cap \{0,1\}^n| \leq s(n)$.

Notice that sparse languages are a proper generalization of compression problems (or at least the decision versions of compression problems) in the following sense. If L contains only strings which can be compressed to $s(n)$ bits, then L must be $2^{s(n)}$-sparse. Our prime example is of course $\mathsf{MCSP}[s(n)]$ which is $2^{O(s(n)\log(n))}$-sparse. Now consider Theorem 1.0.11.

Theorem 1.0.11. **NEXP** $\not\subset$ **P**$/poly$ *if and only if there exists an* $\varepsilon > 0$ *such that for every sufficiently small* $\beta > 0$*, there is a* $2^{n^{\beta}}$*-sparse language* $L \in$ **NTIME**$[2^{n^{\beta}}]$ *without* $n^{1+\varepsilon}$*-size circuits.*

Aside from the generalization to sparse languages, there are some differences worth noting between Theorems 1.0.6 and 1.0.11. Consider the following conjecture.

Conjecture 1.0.12. $\mathsf{MCSP}[n^{10}]$ *does not have circuits of size* $n^{1.0000001}$.

If we managed to prove Conjecture 1.0.12, we could use Theorem 1.0.6 to conclude **NP** $\not\subset$ **P**$/poly$, while we could only use Theorem 1.0.11 to conclude **NEXP** $\not\subset$ **P**$/poly$. At the moment, it seems that generalization has come at the cost of the strength of the lower bounds in the consequent of Theorem 1.0.11.

However, another important difference is that Theorem 1.0.11 is an *equivalence*! As is a common theme in Chapter 5, this shows us that proving lower bounds against some sparse **NP** languages is inescapable in our quest to prove **NEXP** $\not\subset$ **P**$/poly$.

Chapter 2

Preliminaries

In this chapter, we present some basic preliminaries which are common themes in the following chapters. Each chapter will still have its own preliminary section, some of which may be redundant with this chapter.

Throughout this thesis, we assume basic familiarity with computational complexity [13]. Maturity with ideas surrounding circuits and non-uniform computation, oracles, randomness and pseudo-randomness, and of course reductions will be essential for understanding this work.

Background from [13]. From Arora and Barak [13], Chapters 1 through 9 are crucial. Of the later chapters in the book, Chapter 22: *Why are circuit lowerbounds so difficult?* is likely the most important for our work. A reader of this thesis would be further aided by comfort with any additional material, but especially the material from Chapters 10, 12, 13, 15, and 16.

Relational Problems. Essentially every problem in this thesis is a *relational problem*. By relational problem, we simply mean that for a relation $R \subseteq \Sigma^* \times \Sigma^*$, we will be given some input x, and we will output some y such that $(x, y) \in R$ **or** give an output which indicates that there is no such y. This exact way this is handled may vary between contexts and will always be defined as part of the problem. If no alphabet Σ is given, we assume the alphabet is $\{0, 1\}$. In this thesis, many of the problems we will consider can be further placed into one or both of the following categories:

- *Decision problems*, *boolean functions*, or *languages*. These are extremely common notions, and if the reader is not familiar with these, they would be advised to revisit this thesis once they have acquired more maturity with these ideas. These are simply problems for which the solution is always 0 or 1 (or some other pair of symbols or strings denoting "yes" and "no"). Usually, when we refer to a problem L as a language, we mean to describe L as the set of strings which merit the "yes" answer. That is,

$$L(x) = 1 \iff x \in L.$$

- *Function problems* or *functions*. These are problems which, like decision problems, always have exactly one correct response for an input, but this input is not necessarily restricted to being from some binary set. A decision problem is trivially a function.

SAT is a great example of a well-known decision problem, and similarly, #SAT is an example of a function problem. As these are very common categorizations of problems, our purpose in delineating these categories is to point out examples of problems in this thesis which are neither decision problems nor function problems. Recall NOE from Chapter 1.

Reminder of Definition 1.0.2 Non-Occurring Elements (NOE). *Given a list L of n elements each in the set $\{1, ..., n\}$, output a list of each element in the set $\{1, ..., n\}$ which does not occur in L.*

Given a list L, NOE(L) does not necessarily refer to one specific list. In fact, many different lists may be satisfying responses from a machine computing NOE. Because of this, we only use the notation NOE(L) sparingly, and only when *any* valid list of the non-occurring elements of L would suffice. Moreover, we will only use NOE(L) to make claims about the set of non-occurring elements of L.

Another important relational problem appears in Chapter 4.

Definition 2.0.1 (search-MCSP$[s(n)]$)**.** *Given a truth table T, do one of the following:*

- *Output a circuit C of size at most $s(n)$ such that* $\mathsf{TruthTable}(C) = T$,[1] **or**
- *If there is no such circuit, report that there is no circuit C of size at most $s(n)$ such that* $\mathsf{TruthTable}(C) = T$.

Just like in the case of NOE, when computing search-MCSP$[s(n)]$, we are satisfied with a wide variety of answers for some inputs, but we are also satisfied with the answer that there is no such circuit. This will be a common theme with problems in Chapter 4.

Restrictions to n input bits. For any problem Π, unless otherwise denoted, Π_n is the n-bit boolean function which agrees Π on all inputs of length n.

Functions for truth tables and circuit complexity. Given a circuit or formula C deciding some n-bit boolean function, it will be convenient to be able to refer to the truth table of C as $\mathsf{TruthTable}(C)$. More specifically, $\mathsf{TruthTable}(C) \in \{0,1\}^{2^n}$ is the evaluation of C on all possible inputs sorted in lexicographical order.

For every string y, let 2^ℓ be the smallest power of 2 such that $2^\ell > |y|$. We define the circuit complexity of y, denoted as $\mathsf{CircuitComplexity}(y)$, to be the circuit complexity of the ℓ-input function defined by the truth-table $y10^{2^\ell - |y| - 1}$.

Similarly, given n-bit function Π_n, we will denote the size of the smallest circuit computing Π_n as $\mathsf{SIZE}(\Pi_n)$. For functions Π with unrestricted input lengths, we define $\mathsf{SIZE}(\Pi)(n) = \mathsf{SIZE}(\Pi_n)$.

Circuit complexity classes. We assume basic familiarity circuit complexity classes such as $\mathbf{AC}^0$, $\mathbf{ACC}^0$, $\mathbf{TC}^0$, and $\mathbf{NC}^1$. We assume these circuit classes are *non-uniform* unless otherwise specified. An **AC circuit family** is any circuit family over the basis of NOT, unbounded fan-in OR, and unbounded fan-in AND. For $s : \mathbb{N} \to \mathbb{N}$, $\mathbf{SIZE}[s(n)]$ is the class of languages decided by an infinite circuit family where the nth circuit in the family has size at most $s(n)$.

"Parity P". $\oplus\mathbf{P}$ is the closure under polynomial-time reductions of the decision problem Parity-SAT: *Given a Boolean formula, is the number of its satisfying assignments odd?*

[1] $\mathsf{TruthTable}(C)$ is simply the truth table of C. There is a more formal definition given later in this section.

Advice. For a deterministic or nondeterministic class $\mathcal{C}$ and function $a(n)$, $\mathcal{C}/a(n)$ is the class of languages L such that there is an $L' \in \mathcal{C}$ and function $f : \mathbb{N} \to \{0,1\}^\star$ with $|f(n)| \leq a(n)$ for all x, such that $L = \{x \mid (x, f(|x|)) \in L'\}$. That is, the advice string $f(n)$ can be used to solve all n-bit instances within class $\mathcal{C}$. For "promise" classes $\mathcal{C}$ such as **MA** and **AM**, $\mathcal{C}/a(n)$ is defined similarly, except that the promise of the class is only required to hold when the correct advice $f(n)$ is provided.

Time-space lower bounds. Throughout this thesis, we will discuss how much time or memory a machine must use when the other is restricted. For example, we may wonder how much space a machine running in $n^{1.5}$ time needs to compute NOE.

Reminder of Definition 1.0.3. *Let Π be a relational problem. We say that* the time-space product of Π is at least $b(n)$ *if for every algorithm running in $t(n)$ time and $s(n)$ space for Π, it must be the case that $t(n) \cdot s(n) \geq \Omega(b(n))$.*

At times, we may instead say that a problem Π *requires a time-space product* or *has a time-space lower bound* of $b(n)$, though the meaning is not changed.

Chapter 3

Nearly Quadratic Time-Space Lower Bounds Against Natural Problems

3.1 Introduction

Infamously little progress has been made towards the resolution of **P** vs **NP**. It is still open whether there are linear time algorithms for solving generic **NP**-hard problems such as Circuit SAT although for certain **NP**-hard problems, non-linear lower bounds on multitape Turing machines are known [78, 47]. In fact it remains open whether Circuit SAT has a generic algorithm in a random-access machine model running in $O(n^{1.9999})$ time and $O(\log n)$ additional space beyond the input.[1] Currently the best known time lower bound for solving Circuit SAT in $O(\log n)$ space is $n^{2\cos(\pi/7)-o(1)} \geq n^{1.801}$ ([90, 29], building on [44]).

Other prominent work has used combinatorial methods to prove time-space lower bounds for decision problems in **P** [22, 5, 24, 23, 82, 6, 77]. For general random-access models of computation, modest super-linear (e.g. $n \log n$-type) time lower bounds for explicit problems in **P** are known when the space is restricted to $n^{.99}$ or less (for randomized models as well [23, 77]). Thus for such problems, the time-space product

[1] Note that for multitape Turing machines, such lower bounds are not hard to show [80]. However, these lower bounds rely on the sequential access of Turing machines on tapes; once random access is allowed, these lower bounds break.

can be lower-bounded to nearly $\Omega(n^2)$ when the space is close to n; however, when the space is $O(\log n)$, the best time lower bound against RAMs appears to be the aforementioned $n^{1.801}$ bound for Circuit SAT.

All the above cited lower bounds are for *decision* problems. For example, the SAT lower bound applies to algorithms which are only required to determine if a given formula is satisfiable or not. There are several well-studied extensions of SAT which are *function* problems, outputting multiple bits:

1. **How difficult is it to *print* a satisfying assignment, when one exists?** For 3CNF formulas, call this problem Print-3SAT. Of course, Print-3SAT has a polynomial-time algorithm if and only if $\mathbf{P} = \mathbf{NP}$, but perhaps it is easier to prove concrete lower bounds for it, compared to the decision version.

2. **How difficult is #2SAT, in which we *count* the number of satisfying assignments to a 2CNF?** Since #2SAT is #**P**-complete, the problem should intuitively be harder to solve than 3SAT.

3. **How difficult is it to print the truth table of a CNF?** Given a CNF formula F of n variables and up to 2^n size, F can be evaluated on all possible 2^n inputs in $4^n \cdot \text{poly}(n)$ time and $\text{poly}(n)$ space (using a linear-time, log-space algorithm for evaluating a CNF on a given input). Call this problem TTPrint (for truth-table printing). **Is this quadratic running time optimal?** This question is of considerable interest for SAT algorithms; for some circuit classes, the only known SAT algorithms beating exhaustive search (e.g., [91]) proceed by reducing SAT to a quick truth table evaluation.

In this chapter, we prove that for essentially any generic randomized computation model using $n^{o(1)}$ space but having constant-time access to a random oracle (i.e., a random string of $2^{n^{o(1)}}$ bits), the above three problems all require nearly quadratic time to compute with nonzero constant probability. We revisit a quadratic time-space lower bound of Beame [20] for the Unique Elements problem, show that his technique can be used to prove an analogous lower bound for an even "simpler" problem that we call

Non-Occurring Elements, extend the lower bound's reach to include random oracles, and give succinct reductions from Non-Occurring Elements to the above problems, proving hardness for them.

Lower Bound for Non-Occurring Elements. Recall that in Chapter 1, we defined Non-Occurring Elements. Recall as well that $[n] := \{1, \ldots, n\}$.

Reminder of Definition 1.0.2 (Non-Occurring Elements (NOE)). *Given a list L of n elements each in the set $[n]$, output a list in any order containing each element in the set $[n]$ which does not occur in L.*

Observe it is easy to get a space-$O(s(n))$ algorithm on the $\log(n)$-word RAM for computing NOE with running time $O(n^2/s(n))$:

> Partition $[n]$ into $n/s(n)$ blocks of $s(n)$ elements each. Do $n/s(n)$ passes over L, where in the ith pass, check which numbers in the ith block do not occur in L: this can be done in $s(n)$ bits of space by simply keeping a bit vector.

Hence we would say NOE has *time-space product* $O(n^2)$ for all $n \leq t(n) \leq n^2$.

Our first main theorem builds on the aforementioned work of Beame to show that this time-space product is optimal, even when programs have access to a random oracle and can err with high probability.

Theorem 3.1.1. *For all $p \in (0, 1]$, every random oracle n-way branching program family of size $2^{s(n)}$ and height $t(n)$ computing NOE with success probability p has $t(n) \cdot s(n) \geq \Omega(n^2)$ for all sufficiently large n.*

(Note, we need to formally define what a "random oracle branching program" even means. For now, think of it as a non-uniform version of space-bounded computation with constant-time access to a uniform random $2^{s(n)}$-bit string. The preliminaries in Section 3.2 give full detailed definitions.) Using a standard translation of word RAMs into branching programs, it follows that every probabilistic word RAM with a random oracle and wordsize $\log(n)$ computing NOE in time $t(n)$ and space $s(n)$ must have $t(n) \cdot s(n) \geq \Omega(n^2)$ in order to have constant success probability.

Lower Bound for Sorting and Circuit Evaluation. Informally, we say that a reduction from a problem A to a problem B is *succinct* if any particular bit of the reduction can be computed in $\text{poly}(\log n)$ time. (Definitions can be found in the Preliminaries.) As a warm-up, in Theorem 3.3.8 of this chapter, we use a simple succinct reduction to obtain an analogous random oracle lower bound for the problem of sorting n unordered elements from $[n]$, which we call Sort. (A tight lower bound without random oracles was proved by Beame [20].) We combine the sorting lower bound with another simple reduction to obtain an analogous lower bound for evaluating circuits. Let FCircEval be the problem: *given a circuit of size n with at most n inputs (all fixed to 0-1 values) and at most n outputs, determine its output.* The decision version of FCircEval is well-known to be **P**-complete.

Theorem 3.1.2. *For all finite Σ, there is a $k > 0$ such that the random oracle Σ-way branching program time-space product of FCircEval is at least $\Omega(n^2/\log^k n)$.*

As in the case of NOE, Theorem 3.1.2 implies analogous time-space lower bounds on random access machines computing FCircEval. We note that without the "random oracle" modifier, Theorem 3.1.2 is almost certainly folklore; it follows from our simple reduction and the $\tilde{\Omega}(n^2)$ time-space product lower bound on sorting of Borodin and Cook [25].

Lower Bound for Printing SAT Assignments. Any practical algorithm for Satisfiability would need to print satisfying assignments when they exist. We give a succinct reduction from Circuit Evaluation to the printing problem for 3SAT, proving a nearly-quadratic time lower bound in the $n^{o(1)}$ space setting.

Theorem 3.1.3. *For all finite Σ, there is a $k > 0$ such that the random oracle Σ-way branching program time-space product of Print-3SAT is at least $\Omega(n^2/\log^k n)$.*

Lower Bound for Computing Truth Tables of CNFs. We show that the trivial algorithm for computing the truth table of a large CNF has optimal time-space product, even for randomized algorithms.

Theorem 3.1.4. *For all finite Σ, there is a $k > 0$ such that the random oracle Σ-way branching program time-space product of* TTPrint *for CNF formulas with n clauses and $\log(n) + \log\log(n)$ many variables is at least $\Omega(n^2/\log^k n)$.*

The proof of Theorem 3.1.4 works by giving a succinct reduction from NOE to TTPrint, constructing a CNF whose truth table encodes (at the bit level) the non-occurring elements of a given list.

Lower Bound for #2SAT. Finally, we give a succinct reduction from the truth table problem TTPrint for CNF formulas to *counting* SAT assignments to a given 2CNF, implying an analogous lower bound for #2SAT. In particular, we encode the output 2CNF in such a way that the bits of its number of satisfying assignments encode the *value* of the original CNF on various inputs.

Theorem 3.1.5. *For all finite Σ, there is a $k > 0$ such that the random oracle Σ-way branching program time-space product of #2SAT is at least $\Omega(n^2/\log^k(n))$.*

Lower Bounds Beyond? Finally, we note that the random oracle model we consider may not be the most general possible one. We could also consider oracles where the queries are on write-only storage that does not count towards the space bound. Although such oracles are sometimes used in space-bounded complexity ([65]), it is hard for us to see how such queries could help in the *random* oracle setting. In this chapter's conclusion (Section 3.5) we outline how to coherently define this sort of oracle access in branching programs, and conjecture that our lower bounds also hold in this "extended oracle" model.

3.1.1 Intuition for the NOE Lower Bound

Our proof of the NOE lower bound (Theorem 3.1.1) starts from Beame's lower bound for Unique Elements [20]. We generalize his proof, and use our generalization to show that any function problem[2] satisfying two basic properties has a non-trivial branching

[2]This also applies to *relational problems*, such as NOE, though for simplicity only describe the criteria for function problems. Notice that we can treat NOE as a function which outputs a set for the purpose of Theorem 3.1.6.

program lower bound, even when the branching program has access to a huge number of random bits.

Theorem 3.1.6. *Let $\{\Sigma_n\}$ be a family of finite alphabets, $\{f_n : (\Sigma_n)^n \to (\Sigma_n)^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions where each $_n$ is over strings in $(\Sigma_n)^n$, and let $g : \mathbb{N} \to \mathbb{N}$ satisfy the following properties.*

1. **[f typically has "long" outputs].** *For all $\varepsilon > 0$, there is an $n_0 \geq 0$ such that for all $n > n_0$, there is a $\delta > 0$ such that*

$$\Pr_{x \in D_n} [|f_n(x)| > \delta g(n)] > 1 - \varepsilon.$$

2. **[Short random-oracle branching programs have low probability of printing long substrings of f].** *Let U_n be the uniform distribution over Σ_n and let $N \leq 2^{s(n)}$ be an integer. There is an $\varepsilon > 0$ such that for all Σ_n-way branching programs P of height at most $n/4$,*

$$\Pr_{(x,r) \sim D_n \times U_n^N} [(P(xr) \text{ is a substring of } f_n(x)) \wedge (|P(xr)| \geq m)] < e^{-\varepsilon m}.$$

Then, for all $n > n_0$ and $p \in (0, 1]$, and for every random oracle Σ_n-way branching program of size $2^{s(n)}$ and height $t(n)$ computing f_n with success probability at least p on inputs drawn from D_n, it must be that $t(n) \cdot s(n) \geq \Omega(n \cdot g(n))$.

The intuition behind Theorem 3.1.6 is that, if a function f on input x requires a long output (Property 1), then some (possibly many) "subprograms" of an efficient branching program P computing f will need to produce somewhat-long output. But by Property 2, this is "hard" for all short subprograms even with a random oracle: they have low probability of correctly answering a large fraction of f.

Beame's original lower bound for Unique Elements [20] follows a similar high-level pattern (as we review in Theorem 3.3.1). One of our insights is that the Non-Occurring Elements problem satisfies stronger versions of the properties used in his proof for Unique Elements; these stronger properties give us extra room to play with

the computational model in our lower bound against Non-Occurring Elements. Another insight is that the conditioning on uniform random input in his argument can be modified to accommodate very long auxiliary random inputs. Together, this allows us to extend the lower bounds to models equipped with random oracles.

Why Random Oracles? Let us give one comment on the model itself. One may wonder why it is even necessary to add random oracles to branching programs (BPs), which are a *non-uniform* model of computation. Can't we use Adleman's argument [13] to show that the randomness can be hard-coded in the non-uniform model, similarly to how $\mathbf{BPP} \subset \mathbf{P}/poly$?

Indeed, a BP with a random oracle can always be simulated by a deterministic BP; however, the running time of the deterministic BP increases by a multiplicative factor of $\Omega(n)$ (hence, a time lower bound of $\Theta(n^2)$ on a deterministic BP says nothing *a priori* about randomized BPs). Given a randomized BP for a decision problem with constant success probability $1/2 < p < 1$, it is derandomized by taking $\Omega(n)$ copies of the BP (with independent random bits filled in each copy), and computing the majority value of the BP outputs. The $\Omega(n)$ copies are needed because one needs to guarantee correctness over all 2^n possible inputs: this increases the running time by an $\Omega(n)$ multiplicative factor. In our case, the situation is even more complicated because we are concerned with function problems. The upshot is that n^2 lower bounds on random-oracle BPs give strictly more information than a typical randomized model with one-way access to random bits, or a deterministic model.

3.2 Preliminaries

We assume basic familiarity with computational complexity [13]. Here we recall (and introduce) various computational models and notations needed in this chapter.

All of our lower bounds are on the product of time and space for various problems. For clarity, we define the time-space product as follows.

Reminder of 1.0.3. *Let* Π *be a relational problem. We say that* the time-space

product of Π is at least $b(n)$ *if for every algorithm running in* $t(n)$ *time and* $s(n)$ *space for* Π*, it must be the case that* $t(n) \cdot s(n) \geq \Omega(b(n))$.

Random Access Machines With Oracles. For this work, we consider random access machines (RAMs) with access to an oracle and a write-only output tape. More precisely, our RAMs can only append new characters to the end of their output tape.

Definition 3.2.1. *Let* O *be a language over a finite alphabet. An* **oracle RAM** M^O *is a random access machine with an additional* oracle tape*. During a time step, in addition to any normal RAM operations, an oracle RAM can read or write a character to the oracle tape, can move the oracle tape head left or right, or can query the oracle with the contents of the tape to obtain a uniform random bit. Additionally,* M^O *has a write-only output tape to which it can append a character in any step.*

R-Way Branching Programs With Oracles and Output. Our lower bounds for Non-Occurring Elements (Theorem 3.1.1) will be against a generalization of branching programs with a (large) alphabet of size R, often called R-way branching programs [25]. We recall the definition here.

Definition 3.2.2. *Let* $R \geq 2$ *be an integer. An* R**-way branching program** *with* n *inputs* $x_1, \ldots, x_n$ *as a directed acyclic graph with one source node in which every non-sink node has out-degree* R*. Every non-sink node is labeled with an index* $i \in [n]$ *corresponding to an input variable, and each edge is labeled with an element of* $[R]$ *such that no edge* (u, v) *and* (u, w) *where* $v \neq w$ *share a label. Additionally, each vertex* v *is labeled with an instruction to print a value* $p(v)$ *(which may be the empty string). The* **height** *of an* R*-way branching program* P *is the length of the longest path in its graph, and the* **size** *is the number of vertices. The* **computation path of** P **on input** x *is the unique path* $\pi = (v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ *such that* v_0 *is the source node,* v_k *is a sink node, and for all* $i \in \{0, \ldots, k-1\}$*, if vertex* v_i *is labeled* j*, then the label on* (v_i, v_{i+1}) *is the value of* x_j*. The* **output of** P **on input** x*, denoted as* $P(x)$*, is the concatenation of the values printed by the vertices along the computation path of* P *on input* x*, i.e.,* $p(v_0)p(v_1)\cdots p(v_k)$.

Let Σ be a finite alphabet. For notational convenience, we call a P a Σ-way branching program if P is a $|\Sigma|$-way branching program with edges labeled with elements of Σ instead of $[|\Sigma|]$.

Definition 3.2.3. *A function $f : \Sigma^* \to \Sigma^*$ has a* **Σ-way branching program of height $t(n)$ and size $S(n)$** *if for all $n \geq 0$, there is a Σ-way branching program P_n of height $t(n)$ and size $S(n)$ such that for all $x \in \Sigma^n$, $P_n(x) = f(x)$.*

We study a natural generalization of R-way branching programs with oracles whose queries count towards the space bound. This is a natural set-up for the random oracle setting, where random oracles model truly random hash functions. Let $O : \Sigma^\star \to \{0, 1\}$ in the following.

Definition 3.2.4. *An* **O-oracle R-way branching program** *with n inputs is an R-way branching program with the following additional properties. Each vertex is labeled with either a* variable index *$i \in [n]$ or with a string $q_j \in \{0, 1\}^\star$ (where j is an integer ranging from 1 to the number of vertices in the program); we call the latter* Q-vertices *(for "Query"). All Q-vertices have two outgoing arcs, labeled with the two possible query answers* yes *or* no *(i.e., 1 or 0). Computation on an O-oracle branching program is defined analogously to usual branching programs, with the following extra rule for Q-vertices. Each time a Q-vertex v is reached during a computation, the outgoing* yes *(i.e., 1) edge of v is taken in the computation path if and only if the label q_j of v satisfies $O(q_j) = 1$.*

Remark 3.2.5. *Note that in the above definition, the oracle queries could be strings of* arbitrary *length, but in a size-S branching program we are only allowed S possible distinct oracle queries over all possible inputs. So the above definition is more general than the condition that "oracle queries count towards the space bound". In the random oracle setting, this distinction will make little difference; we might as well think of the query strings as being of length about $\log_2(S)$.*

It is natural to augment branching programs with oracles. Barrington and McKenzie [17], motivated by an approach to proving $\mathbf{NC}^1 \neq \mathbf{P}$, defined an oracle branching

program model in terms of finite automata, where instead of branching on individual input bits, the program can branch on (in principle) all possible n-bit inputs, corresponding to oracle queries on the input. The usual branching program model is captured in their model by a branching program with an oracle for the predicate $BIT(x, i)$ which returns the ith bit of the input x. They proved exponential size lower bounds for branching programs with certain weak oracles. Our oracle model is designed to give a natural correspondence between O-oracle R-way branching programs and word RAMs with wordsize $\log(R)$ and oracle access to O.

It will be helpful to think of oracle branching programs as simply branching programs *which receive the oracle as part of their input.* The following proposition formalizes this:

Proposition 3.2.6. *Let P^O be an O-oracle Σ-way branching program with n inputs, height T, and size S, and let $q_1, ..., q_m$ be the set of all possible oracle queries appearing at Q-vertices of P^O. There is a Σ-way branching program P' of height T and size S such that for all $x \in \Sigma^n$, $P^O(x) = P'(xy)$, where $y = O(q_1), ..., O(q_m)$.*

Proof. The branching program P' is simply a relabeling of P in which every Q-vertex with label q_j is instead labeled by the variable index $n + j$. □

Random Oracles. We will ultimately study R-way branching programs with random oracles. All of our random oracles will have the form $O : \Sigma^* \to \Sigma$ for a finite alphabet Σ. Thus we define random oracles as a random elements of the set $(\Sigma^* \to \Sigma)$.

Definition 3.2.7. *$\mathcal{D}_\Sigma$ is the uniform distribution over functions in $(\Sigma^* \to \Sigma)$. That is, drawing some $f \sim \mathcal{D}_\Sigma$ is equivalent to, for each $x \in \Sigma^*$, choosing an element of Σ uniformly at random as $f(x)$.*

We define computation with a random oracle branching program as follows.

Definition 3.2.8. *A function f has a* **random oracle branching program family of height $t(n)$ and size $2^{s(n)}$ with success probability** $p \in (0, 1]$ *if there is a family of oracle branching programs $\{P_n^O\}$ such that the height of each P_n^O is at most $t(n)$,*

the size of each P_n^O *is at most* $2^{s(n)}$*, and for all inputs* x*,*

$$\Pr_{O \sim \mathcal{D}_\Sigma} \left[P_{|x|}^O(x) = f(x) \right] \geq p.$$

Borodin and Cook show that R-way branching programs of height $O(t(n))$ and size $2^{O(s(n))}$ can simulate time-$t(n)$ space-$s(n)$ RAMs of wordsize $\lfloor \log_2 R \rfloor$ [25]. We observe that their proof relativizes to allow oracle R-way branching programs to simulate oracle RAMs with wordsize $\lfloor \log_2 R \rfloor$.

Lemma 3.2.9. *For every language* O *and oracle RAM* M^O *running in time* $t(n) \leq 2^{O(s(n))}$ *and space* $s(n) \geq \log(n)$ *with wordsize* $\log(\Sigma)$*, there is an oracle* Σ*-way branching program of height* $O(t(n))$ *and size* $2^{O(s(n))}$ *such that* $M^O(x) = P^O(x)$ *for all inputs* x*.*

Proof. Without loss of generality, assume that at any step, M^O either accesses its input of length n by writing an index $j \in [n]$ to an "access register", or M^O queries O by writing the character Q to the access register. Let the configuration of machine M contain the description of the random access memory of M, the state of M, and the character currently being output, if any. For integer $n \geq 1$, define a graph $G_n = (V_n, E_n)$ as follows. Let V_n be the set of triples (C, i, j) such that C is a possible space-$s(n)$ configuration of M, $i \in \{0, 1, \ldots, t(n)\}$, and $j \in [n] \cup \{Q\}$.

Our corresponding branching program, has each vertex (C, i, j) with $j \in [n]$ labeled by j, and those (C, i, j) with $j = Q$ are Q-vertices, which we label with a string q_j representing the content of the oracle tape of M^O in configuration C.

For $j \in [n]$, we put the edge $((C_1, i, j), (C_2, i + 1, j')) \in E_n$ and label it with a string p if and only if M^O in configuration C_1 would transition to configuration C_2 given that in configuration C_1, j is put in the access register, $x_j = p$, and j' is put in the access register in C_2. For configurations making an oracle query, put $((C_1, i, Q), (C_2, i + 1, j')) \in E_n$ and label it with $b \in \{0, 1\}$ if and only if M^O in configuration C_1 would transition to configuration C_2 given that in configuration C_1, Q is written on the access register in C_1, the oracle query O returns b, and j' is written on access register in C_2.

Observe that $|V_n| \leq 2^{O(s(n))} \cdot t(n) \cdot n \leq 2^{O(s(n)}$, assuming the size of the tape alphabet and the number of states in the state machine of M^O are constants. Further observe that all paths in G_n have length at most $t(n)$, since every step in the path must be from some (C, i, j, k) to some $(C', i+1, j', k')$ and $i, i+1 \in [t(n)]$. Finally, the oracle branching program P_n^O is defined to be the Σ-way branching program which is the induced subgraph of G_n containing all vertices reachable from $v_0 = (C_0, 0, j_0)$ (with the labels prescribed above), where C_0 is the initial configuration of M^O, and j_0 is the index of the input read in the initial configuration. We see that by construction, for all $x \in \Sigma^n$, $P_n^O(x) = M^O(x)$. □

Lemma 3.2.9 immediately implies that R-way branching program lower bounds provide analogous RAM lower bounds:

Proposition 3.2.10. *Let $f : \Sigma^* \to \Sigma^*$. Suppose there is no R-way branching program family of height $O(t(n))$ and size $2^{O(s(n))}$ computing f. Then there is no RAM of wordsize $\log(R)$ running in time $t(n)$ with space $s(n)$ computing f.*
Furthermore, if there is no random oracle R-way branching program family of height $O(t(n))$ and size $2^{O(s(n))}$ computing f with success probability p, then there is no oracle RAM M^O with wordsize $\log(R)$ such that for all $x \in \Sigma^\star$, $\Pr_{O \sim \mathcal{D}_\Sigma}[M^O(x) = f(x)] \geq p$.

Succinct Reductions. After our lower bound for NOE has been proved, we can apply very efficient reductions from NOE to extend the lower bound to other problems. For this purpose, we need some notation. In what follows, let f, g, and π be functions from Σ^* to Σ^*.

Definition 3.2.11. *We say A has a $t(n)$-time reduction with blowup $b(n)$ to B if there is a many-one reduction f reducing A to B such that $|f(x)| \leq b(|x|)$, and any character in the string $f(x)$ can be computed by an algorithm running in time $t(|x|)$, given the index $i = 1, \ldots, |f(x)|$ of the character and random access to x.*

Polylog-time reductions with quasi-linear blowup essentially preserve lower bounds for our branching program model, up to polylog factors. The proof is relatively straightforward.

Lemma 3.2.12. *Let π be an $O(\log^k(n))$-time reduction with blowup $b(n)$ from f to g, and suppose g has an oracle branching program family $\{P_n^O\}$ of height $t(n)$ and size $2^{s(n)}$ with success probability $p > 0$. Then f has an oracle branching program family $\{Q_n^O\}$ of height $t(b(n))\log^k(n)$ and size $2^{O(s(b(n))+\log^k(n))}$ with success probability at least p.*

Proof. Suppose π is an $O(\log^k(n))$-time reduction with blowup $b(n) \leq O(n\log^k(n))$ from f to g witnessed by a RAM R, and there is an oracle branching program family $\{P_n^O\}$ and p such that for all x, given a random oracle O, $\Pr_O[P_{|x|}(x) = g(x)] \geq p$. By Lemma 3.2.9, there is a branching program family R_n of height $O(\log^k(n))$ and size $2^{O(\log^k(n))}$ which computes the i-th character of $\pi(x)$ given x and i, where $|x| = n$. For ease of notation, for any fixed i, we denote the branching program $R_n(x, i)$ as $R_{n,i}(x)$. Note that $R_{n,i}(x)$ has one output character in every computation path.

We now describe an oracle branching program family Q_n^O of size $2^{O(s(b(n))+\log^k(n))}$ and height $t(b(n)) \cdot \log^k(n)$ such that for all x, $\Pr_O[Q_{|x|}(x) = f(x)] \geq p$. To construct Q_n^O, we compose P_n^O and R_n in the natural way. Start with the program $P_{b(n)}^O$, with the intention of using $P_{b(n)}^O$ to compute $g(\pi(x))$. To do this, for each vertex v in $P_{b(n)}$ which would query the i-th character y_i from $\pi(x)$, we replace v with an instance of $R_{n,i}$ (which makes queries to x and prints y_i). However, we modify the instance of $R_{n,i}$ so that, at every node w where a character $\sigma \in \Sigma$ would be output, we instead put an edge from w to the vertex u in $P_{b(n)}^O$ such that (v, u) was an edge labeled σ in $P_{b(n)}^O$. Finally, if the vertex v had an instruction to output a character τ, the source node of the instance of $R_{n,i}$ replacing v now outputs τ instead.

Observe that Q_n has the height of $P_{b(n)}$ with an additional $\log^k(n)$ factor introduced by the reduction subprograms, and the size is simply $|P_{b(n)}| \cdot |R_n|$. Therefore the height of Q_n is $t(b(n))\log^k(n)$ and the size is $2^{O(s(b(n))+\log^k(n))}$. Finally, observe that the output behavior of Q_n^O on x is the same as $P_n^O(\pi(x))$, so for all x, $\Pr_{O\sim\mathcal{D}_\Sigma}[Q_n(x) = f(x)] \geq p$. □

Definition 3.2.13. *The* **random oracle Σ-way branching program time-space product of f is at least** $b(n)$ *if for every constant $p \in (0, 1]$ and every random*

oracle branching program family of height $t(n)$ and size $2^{s(n)}$ computing f with success probability p, it must be that $t(n) \cdot s(n) \geq \Omega(b(n))$.

We also note that lower bounds on time-space products are roughly preserved by polylog-time reductions of quasi-linear blowup.

Lemma 3.2.14. *Let π be an $O(\log^j(n))$-time reduction with blowup $O(n \log^j(n))$ from f to g. Suppose there are k and d such that the random oracle Σ-way branching program time-space product of f is at least $\Omega(n^d/\log^k(n))$. Then there is some k' such that the random oracle Σ-way branching program time-space product of g is at least $\Omega(n^d/\log^{k'}(n))$.*

Proof. Suppose there is such a reduction π from f to g, and suppose that for all $k' > 0$, g has an oracle Σ-way branching program family of height $t_{k'}(n)$ and size $2^{s_{k'}(n)}$ with success probability $c > 0$ such that $t_{k'}(n)s_{k'}(n) = O(n^d/\log^{k'}(n))$. By Lemma 3.2, for all $k' \geq 0$, f has an oracle Σ-way branching program family of height $t'_{k'}(n)$ and size $s'_{k'}(n)$ such that $t'_{k'}(n)s'_{k'}(n) = O(t(n\log^j(n))\log^j(n)(s(n\log^j(n)) + \log^j(n))) = O(n^d \log^{(d+2)j}(n)/\log^{k'}(n)) = O(n^d/\log^{k'-(d+2)j}(n))$. Letting $k' > k + (d+2)j$, we arrive at a contradiction with our assumption. □

For our #2SAT lower bound, we inspect a reduction of Hoffmeister, and note that it is succinct.

Theorem 3.2.15 ([54])**.** *There is a $\log(n)$-time reduction from #3SAT to #2SAT with blowup $\tilde{O}(n)$.*

Finally, our lower bound for FCircEval exploits the fact that there are highly efficient circuits for sorting n items from $[n]$.

Theorem 3.2.16 ([18, 88, 40])**.** *There is a k such that Sort has $O(\log^k(n))$ time uniform circuits of size $O(n\log^k(n))$.*

Other Related Work. Besides the work cited earlier, there is an extensive literature on proving $\tilde{\Omega}(n^2)$ time-space product lower bounds for computing functions in generic word-RAM-like models (usually by proving a R-way branching program

lower bound). The functions include matrix multiplication and the discrete Fourier transform [93], generalized string matching [3], bit-vector convolution and integer multiplication [4], universal hashing from n bits to $O(n)$ bits [69], and computing various functions over sliding windows [21].

3.3 Lower Bound for NOE and Sorting

In this section, we abstract out key properties of the Unique Elements problem that were used in Beame's proof of an $\Omega(n^2)$ time-space product lower bound for Unique Elements against R-way branching programs [20]. This abstraction is useful in two ways. First, it allows us to easily prove lower bounds for other problems, by simply verifying that the key properties hold. Second, this level of abstraction helps us identify stronger generalizations of the lower bounds: average-case lower bounds against R-way branching programs and RAMs *with random oracles*.

3.3.1 Beame's Method (Without Random Oracles)

To give intuition for our lower bound theorem for branching programs with random oracles (Theorem 3.1.6), we begin with a similar but weaker theorem, which is an abstraction of the technique used by Beame [20].

Theorem 3.3.1. *Let $\{f_n : (\Sigma_n)^n \to (\Sigma_n)^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions, and $g : \mathbb{N} \to \mathbb{N}$ with the following properties.*

1. **[f typically has "long" outputs].** *There is an $\varepsilon > 0$ and $\delta > 0$ such that*

$$\Pr_{x \sim D}[|f(x)| > \delta g(n)] > \varepsilon.$$

2. **[Short branching programs have low probability of printing long substrings of f].** *There is an $\varepsilon > 0$ such that, for all Σ_n-way branching programs P of height at most $n/4$, and for all $m \geq 1$,*

$$\Pr_{x \sim D}[P(x) \text{ is a substring of } f(x) \wedge |P(x)| \geq m] < e^{-\varepsilon m}.$$

Then for $n > n_0$, every Σ_n-way branching program of size $2^{s(n)}$ and height $t(n)$ for f_n has $s(n)t(n) \geq \Omega(n \cdot g(n))$.

Theorem 3.3.1 is motivated by the observation that if a function f on input x requires a long output (Property 1), then some (possibly many) subprograms of a branching program P computing f need to output a large fraction of f. But by Property 2, this is "hard" for all short subprograms: they have low probability of correctly answering a large fraction of f.

Proof of Theorem 3.3.1. Let $f, D, g(n), \varepsilon, \delta$ be given as above. We prove the lower bound by demonstrating that any space-S branching program P of sufficiently low height T has a nonzero probability of error on an input x drawn from D. To do this, we lower bound the probability of error by

$$\Pr_{x \sim D}[|f(x)| > \delta g(n)] - \Pr_{x \sim D}[(|f(x)| > \delta g(n)) \wedge (P(x) = f(x))] .$$

By Property 1 of the hypothesis, the first term is lower bounded by some constant $\varepsilon > 0$. The second term is at most $\Pr_{x \sim D}[|P(x)| > \delta g(n)]$, giving us an error probability of at least

$$\varepsilon - \Pr_{x \sim D}[|P(x)| > \delta g(n)].$$

We now upper-bound $\Pr_{x \sim D}[|P(x)| > \delta g(n)]$.

Without loss of generality, let P be layered, having size 2^S in each layer, and height T. Partition P into $4T/n$ layers of height $n/4$. By the pigeonhole principle, some layer must output at least $ng(n)/4T$ elements of the output. There are at most 2^S such subprograms in that layer, and the probability that P outputs at least $\delta g(n)$ elements correctly is upper bounded by the probability that some layer outputs $m := \delta n g(n)/4T$ elements correctly. As there are at most 2^S such subprograms, by a union bound over property 2 of the hypothesis, the probability that P outputs all elements correctly is upper bounded by

$$2^S e^{-\varepsilon' \delta n g(n)/4T}$$

for some $\varepsilon' > 0$. This implies that the error probability of P on an input x drawn from D is at least $\varepsilon - 2^S e^{-\varepsilon' \delta n g(n)/4T}$.

Finally, we observe that if $ST \leq \alpha \cdot ng(n)$ for sufficiently small $\alpha > 0$, the term $2^S e^{-\varepsilon' \delta n g(n)/4T}$ becomes less than ε. Thus there is some input length n such that the probability of error for P on an input x drawn from D is nonzero. This implies that $ST \geq \Omega(n \cdot g(n))$. □

Beame's lower bound against Unique Elements follows from showing that Unique Elements has the properties required by Theorem 3.3.1. In particular, on random inputs Unique Elements has long outputs with high probability, and it is difficult to guess even a small number of elements in the output of a random input, without seeing most of the input. Instead of reproving the Unique Elements lower bound, we give a lower bound against the problem Non-Occurring Elements (NOE) defined in the introduction, as it admits a slightly easier analysis.

First let us verify Property 1 of Theorem 3.3.1, for the uniform distribution U_n^n and $g(n) = n$.

Proposition 3.3.2 (Property 1 holds for NOE)**.** *For all $\varepsilon > 0$, there is a $\delta > 0$ such that for sufficiently large n,*

$$\Pr_{x \in U_n^n}[|\mathit{NOE}(x)| > \delta n] > 1 - \varepsilon.$$

Proof. The desired bound reduces to a weak version of a well-known Balls-and-Bins bound (see [42, p.75] for a reference). In particular, when selecting m integers from $[n]$ uniformly at random, the number Z of integers in $[n]$ not selected (the non-occurring elements) is tightly concentrated around its mean:

$$\Pr[|Z - E[Z]| > t] \leq 2\exp(-2t^2/m).$$

For our problem, we have $n = m$ and $E[Z] = n/e$. Let $c > 0$ be a parameter, and let

$t = cn$. Therefore we have

$$\Pr[|Z - n/e| > cn] \leq 2\exp(-2c^2 n).$$

Complementing and rearranging variables, the inequality becomes $\Pr[Z \geq n/e - cn] \geq 1 - 2\exp(-2c^2 n)$. Finally, for any $\varepsilon > 0$, we can pick $c \in (0, 1/e)$ such that $\varepsilon > 2\exp(-2c^2 n)$ for sufficiently large n. Letting $\delta \in (0, 1/e - c)$, we have $\Pr[Z > \delta n] > 1 - \varepsilon$. □

Note that the bound of Proposition 3.3.2 is stronger than that required by Theorem 3.3.1. This will be useful for the extension to random oracles later (Theorem 3.1.6).

Next, we verify that for Non-Occurring Elements, Property 2 of Theorem 3.3.1 holds as well. In fact, we prove a stronger statement that allows for extra side randomness in the input (and therefore random oracle branching programs). This randomness can be ignored in the application of Theorem 3.3.1.

Proposition 3.3.3 (Property 2 holds for NOE)**.** *For all integers $n, k, N \geq 0$ and all branching programs P of height at most $n/4$ computing a function with m outputs,*

$$\Pr_{x \sim U_n^n, r \sim U_k^N}[P(xr) \text{ outputs } m \text{ non-occurring elements of } x] < e^{-3m/4}.$$

Proof. Let $Q_\pi(x)$ be the event that $P(x)$ follows π. The desired probability equals

$$\sum_{\text{comp. paths } \pi \text{ in } P} \Pr_{x,r}[Q_\pi(xr)] \cdot \Pr_{x,r}[P(xr) \text{ has } m \text{ non-occurring elements of } x \mid Q_\pi(xr)].$$

We show that for all such π,

$$\Pr_{x,r}[P(xr) \text{ has } m \text{ non-occurring elements of } x \mid P(xr) \text{ follows } \pi] < e^{-3m/4}.$$

From this, it will follow that our desired probability is less than $e^{-3m/4}$.

For a path π, let q be the number of distinct variable queries to x and let q' be

the number of distinct queries made to r. Notice that $q + q' \leq n/4$. To bound the probability that $P(xr)$ has at least m non-occurring elements of x, given that $P(xr)$ follows π, we simply count the relevant numerator and denominator.

The total number of xr that follow π is $n^{n-q} \cdot k^{N-q'}$: there are $n - q$ unqueried inputs of x, and $N - q'$ unqueried inputs of r. The total number of xr that follow π, and for which the m outputs of π are non-occurring elements of x, is at most $(n-m)^{n-q} \cdot k^{N-q'}$: since the m outputs are supposed to be non-occurring in x, none of the remaining $n - q$ unqueried variables can take on any of the m outputs. By simple manipulation, we have

$$\frac{(n-m)^{n-q}k^{N-q'}}{n^{n-q}k^{N-q'}} = (1-m/n)^{n-q} \leq (1-m/n)^{n-n/4} \leq (1-m/n)^{3n/4} \leq e^{-3m/4}.$$

This completes the proof. □

3.3.2 Lower Bounds With Random Oracles

We are now ready to present our main lower bound theorem against branching programs with random oracles.

Restatement of Theorem 3.1.6. *Let $\{f_n : (\Sigma_n)^n \to (\Sigma_n)^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions, and let $g : \mathbb{N} \to \mathbb{N}$ satisfy the following properties.*

1. **[f typically has "long" outputs].** *For all $\varepsilon > 0$, there is an $n_0 \geq 0$ such that for all $n > n_0$, there is a $\delta > 0$ such that*

$$\Pr_{x \in D_n}[|f_n(x)| > \delta g(n)] > 1 - \varepsilon.$$

2. **[Short random-oracle branching programs have low probability of printing long substrings of f].** *Let U_n be the uniform distribution over Σ_n and let $N \leq 2^{s(n)}$ be an integer. There is an $\varepsilon > 0$ such that for all Σ_n-way*

branching programs P of height at most $n/4$,

$$\Pr_{(x,r)\sim D_n \times U_n^N}[(P(xr) \text{ is a substring of } f_n(x)) \wedge (|P(xr)| \geq m)] < e^{-\varepsilon m}.$$

Then, for all $n > n_0$ and $p \in (0,1]$, and for every random oracle Σ_n-way branching program of size $2^{s(n)}$ and height $t(n)$ computing f_n with success probability at least p on inputs drawn from D_n, it must be that $t(n) \cdot s(n) \geq \Omega(ng(n))$.

Theorem 3.1.6 prescribes a general scheme for proving time-space product lower bounds against random oracle R-way branching programs, and thus against word RAMs with random oracles as well.

Proof of Theorem 3.1.6. The proof is similar to Theorem 3.3.1; we focus on highlighting what is different. First, we note that by Proposition 3.2.6, it is sufficient to demonstrate that for all $n > n_0$ and $p \in (0,1]$, and for every Σ_n-way branching program of size $2^{s(n)}$ and height $t(n)$ where $\Pr_{x\sim D_n, r\sim U_n^N}[P(xr) = f_n(x)] > p$, it must be that $t(n) \cdot s(n) \geq \Omega(ng(n))$.

As in Theorem 3.3.1, we establish the lower bound by demonstrating that every $n/4$-height branching program P has a large probability of error on an input $x \sim D_n$. In what follows, assume P successfully computes $f_n(x)$ on inputs (x,r) drawn from $D_n \times U_n^N$ with probability at least p. By property 1 of the hypothesis, letting $\varepsilon := p/2$, there is a δ such that $\Pr_{x\in D, r\sim U_n^N}[|f(x)| > \delta g(n)] > 1 - p/2$.

Analogously as in the proof of Theorem 3.3.1, we lower bound the probability of error of P on D_n by

$$\Pr_{x\sim D}[|f(x)| > \delta g(n)] - \Pr_{x\sim D, r\sim U_n^N}[|f(x)| > \delta g(n) \wedge P(xr) = f(x)];$$

note that this probability is at least $(1 - p/2) - \Pr_{x\sim D, r\sim U_n^N}[|P(xr)| > \delta g(n)]$.

Applying a union bound over all $2^{s(n)}$ subprograms of P as before, but substituting property 2 from the hypothesis, we determine by an analogous argument as Theorem 3.3.1 that $\Pr_{x\sim D, r\sim U_n^N}[|P(xr)| > \delta g(n)]$ is at most $2^{s(n)} e^{-\varepsilon \delta n g(n)/4t(n)}$. Therefore

the error probability of P on D_n is at least

$$1 - p/2 - 2^S e^{\frac{-\varepsilon\delta n g(n)}{4t(n)}}.$$

Finally, if we assume $s(n)t(n) \leq \alpha n \cdot g(n)$ for all $\alpha > 0$, we can tune $2^{s(n)}e^{-\varepsilon\delta n g(n)/(4t(n))}$ to be arbitrarily small: it is at most $2^{s(n)}e^{-\varepsilon\delta s(n)/(4\alpha)}$. Thus we can make the error probability for P on an input x drawn from D_n to be at least $1 - 2p/3$, implying that the success probability is at most $2p/3 < p$. This is a contradiction, so there is some $\alpha > 0$ (depending on δ, ε, and p) such that $s(n)t(n) > \alpha n \cdot g(n)$. □

Remark 3.3.4. *If a function does not satisfy Property 2 of Theorem 3.1.6 but satisfies a slightly weaker property, namely that there is an $\varepsilon > 0$ such that for all $|\Sigma_n|$-way branching programs P of height at most $n/4$,*

$$\Pr_{x \sim D_n}[P(x) \text{ is a substring of } f(x) \wedge P(x) \geq m] < e^{-\varepsilon m},$$

we can still obtain an average case lower bound against f for inputs drawn from D_n, but not necessarily a lower bound against random-oracle branching programs. We omit an exposition of this result, because we have not yet found applications of it.

We conclude this subsection with the lower bound for computing NOE with a random oracle.

Restatement of Theorem 3.1.1. *For every $p \in (0, 1]$, every random oracle n-way branching program family of size $2^{s(n)}$ and height $t(n)$ computing NOE with success probability p must have $t(n) \cdot s(n) \geq \Omega(n^2)$ for all sufficiently large n.*

Proof. Applying Proposition 3.3.2 and 3.3.3 and set $D_n := U_n^n$ and $\varepsilon := 3/4$. Then both properties 1 and 2 of Theorem 3.1.6 hold for Non-Occurring Elements. □

3.3.3 Sort and Random Oracles

By reducing from Non-Occurring Elements to Sort, it follows that random oracle n-way branching programs still require a time-space product of $\Omega(n^2)$ to sort a list $L \in [n]^n$.

Theorem 3.3.5. *For any constant $c \in (0, 1]$, let $\{P_n^O\}$ be an oracle n-way branching program family of size $2^{s(n)}$ and height $t(n)$ such that for all $n, x \in [n]^n$,*

$$\Pr_{O \sim \mathcal{D}_{[n]}} [P_n^O(x) = \mathsf{Sort}(x)] \geq c.$$

Then, $ST = \Omega(n^2)$.

Proof. By contradiction. Assuming the premise is false, we show that Non-Occurring Elements must have random oracle branching programs that are too efficient to exist. Suppose there are $c > 0$, $\{P_n^O\}$, $t(n)$, $2^{s(n)}$ such that $t(n)s(n) \leq o(n^2)$ and the success probability of P_n^O is at least c. We define a new the branching program family Q_n^O, where Q_n^O is constructed by taking the union of $n+1$ distinct copies of P_n^O, called $P_{n,1}^O, ..., P_{n,n+1}^O$, and modifying them as follows. Let the root node of $P_{n,1}^O$ be the root node of Q_n^O. If vertex $v_{i,j}$ in $P_{n,i}^O$ prints i, Q_n^O instead prints nothing and transitions to the corresponding vertex $v_{i+1,j}$ in $P_{n,i+1}^O$. If $v_{i,j}$ prints $k > i$, then Q_n^O prints i and transitions to $v_{i+1,j}$. Finally, we remove all print instructions from $P_{n,n+1}^O$.

Observe now that, on any input x and random oracle O for which $P_n^O(x)$ sorts x, $Q_n^O(x)$ prints the list of non-occurring elements of x in sorted order. Q_n^O essentially enumerates the elements of its input x in sorted order, keeping track of the least number $i \geq 1$ it has not enumerated by being at a vertex in $P_{n,i}^O$. Finally, note that if $P_{n,i}^O$ attempts to print a number greater than i, then i does not occur in the input x.

Furthermore, note the height of Q_n^O is $t'(n) = O(t(n) + n)$ and its size is $2^{s'(n)} = O(2^{s(n)}n)$, giving a time space product of $O((t(n)+n)(s(n)+\log(n)))$. Since, without loss of generality, $t(n) \geq n$ and $s(n) \geq \log(n)$, it follows that the time-space product of $\{Q_n^O\}$ is $t'(n)s'(n) = O(t(n)s(n))$. However, this would imply that $t'(n)s'(n) \leq \Omega(n^2)$, contradicting the lower bound for NOE (Theorem 3.1.1). □

To prove our lower bounds for other problems in the following section, we require a somewhat stronger result: a nearly-quadratic time-space product lower bound for computing the non-occurring elements of a list of n items from $[n]$, as well as sorting n items from $[n]$ for lists encoded over a *finite* alphabet Σ. By a reduction, we can prove this using the general lower bounds of Theorem 3.1.1 and Theorem 3.3.5.

Lemma 3.3.6. *Let $\{f_n : (\Sigma_n)^n \to (\Sigma_n)^*\}$ be a family of functions, and let $f_\Sigma : \Sigma^* \to \Sigma^*$ be such that $f(\langle x \rangle_\Sigma) = \langle f_{|x|}(x) \rangle_\Sigma$, where $\langle s \rangle_\Sigma$ is s encoded by a string over Σ. Suppose $\{f_n\}$ requires a random oracle Σ_n was branching program time-space product of $\Omega(n^d)$. Then there is some $k > 0$ such that f_Σ requires a random oracle Σ-way branching program time-space product of $\Omega(n^d/\log^k(n))$.*

Proof. By contradiction. Suppose for all $k > 0$, f_Σ has random oracle Σ-way branching program family $\{P_n^O\}$ of height $t(n)$ and size $2^{s(n)}$ with success probability $p > 0$ where $t(n)s(n) \neq \Omega(n^d/\log^k(n))$. We show that $\{f_n\}$ has a random oracle n-way branching program family $\{Q_n^O\}$ of height $t'(n)$ and size $2^{s'(n)}$ with success probability $p > 0$ where $t'(n)s'(n) \neq \Omega(n^d/\log^k(n))$. To do this, we simply simulate P_n^O with Q_n^O. We first consider the input $l \in [n]^n$ a length $n\lceil \log_{|\Sigma|}(n) \rceil$ string where each number is represented by some string of characters from Σ. We then modify $P^O_{n\lceil \log_{|\Sigma|}(n) \rceil}$ as follows. For each vertex v which queries an input i, instead query input $\lfloor i/\log_{|\Sigma|}(n) \rfloor$. Then, for all edges (u, v) with label $\alpha \in \Sigma$ in $P^O_{n\lceil \log_{|\Sigma|} \rceil}$, add an edge from u to v with label $\beta \in [n]$ for all β whose representation in base Σ contains α at position i mod $\lceil \log_{|\Sigma|}(n) \rceil$. Finally, we need only modify the output behavior of $P^O_{n\lceil \log_{|\Sigma|}(n) \rceil}$. It prints the representation of characters $\sigma \in \Sigma_n$ using characters from Σ. We need only $O(\log(n))$ extra bits of storage to remember the last $\lceil \log_{|\Sigma|} \rceil$ characters $P^O_{n\lceil \log_{|\Sigma|}(n) \rceil}$ would have printed, and upon reaching enough characters, we simply print the corresponding element of Σ_n and remember that we have started a new character. As in the proof of Theorem 3.3.5, we can do this by creating $O(n)$ copies of our modified branching program and transitioning accordingly and letting this be Q_n^O.

Finally, we see that by construction, $\{Q_n\}$ computes $\{f_n\}$ with the success probability p. Further, we see that the height of Q_n is $O(t(n\log(n)))$ and the size is $2^{O(s(n)+\log(n))}$. By our assumption, we see that $t(n\log(n))(s(n)+\log(n)) = \Omega(n^d)$ and $t(n\log(n))(s(n)+\log(n)) \neq \Omega((n\log(n))^d/\log^{d+1}(n))$. However, this is a contradiction. □

From this we can conclude lower bounds for Non-Occurring Elements and Sort for strings over finite alphabets. First we need to formally define these relations.

Definition 3.3.7 (Non-Occurring Elements and Sort over finite alphabets)**.**

- *Non-Occurring Elements*$_\Sigma : \Sigma^* \to \Sigma^*$ *(or* *NOE*$_\Sigma$*) is a relation which maps the encoding in* Σ *of a list* $\langle L \rangle$ *(where* $L \in [n]^n$ *for some* n*) to* $\langle \mathit{NOE}(L) \rangle$*.*

- *Sort*$_\Sigma : \Sigma^* \to \Sigma^*$ *is the function which maps the encoding of a list* $\langle L \rangle$ *where* $\exists n \in \mathbb{N}(L \in [n]^n)$ *to* $\langle \mathit{Sort}(L) \rangle$*.*

Now we can conclude Lemma 3.3.8.

Lemma 3.3.8. *For all* Σ*, there is some* $k > 0$ *such that, the random oracle* Σ*-way branching program time-space product for both* *Non-Occurring Elements*$_\Sigma$ *and* *Sort*$_\Sigma$ *are both at least* $\Omega(n^2/\log^k n)$*.*

Proof. This follows directly from Theorems 3.1.1 and 3.3.5 and Lemma 3.3.6. □

Remark 3.3.9. *It will be important later that the branching program lower bounds we have established hold even if we allow the branching program to make small perturbations on the output. For example, all above lower bound proofs still go through, if we permit branching programs to output* 0 *at any node that does not already output some other number. Thus our lower bounds hold for any branching program computing* *Non-Occurring Elements* *or* *Non-Occurring Elements*$_\Sigma$ *which prints the binary representation of a list* L*, such that* L *contains all non-occurring elements of* x*, along with any number of elements which are* 0*.*

3.4 Reductions

In this section, we give a series of reductions from NOE and Sort, showing nearly-quadratic time-space product lower bounds for several natural circuit-analysis problems. Each of these reductions are efficient many-one reductions, requiring only poly$(\log n)$) time to look up any bit of the output of the reduction, and only creating problem instances of size $\tilde{O}(n)$ from inputs of size n.

As a warm-up, we observe a very simple reduction from Sort$_{\{0,1\}}$ (sorting $n \log(n)$-bit strings) to FCircEval. Recall in the FCircEval problem, we are given a circuit of

size n with at most n inputs (all fixed to 0-1 values) and at most n outputs, and want to determine its output.

Restatement of Theorem 3.1.2. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of* FCircEval *is at least* $\Omega(n^2/\log^k n)$.

Proof. The idea is that, on an unsorted input, we can succinctly produce a circuit for sorting, and simply ask FCircEval for its output.

We observe that there is an $O(\log n)$-time reduction with blowup $O(\log^k n)$ from Sort to FCircEval. Given a input list x which we would like to sort, we simply produce a circuit C for Sort with x hard-coded as the input. Any bit of the representation can be computed in $O(\log n)$ time, as Sort has $O(\log n)$-time uniform circuits of size $O(n \log^k n)$ and any bit of x can trivially be produced in $O(\log n)$ time. By Lemmas 3.3.8 and 3.2.14, we conclude there is a $k > 0$ such that FCircEval has a time-space lower bound of $\Omega(n^2/\log^k n)$. □

A straightforward corollary of Theorem 3.1.2 is a similar lower bound against a seemingly weaker version of 3SAT.

Definition 3.4.1. *Let Promise-Printing-Unique-SAT be the problem: given a circuit C which is promised to have exactly one satisfying assignment, print the satisfying assignment of C.*

Lemma 3.4.2. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of Promise-Printing-Unique-SAT is at least* $\Omega(n^2/\log^k n)$.

Proof. We give a reduction from FCircEval to Promise-Printing-Unique-SAT. Given an instance C of FCircEval with only constant valued inputs and m output bits $y_1, ..., y_m$, we define a new circuit $C'(x)$ with m free input bits $x_1, ..., x_m$ and one output bit. The circuit C' can be made such that $C'(x_1, ..., x_m) = 1$ if and only if $x_i = y_i$ by using only as many gates as needed to construct C, and $O(m)$ additional gates to check equality of the output bits of C with the free input bits and then to take the

conjunction of these equalities. Observe that $|C'| \leq O(|C|)$. Finally we see this gives us an $O(\log(n))$-time reduction with blowup $O(n)$ from FCircEval to Promise-Printing-Unique-SAT, since FCircEval(C) = Promise-Printing-Unique-SAT(C'), and each bit of the description of C' can be computed by simply looking up bits of C directly or deciding if they are part of the equality check or the conjunction of the equality checks added to C. By Lemma 3.2.14, we can conclude that Promise-Printing-Unique-SAT has a time-space lower bound of $\Omega(n^2/\log^k n)$. □

To establish a connection with printing satisfying assignments for 3CNFs, we consider the problem Promise-Printing-Unique-3SAT, which is just Promise-Printing-Unique-SATrestricted to 3CNF formulas. By another straightforward reduction, it follows that Promise-Printing-Unique-3SAT cannot be computed more efficiently than FCircEval.

Lemma 3.4.3. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of Promise-Printing-Unique-3SAT is at least $\Omega(n^2/\log^k n)$.*

Proof. We present a reduction from Promise-Printing-Unique-SAT to Promise-Printing-Unique-3SAT. Consider the standard reduction from Circuit SAT to 3SAT in which a circuit C with n variables $x_1, ..., x_n$ and m gates is mapped to a CNF ϕ with $O(m)$ clauses, variables $x_1, ..., x_n$ and $O(m)$ extra variables. Assume without loss of generality, blowing up only an additional constant factor in size otherwise, that C is comprised only of NAND gates. In the traditional reduction from Circuit SAT to 3SAT, we use additional variables $y_1, ..., y_m$ for each gate $g_1, ..., g_m$. For each gate $g_i = \neg(g_j \wedge g_k)$, $g_i = \neg(g_j \wedge x_k)$, or $g_i = \neg(x_j \wedge x_k)$, we add to ϕ the constraints $(y_i = \neg(y_j \wedge y_k))$, $(y_i = \neg(y_j \wedge x_k))$, or $(y_i = \neg(x_j \wedge x_k))$ accordingly, where each requires only a constant number of clauses of width 3. It is important to note that this reduction is not strictly speaking a many-one reduction as defined since the satisfying assignment contains extra information. However, notice that printing the first n bits of a satisfying assignment to ϕ in fact is sufficient for computing a satisfying assignment to C, and observe that this reduction can be done in $O(\log^k(n))$ time with

blowup $O(\log^k(n))$ for some k. Just as our argument from lemma 3.2.14, we can conclude that Promise-Printing-Unique-3SAT requires a random oracle Σ-way branching program time-space product of $\Omega(n^2/\log^{k'}(n))$ for some $k' > 0$. □

As a corollary, we can immediately conclude that the harder problem of Print-3SAT also has a time-space lower bound of $\Omega(n^2/\log^k n)$ for some k.

Restatement of Theorem 3.1.3. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of Print-3SAT is at least $\Omega(n^2/\log^k n)$.*

Proof. Follows from the trivial reduction from Promise-Printing-Unique-3SAT to Print-3SAT and Lemma 3.2.14. □

Next, we show by a reduction directly from Non-Occurring Elements$_{\{0,1\}}$ that printing the truth tables of CNF formulas with a small number of variables admits a time-space lower bound similar to those above.

Restatement of Theorem 3.1.4. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of TTPrint for CNF formulas with n clauses and $\log(n)+\log\log(n)$ many variables is at least $\Omega(n^2/\log^k n)$.*

Proof. We consider that the output of a machine computing NOE$_{\{0,1\}}$ can be a list L which contains the non-occurring elements of its input as well as any number of elements which are 0, as in Remark 3.3.9. We give a poly$(\log(n))$-time reduction with blowup $\tilde{O}(n)$ from Non-Occurring Elements$_{\{0,1\}}$ to TTPrint. That is, we will show that given a list L of n elements from $[n]$, we can produce a CNF formula whose truth table is a representation of a list L' of n strings each of $\log(n)$ bits, where the i-th string in L' equals i if $i \in [n] - L$, otherwise the i-th string equals all-zeroes. Then, the lower bound for Non-Occurring Elements$_{\{0,1\}}$ will carry over to TTPrint.

Given a list $L \in \{1, \ldots, n\}^n$, we show how to efficiently construct a CNF formula F with $\log(n) + \log\log(n)$ variables and $O(n)$ clauses such that the truth table of F is the binary representation of the list of non-occurring elements of L separated by strings of 0's.

Denote the first $\log(n)$ variables of F by $x = x_1 \cdots x_{\log(n)}$. Letting $b \in \{0,1\}^{\log(n)}$, we make a clause $C_b(x)$ expressing that $x \neq b$. In detail, suppose b is represented by the bit string $b_1, \ldots, b_{\log(n)}$. Then

$$C_b(x) := ((x_1 \oplus b_1) \vee ... \vee (x_{\log(n)} \oplus b_{\log(n)})).$$

Note for all i, we can think of C_b as containing the literal $\overline{x}_i$ if $b_i = 1$, otherwise it contains the literal x_i.

Given a list $L = \ell_1, \ldots, \ell_n$ of elements of $\{0,1\}^{\log(n)}$, we first construct a CNF formula $F'(x)$ which says that x is a non-occurring element of ℓ:

$$F'(x) := C_{\ell_1}(x) \wedge \cdots \wedge C_{\ell_n}(x).$$

Suppose we can construct a CNF formula $D(x,i)$ which is true exactly when the i-th bit of x is 1, and define our output formula to be

$$F(x,i) := F'(x) \wedge D(x,i)$$

where $i = i_1 \cdots i_{\log\log(n)}$. This F would have $\log(n) + \log\log(n)$ variables, and its truth table in lexicographical order could be viewed as a list of n different $\log(n)$-bit strings, each of which are either a non-occurring element of ℓ, or the all-zeroes string. (This would complete our reduction.)

We show how to construct such a $D(x,i)$ with $|x| = \log(n)$ clauses. For each variable of x, and $j = 1, \ldots, |x|$, we construct a clause E_j which is true if and only if either $i \neq j$ or $x_j = 1$. That is,

$$E_j = ((i_1 \oplus j_1) \vee ... \vee (i_{\log(|x|)} \oplus j_{\log(|x|)}) \vee x_j).$$

Then we can define

$$D(x,i) := \bigwedge_{j=1}^{|x|} E_j.$$

The final formula $F(x_1, ..., x_{\log(n)}, i_1, ..., i_{\log\log(n)})$ is a width-$(\log(n)+1)$ CNF of $O(n)$ clauses, and all of the clauses of F can be efficiently constructed in a local way. □

Using the lower bound on truth table printing, we can now show the lower bound for counting SAT assignments (Theorem 3.1.5). Again we do this by providing a poly$(\log(n))$-time reduction with blowup $\tilde{O}(n)$ from one problem to another. We show how to modify a circuit C to efficiently produce another circuit C', such that the bit representation of the number of satisfying assignments of C equals the truth table of C.

Lemma 3.4.4. *Let C be a circuit of size s with n input variables $x_1, \ldots, x_n$. Then, there exists a circuit C' of size $O(s + 2^n)$ with input variables $x_1, ..., x_n, y_1, ..., y_{2^n}$ such that $\#\mathsf{SAT}(C') = \mathsf{TruthTable}(C)$. Moreover, there is an algorithm that given such a circuit and index i can produce the i-th bit of the description of C' in time $\tilde{O}(\log(s + n + i))$.*

Proof. Let C be a circuit of size s with inputs $x_1, ..., x_n$. First we make a circuit $D(x_1, \ldots, x_n, y_1, \ldots, y_{2^n})$ which outputs 1 if and only if $y \leq 2^{\mathrm{bin}(x)}$, where $\mathrm{bin}(x)$ is the number in $\{1, \ldots, 2^n\}$ represented by the binary string $x_1 \cdots x_n$. Note that D can be constructed with $O(2^n)$ gates (by standard arguments), and any particular gate of D can constructed in poly$(\log(n))$ time. Finally, we define the circuit C' on variables $x_1, \ldots, x_n, y_1, \ldots, y_{2^n}$ by

$$C'(x, y) := C(x) \wedge D(x, y).$$

Observe that the number of assignments satisfying C' is $\Sigma_{x \in \{0,1\}^n} C(x) \cdot 2^{\mathrm{bin}(x)}$, as for each $x \in \{0,1\}^n$ there are $2^{\mathrm{bin}(x)}$ assignments to $y \in \{0,1\}^{2^n}$ such that $C'(x, y) = 1$. This is exactly the number represented in binary by $\mathsf{TruthTable}(C)$. □

From Lemma 3.4.4 and Theorem 3.1.4, we can conclude Lemma 3.4.5.

Lemma 3.4.5. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of $\#\mathsf{Circuit\ SAT}$ is at least $\Omega(n^2/\log^k n)$.*

From Lemma 3.4.5, we can conclude Lemma 3.4.6.

Lemma 3.4.6. *For all finite Σ, there is some $k > 0$ such that the random oracle Σ-way branching program time-space product of #3SAT is at least $\Omega(n^2/\log^k n)$.*

Proof. The standard reduction from #Circuit SAT to #3SAT can be implemented as an $O(\log(n))$-time reduction with blowup $\tilde{O}(n)$. □

From Lemmas 3.4.6 and 3.2.15, we can conclude Theorem 3.1.5.

Restatement of Theorem 3.1.5. *For all finite Σ, there is a $k > 0$ such that the random oracle Σ-way branching program time-space product of #2SAT is at least $\Omega(n^2/\log^k n)$.*

Proof. Lemma 3.2.15 gives an $O(\log^k(n))$ time reduction with blowup $O(n \log^k(n))$ from #3SAT to #2SAT. Hence, by Lemmas 3.4.6 and 3.2.14, we can conclude that the random oracle Σ-way branching program time-space product of #2SAT is at least $\Omega(n^2/\log^k(n))$. □

3.5 Conclusion

In this chapter, we have extended a lower bound framework for branching programs, lifting it to random oracles. We showed its utility for lower bound results by giving several unorthodox reductions which show sharp relationships between counting satisfying assignments, printing truth tables of CNFs, printing satisfying assignments, and evaluating circuits on given inputs. It would be interesting to find more applications of the various encoding techniques used in our reductions. Perhaps they can be used to show lower bounds for other models, or better algorithms.

Another interesting direction would be to explore other lower bound methods, such as those for *decision* problems, and determine to what extent they can be "lifted" to lower bounds with random oracles. To give one tantalizing example, although it is known that deciding SAT requires $n^{1.8}$ time on deterministic $O(\log n)$-space machines, and this chapter shows that *printing* SAT assignments requires $n^{2-o(1)}$ time on *randomized* $O(\log n)$-space machines with constant error probability, it is still open whether *deciding* SAT is in $O(n)$ time and $O(\log n)$ space on randomized

machines with two-sided error(!). Perhaps it is easier to find bridges between function problems and decision problems when we consider efficient programs for **NP** problems (rather than decision problems in **P**).

Lower Bounds for an Extended Oracle Model? Finally, our oracle model for branching programs is not the most general that one could imagine (although for the *random* oracle case, we suspect it does not matter). One can define a sensible "extended oracle" model, where oracle queries can be as long as the height of the branching program. (This model is not very practical in a truly space-bounded setting, e.g. when we view a random oracle as a random hash function, but it would be very interesting to prove lower bounds against.)

At a high level, here is how such an "extended oracle" model can be defined. In each step, we allow the branching program to output an "oracle character" σ from the input alphabet of the oracle (along with its usual outputs). Instead of labeling the query vertices of the branching program with specific query strings (as in Definition 3.2.4), we instead label them with a special symbol Q. The Q-vertices still have two outgoing arcs for their yes/no query answers. However, each time a Q-vertex v is reached during a computation, the outgoing *yes* edge of v is now taken in the computation path if and only if the string of oracle characters $y = \sigma_1 \cdots \sigma_t$ output since the previous Q-vertex (or source node, if there is no previous Q-vertex) satisfies $Q(y) = 1$. One can think of this as allowing the branching program "append-only" access to an arbitrarily long oracle tape, for which it can ask queries, and for which the oracle tape is reset to blank after each query (as in the oracle model of Ladner and Lynch [65]).

With the above model, we can ask queries whose length is only bounded by the height of the branching program (rather than the logarithm of its size). We strongly believe that our lower bounds also hold for random oracles in this more powerful model, but it appears difficult to prove. The main conceptual bottleneck is that, unlike normal branching programs, we cannot easily partition these extended-oracle branching programs into short independent branching programs: the oracle queries

have “memory” that can stretch all the way back to the source node.

Chapter 4

Hardness Magnification for Compression Problems

4.1 Introduction

In this chapter, we show how extremely weak-looking lower bounds on solving canonical string compression problems would imply major separations in complexity theory. Let $s : \mathbb{N} \to \mathbb{N}$ satisfy $s(n) \geq n$ for all n. We start by considering the following circuit synthesis problem:

Problem: $\mathsf{MCSP}[s(n)]$

Given: A function $f : \{0,1\}^n \to \{0,1\}$, presented as a truth table of $N = 2^n$ bits.

Decide: Does f have a (fan-in two) Boolean circuit of size at most $s(n)$?

We can naturally view MCSP as a compression problem: given a string, find a small circuit whose truth table reproduces the string. MCSP seems to have first been studied in the 1950s [85]. Note $\mathsf{MCSP}[s(n)]$ is only non-trivial for sufficiently small $s(n)$, e.g., $s(n) \leq 2^{n+1}/n$ (otherwise, *every* input is a YES instance), and for such non-trivial $s(n)$, $\mathsf{MCSP}[s(n)]$ is in **NP**. Moreover, there is an algorithm for $\mathsf{MCSP}[s(n)]$ running in both $s(n)^{O(s(n))}$ time and $\tilde{O}(s(n)+n)$ space (enumerating all possible size-

$s(n)$ circuits). No further improvements in the time complexity of $\mathsf{MCSP}[s(n)]$ are known.

In 2001, Kabanets and Cai [61] revived the "modern" study of MCSP in theoretical computer science, observing that the Natural Proofs barrier [79] strongly suggests that $\mathsf{MCSP}[n^c]$ is *not* in **P** for sufficiently large c; otherwise, strong one-way functions do not exist.[1] Thus it is widely believed that $\mathsf{MCSP}[\mathrm{poly}(n)]$ is not efficiently solvable, although it is still open whether the general MCSP problem is **NP**-hard, despite much recent work [74, 10, 53, 52, 9, 51, 50].[2] MCSP has recently taken on larger significance for cryptography, since Hirahara recently showed in a celebrated work [49] that there is a worst-case to average-case reduction for the approximation version.

Considering the enormous difficulty of proving polynomial-time lower bounds on problems in **NP**, we may ask if it is possible to prove instead that $\mathsf{MCSP}[\mathrm{poly}(n)]$ cannot be solved in almost-linear time (e.g. $N^{1+o(1)}$) and sub-polynomial space (e.g. $N^{o(1)}$). Over the years, there has been considerable progress in lower bounds for the low time-space setting [22, 5, 23, 6, 44, 90, 29], so there is hope that such lower bounds could be proved.

In this chapter, we show (among many other results) that an $N \cdot \mathrm{poly}(\log N)$-time lower bound on $\mathrm{poly}(\log N)$-space **deterministic streaming algorithms** for $\mathsf{MCSP}[n^c]$ is already as difficult as separating **P** from **NP**! Note that deterministic streaming algorithms are very restricted (one cannot even compute whether two given N-bit strings are equal with $o(N)$ space). Our result is one consequence of a more general theorem proved about a generalization of MCSP, in which we allow compression by circuits with oracle gates (as defined in prior work [10, 59]). This "boosting" of a modest time-space lower bound for an **NP** problem to $\mathbf{P} \neq \mathbf{NP}$ can be seen as a form of *hardness magnification*, a term recently coined by Oliveira and Santhanam [76]. Intuitively, a hardness magnification result is a kind of "slippery slope" theorem, showing how a very modest-looking lower bound for a believably-

[1] In particular, if $\mathsf{MCSP}[n^c] \in \mathbf{P}$ for all $c \geq 1$, then there are **P**-natural properties useful against $\mathbf{P}/poly$. For a more fine-grained discussion of the relationship, see Chow [39].

[2] Observe that $\mathsf{MCSP}[n^c]$ can be solved in $2^{\tilde{O}(n^c)}$ time, which is quasi-polynomial in the input length $N = 2^n$, so it is probably not **NP**-hard; nevertheless it is widely believed to not be in **P**.

hard problem counterintuitively implies incredibly strong lower bounds (perhaps for a different hard problem). Other examples of similar phenomena are known, for $n^{1-\varepsilon}$-approximations to CLIQUE [83], low-depth circuit lower bounds for $\mathbf{NC}^1$ [11], and sublinear-depth circuit lower bounds for $\mathbf{P}$ [67].

4.1.1 Our Results

To state our theorems, we first need a couple of concepts for "generalized" Boolean circuits. Let $A : \{0,1\}^\star \to \{0,1\}$. An *$A$-oracle circuit* has a gate basis of OR, AND, NOT, and all finite slices of A (the function A restricted to inputs of a fixed length). The $\mathsf{MCSP}^A[s(n)]$ problem asks, given $f : \{0,1\}^n \to \{0,1\}$ presented as a truth table, whether f has an A-oracle circuit of at most $s(n)$ gates. Its search version, $\mathsf{search\text{-}MCSP}^A[s(n)]$, requires *outputting* an A-oracle circuit when it exists. The $\Sigma_3\mathsf{SAT}^A$ problem asks, given an A-oracle formula $F(x,y,z)$ *as input* (a Boolean formula over Boolean variables with $\wedge$, $\vee$, $\neg$ and A-oracle predicates), whether there exists an assignment to x such that for all assignments to y, there exists an assignment to z such that $F(x,y,z)$ outputs true.

Our first main theorem constructs a super-efficient $\mathbf{AC}$ circuit family for MCSP^A with short oracle calls to $\Sigma_3\mathsf{SAT}^A$:

Theorem 4.1.1 (Section 4.3)**.** *Let $s(n) \geq n$ and let $\ell(n) \geq s(n)^2$ for all n, where both are time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a uniform $\mathbf{AC}$ circuit family for $\mathsf{MCSP}^A[s(n)]$ on 2^n-bit inputs of $\tilde{O}(2^n \cdot s(n)^2)$ size and $O(n/\log \ell(n))$ depth with $\Sigma_3\mathsf{SAT}^A$ oracle gates, where each oracle gate takes only $\tilde{O}(\ell(n))$ bits of input.*

Note there are three tunable parameters in Theorem 4.1.1: the oracle A, the MCSP parameter $s(n)$, and a larger parameter $\ell(n)$ which affects the circuit depth and the fan-in of the oracle gates. By setting them appropriately, we can derive many consequences regarding hardness magnification for MCSP.

Our second main theorem gives an efficient streaming algorithm for $\mathsf{MCSP}^A[s(n)]$, assuming the algorithm has oracle access to $\Sigma_3\mathsf{SAT}^A$ and can make short queries to

the oracle.

Theorem 4.1.2 (Section 4.4)**.** *Let $s(n) \geq n$ for all n, where $s(n)$ is time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a (one-pass) streaming algorithm for $\mathsf{search\text{-}MCSP}^A[s(n)]$ on 2^n-bit inputs running in $2^n \cdot \tilde{O}(s(n))$ time with $\tilde{O}(s(n)^2)$ update time and $\tilde{O}(s(n))$ space, using an oracle for $\Sigma_3\mathsf{SAT}^A$ with queries of length $\tilde{O}(s(n))$.*

One way to prove Theorem 4.1.2 would be to show that the circuit family derived in Theorem 4.1.1 can be evaluated in a time and space-efficient way. We instead give a direct proof (an explicit streaming algorithm).

Weak Streaming Lower Bounds for Compression Imply P $\neq$ NP. Applying the assumption **P** = **NP** to the oracle calls in Theorem 4.1.2, we obtain a magnification from modest streaming algorithm lower bounds all the way to **P** $\neq$ **NP**:

Theorem 4.1.3 (Section 4.5)**.** *If there is some time-constructible $s(n) \geq n$ and an $A \in$ **PH** such that $\mathsf{search\text{-}MCSP}^A[s(n)]$ is not solvable by a poly$(s(n))$-space streaming algorithm with poly$(s(n))$ update time, then* **P** $\neq$ **NP**.

(Note that we do not require streaming algorithms to read a fresh bit in each step; the algorithm may work for some time before requesting the next bit of the stream.) For example, if one can show that we cannot compress N-bit strings to SAT-oracle circuits of size $N^{o(1)}$ by streaming algorithms running in $N^{1+\varepsilon}$ total time and N^ε space for all $\varepsilon > 0$, then **P** $\neq$ **NP**. Note there is no *information-theoretic* barrier to such a streaming algorithm, only a computational one: in N^ε space, one could easily hold a circuit of size $N^{o(1)}$. Let us stress that the hypothesis of Theorem 4.1.3 is widely believed to hold: if it were false, then MCSP is trivially in **P** (and much more), implying that strong one-way functions do not exist and extremely strong compression is possible efficiently.

Comparing Theorem 4.1.3 With [76].

Oliveira and Santhanam [76] show that if *approximately* solving $\mathsf{MCSP}[2^{\sqrt{n}}]$ (the decision version of MCSP, with no oracle) to within an $O(n)$ factor requires $\Omega(N)$-time by randomized algorithms with two-sided error, then **BPP** $\not\subseteq$ **NP** (equivalent to

$\mathbf{RP} \neq \mathbf{NP}$). In particular, their approximation version of MCSP only requires that a candidate algorithm distinguish truth tables with circuit complexity at most $s(n)$, from truth tables which need a *constant fraction of entries modified* in order to have circuit complexity at most $s(n)$. Proving a lower bound against such a weak guarantee seems intuitively much more difficult than proving a worst-case lower bound against MCSP. (They only need an $\Omega(N)$-time lower bound, because — as in Srinivasan [83] — a random sample of the input preserves the answer to the approximation problem with high probability; this does not hold for the worst-case version.) While their consequence is stronger ($\mathbf{RP} \neq \mathbf{NP}$ rather than $\mathbf{P} \neq \mathbf{NP}$), the hypothesis of Theorem 4.1.3 only requires a slightly super-linear time worst-case lower bound for the *search* version of MCSP against deterministic streaming algorithms using sub-linear space.

Weak Circuit Lower Bounds for Compression Imply Super Polynomial Circuit Lower Bounds. Applying Theorem 4.1.1 with different parameter settings, we show how modest $\mathbf{TC}^0$ lower bounds for $\mathsf{MCSP}^{\mathsf{SAT}}$ imply $\mathbf{NP} \not\subset \mathbf{TC}^0$, modest log-depth circuit lower bounds imply $\mathbf{NP} \not\subset \mathbf{NC}^1$, and larger-depth circuit lower bounds imply $\mathbf{NP} \not\subset \mathbf{P}/poly$.

Theorem 4.1.4 (Section 4.5)**.** *Let $s(n) \geq n$, and let $A \in \mathbf{PH}$.*

- *If there exists an $\varepsilon > 0$ such that for every $c \geq 1$, search-MCSP$^A[2^{\varepsilon n/c}]$ on inputs of length $N = 2^n$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{NP} \not\subset \mathbf{TC}^0$.*[3]

- *If search-MCSP$^A[s(n)]$ on inputs of length $N = 2^n$ does not have circuits of $N \cdot poly(s(n))$ size and $O(\log N)$ depth, then $\mathbf{NP} \not\subset \mathbf{NC}^1$.*

- *If search-MCSP$^A[s(n)]$ on inputs of length $N = 2^n$ does not have circuits of $N \cdot poly(s(n))$ size and $poly(s(n))$ depth, then $\mathbf{NP} \not\subset \mathbf{P}/poly$.*

[3]Note that $\mathbf{TC}^0$ could be substituted with any other constant-depth circuit family, such as $\mathbf{AC}^0[6]$.

For example, if we find an $c \geq 1$ and A in the polynomial-time hierarchy such that compression of N-bit strings to $(\log N)^c$-size A-oracle circuits cannot be performed by $N \cdot \text{poly}(\log N)$-size $O(\log N)$-depth circuits, then we can conclude **NP** does not have polynomial-size formulas. If the depth lower bound can be improved to arbitrary polylogs, then already **NP** is not in $\mathbf{P}/poly$.

Comparison With [76] and [75]. Oliveira and Santhanam show that lower bounds of size $N \cdot \text{poly}(\log N)$ for *formulas* solving $\mathsf{MCSP}[n^c]$ (decision version, with no oracle) in an *approximate* or *average-case* setting would imply $\mathbf{NP} \not\subset \mathbf{NC}^1$. While their required formula lower bound is more modest, the "decision version", "average-case", "approximate", and "no oracle" restrictions would presumably make proving a lower bound considerably more difficult. Oliveira, Pich, and Santhanam [75] show that lower bounds on $O(n)$-approximation to $\mathsf{MCSP}[2^{\varepsilon n}]$ (for all $\varepsilon > 0$) with $N^{1+\delta}$-size circuits imply $\mathbf{NP} \not\subset \mathbf{P}/poly$, whereas our results show $\mathbf{NP} \not\subset \mathbf{P}/poly$ follows from $N^{1+\delta}$-size N^{δ}-depth circuit lower bounds on *exactly* solving $\mathsf{MCSP}[2^{\varepsilon n}]$. However, note that proving even lower bounds for $O(N)$-size $O(\log N)$-depth circuits remains an open challenge in complexity theory, even for functions with $N^{\Omega(1)}$ outputs.

Comparison With [11]. It is also interesting to compare the $\mathbf{TC}^0$ results of Theorem 4.1.4 with the work of Allender and Koucký on amplifying lower bounds for low-depth circuit classes [11]. They showed that for some natural $\mathbf{NC}^1$ problems such as Boolean Formula Evaluation, proving that there are no $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits for those problems imply a full separation: $\mathbf{NC}^1 \not\subset \mathbf{TC}^0$. The first bullet of Theorem 4.1.4 manages to prove an analogous result for some problems in **PH**! Allender and Koucký also show unconditionally that for sufficiently large d, SAT does not have $n^{1+1/(3d)}$-size d-depth *uniform* $\mathbf{TC}^0$ circuits. The proof of Theorem 4.1.4 implies that slightly stronger results for $\mathsf{MCSP}^{\mathsf{SAT}}[2^{o(n)}]$ would separate **NP** and uniform $\mathbf{TC}^0$. (In fact, analogous statements hold for compression problems in $\mathbf{P}^{\mathbf{PP}}$, **PSPACE**, and **EXP**; see Theorems 4.1.5 and 4.1.6 below.)

Super-Polynomial Lower Bounds for Larger Complexity Classes. In general, MCSP^A gets "harder" as the oracles A get more expressive. By choosing more

powerful oracles A in the statement of Theorem 4.1.1, we obtain magnification for problems in **EXP**, **PSPACE**, and $\mathbf{P}^{\mathbf{PP}}$ as well. Allender-Buhrman-Koucký-van Melkebeek-Ronneburger [8] showed that $\mathsf{MCSP}^{\mathsf{QBF}}$ is **PSPACE** complete under randomized poly-time Turing reductions: for all $\delta > 0$, every **PSPACE** language can be decided in randomized polynomial time with oracle calls to $\mathsf{MCSP}^{\mathsf{QBF}}[2^{\delta n}]$.[4] They also showed $\mathsf{MCSP}^{\mathbf{EXP}}$ is **EXP**-complete under $\mathbf{P}/poly$ and **NP** Turing reductions. Impagliazzo, Kabanets, and Volkovich [59] gave an oracle $A \in \mathbf{PP}$ such that MCSP^A is **PP**-complete under randomized poly-time Turing reductions. For these complexity classes $\mathcal{C}$, Theorem 4.1.1 implies that modest lower bounds for MCSP^A with $A \in \mathcal{C}$ already proves $\mathbf{P} \neq \mathcal{C}$ and $\mathcal{C}$. Here are two formalizations of this general statement.

Theorem 4.1.5 (Magnifying Streaming Lower Bounds for Harder MCSP Versions, Section 4.5)**.** *Let $\mathcal{C}$ be one of* **NP**, **PP**, *or* **PSPACE**. *Suppose there is a constructible $s(n)$ and oracle $A \in \mathcal{C}$ such that for all $c \geq 1$, $\mathsf{search\text{-}MCSP}^A[s(n)]$ on inputs of length 2^n has no $s(n)^c$-space streaming algorithm with update time $s(n)^c$. Then $\mathbf{P} \neq \mathcal{C}$.*

Theorem 4.1.6 (Magnifying Low-Depth Circuit Lower Bounds for Harder MCSP, Section 4.5)**.** *Let $\mathcal{C}$ be one of* **NP**, **PP**, *or* **PSPACE**. *Suppose there is some $s(n)$ and oracle $A \in \mathcal{C}$ such that for all $c \geq 1$, $\mathsf{search\text{-}MCSP}^A[s(n)]$ on inputs of length 2^n has no circuits of depth $O(n)$ and $2^n \cdot s(n)^c$ size. Then $\mathcal{C}$ does not have polynomial-size formulas (i.e., $\mathcal{C} \not\subset \mathbf{NC}^1$).*

For instance, proving $\mathsf{MCSP}^{\mathbf{EXP}}[n^{10}]$ does not have quasi-linear size circuits of logarithmic depth would imply $\mathbf{EXP} \not\subset \mathbf{NC}^1$.

Kt and KT Complexity. Our techniques can also be applied to time-bounded analogues of Kologorov complexity:

Problem: $\mathsf{MKtP}[p(N)]$ [66]
Given: A string $x \in \{0,1\}^N$.

[4] Their result was not explicitly stated in this way, but it follows easily from padding.

Decide: Is there a Turing machine M of description length c that prints x in at most t steps, where $c + \log_2(t) \leq p(N)$?

Problem: $\mathsf{MKTP}[p(N)]$ [7]

Given: A string $x \in \{0,1\}^N$.

Decide: Is there a Turing machine M of description length c that, given i, prints the i-th bit of x in at most t steps, where $c + t \leq p(N)$?

While $\mathsf{MKTP}[p(N)]$ is an **NP** problem like MCSP (in fact, the KT complexity of a truth table, which minimizes the measure in the MKTP definition above, is always within polynomial factors of the minimum circuit complexity [7]), $\mathsf{MKtP}[p(N)]$ is a more difficult problem (due to the logarithm of running time). Furthermore, MKtP has analogous properties to $\mathsf{MCSP}^{\mathbf{EXP}}$: for large enough $p(N)$, $\mathsf{MKtP}[p(N)]$ is complete for **EXP** under **NP** and $\mathbf{P}/poly$ Turing reductions [8]. Thus MKTP and MKtP are closely related to MCSP and $\mathsf{MCSP}^{\mathbf{EXP}}$, respectively. We observe that the proof strategy of Theorems 4.1.1 and 4.1.2 extend to MKTP and MKtP in an analogous way (Theorems 4.3.2 and 4.4.2 for MKTP, Theorems 4.3.3 and 4.4.2 for MKtP), leading to the following consequences:

Theorem 4.1.7 (Consequences for MKTP and MKtP)**.** *Let the function $p(N) \geq \log(N)$ be time constructible.*

- *If $\mathsf{MKTP}[p(N)]$ is not solvable by a $poly(p(N))$-space streaming algorithm with $poly(p(N))$ update time, then $\mathbf{P} \neq \mathbf{NP}$.*
- *If there is an $\varepsilon > 0$ such that for all $c \geq 1$, $\mathsf{MKTP}[N^{\varepsilon/c}]$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{NP} \not\subset \mathbf{TC}^0$.*
- *If $\mathsf{MKTP}[p(N)]$ does not have $N \cdot poly(p(N))$-size $O(\log(N))$-depth circuits, then $\mathbf{NP} \not\subset \mathbf{NC}^1$.*
- *If $\mathsf{MKTP}[p(N)]$ does not have $N \cdot poly(p(N))$-size $poly(p(N))$-depth circuits, then $\mathbf{NP} \not\subset \mathbf{P}/poly$.*

- *If* $\mathsf{MKtP}[p(N)]$ *does not have* $N \cdot poly(p(N))$*-size* $poly(p(N))$*-depth circuits, then* $\mathbf{EXP} \not\subset \mathbf{P}/poly$.

- *If* $\mathsf{MKtP}[p(N)]$ *does not have* $N \cdot poly(p(N))$*-size* $O(\log(N))$*-depth circuits, then* $\mathbf{EXP} \not\subset \mathbf{NC}^1$.

- *If there is an* $\varepsilon > 0$ *such that for all* $c \geq 1$, $\mathsf{MKtP}[N^{\varepsilon/c}]$ *does not have* $N^{1+\varepsilon}$*-size* $O(1/\varepsilon)$*-depth* $\mathbf{TC}^0$ *circuits, then* $\mathbf{EXP} \not\subset \mathbf{TC}^0$.

As might be expected, the above claims also hold for the corresponding search versions, MKTP and search-MKtP, and relativized versions with MKTP^A and MKtP^A hold with appropriate modifications.

Comparison With [76] and [75]. Both references [76] and [75] show that lower bounds on approximating MKtP would imply lower bounds against **EXP**. Consider the problem Gap-MKtP$[\alpha(N), \beta(N)]$ where we are promised that the Kt complexity of a given string of length N is either at most $\alpha(N)$ or at least $\beta(N)$, and we have to distinguish the two cases. The first reference shows that if there is a $\delta > 0$ and an $N^{1+\delta}$-size lower bound on Gap-MKtP$[N^\varepsilon, N^\varepsilon + 5\log(N)]$ for all $\varepsilon > 0$, then $\mathbf{EXP} \not\subset \mathbf{P}/poly$. The second reference generalizes this connection to other circuit classes $\mathcal{C} \subset \mathbf{P}/poly$, showing that an $N^{1+\delta}$-size lower bound on $\mathcal{C}$-circuits for Gap-MKtP$[N^\varepsilon, N^\varepsilon + c\log(N)]$ for all $\varepsilon > 0$ would imply $\mathbf{EXP} \not\subset \mathcal{C}$. Theorem 4.1.7 strengthens several of the prior magnification results in multiple ways. In particular, we show lower bounds on the *exact* MKtP problem (with no gap) against $N^{1+\varepsilon}$-size N^ε-depth circuits are already sufficient for $\mathbf{EXP} \not\subset \mathbf{P}/poly$. Finally, [75] also show other interesting connections of MKtP to other classes, e.g., B_2 and U_2 formula lower bounds and AND-THR-THR-XOR circuit lower bounds, which we do not discuss here. Our techniques do not seem to apply to these cases.

4.1.2 Intuition

To give a feeling for our results, let us describe the proof of Theorem 4.1.3 at a high level: how minor streaming lower bounds on compression by circuits would

imply $\mathbf{P} \neq \mathbf{NP}$. We proceed by proving the contrapositive: assuming $\mathbf{P} = \mathbf{NP}$, we construct an extremely efficient streaming algorithm that can compress 2^n-bit strings with small $s(n)$-size circuits, where $s(n) \geq n$.

It would suffice to design a good streaming algorithm that always maintains a circuit of size $s(n)$ that "agrees" with all bits it has seen so far, or reports when no such circuit exists. Our first idea is to identify an intermediate problem that we call Basic-Circuit-Merge,[5] which is not too complex, but can help maintain such a circuit over time. For simplicity, here let's say that Basic-Circuit-Merge takes as input two $s(n)$-size circuits C_1 and C_2, along with disjoint intervals $I_1, I_2 \subseteq [1, 2^n]$ (specified in $O(n)$ bits), and produces a circuit C' of size $s(n)$ which agrees with C_1 on all inputs x_i where $i \in I_1$, and agrees with C_2 on all inputs x_j where $j \in I_2$. (If there is no such C', Circuit-Min-Merge reports that.) Now we observe:

- Basic-Circuit-Merge is computable in the polynomial-time hierarchy. In particular, the problem of printing bits from the description of a fixed C' can be placed in $\Sigma_3\mathbf{P}$, by guessing a C' of size at most $s(n)$, checking for all inputs in I_1 and I_2 that C' is consistent with C_1 and C_2 (respectively), and checking for all $C'' < C$ (with respect to some ordering) that C'' is not consistent with C_1 and C_2. Our assumption $\mathbf{P} = \mathbf{NP}$ implies that Basic-Circuit-Merge has a $\text{poly}(s(n))$ time algorithm on inputs of length $\tilde{O}(s(n))$.

- Armed with a $\text{poly}(s(n))$-time algorithm for Basic-Circuit-Merge, we *can* quickly maintain a circuit of size $s(n)$ consistent with the input read: inductively suppose that we have a circuit C that is consistent with all previous bits read. For the next block B of $s(n)$ consecutive bits of the input, we can construct a trivial DNF F of size $n \cdot s(n)$ that is consistent with the inputs in B, and run Basic-Circuit-Merge in $\text{poly}(s(n))$ time on C and F to produce a new circuit that is consistent with all input read so far, including B. This uses $\text{poly}(s(n))$ update time per bit read, and $\text{poly}(s(n))$ space.

[5] We define a slightly different language called Stream-Merge for the proofs given in Section 4.4, but Basic-Circuit-Merge is sufficient for explaining the idea here.

We conclude that if $\mathbf{P} = \mathbf{NP}$, then for all (reasonable) functions $s(n)$, we can compress strings to size-$s(n)$ circuits with a streaming algorithm using $\text{poly}(s(n))$ update time and $\text{poly}(s(n))$ space.

The above approach can be generalized in many ways. For one, we can easily extend it to MCSP with oracle gates, by defining a stronger Circuit-Min-Merge problem that can handle such gates. (For example, the SAT-oracle version of Circuit-Min-Merge is in $\Sigma_4\mathbf{P}$, thus $\mathbf{P} = \mathbf{NP}$ also implies efficient streaming algorithms for compression with SAT-oracle circuits.) For another, the ideas can be extended to time-bounded Kolmogorov complexity (both KT and Kt complexity), by defining appropriate analogues of Circuit-Min-Merge which allow us to consistently maintain a short Turing machine description of our input over time.

Extending the above outline to restricted circuit classes (rather than streaming algorithms) is more involved. To show that (for example) $\mathbf{NP} \subset \mathbf{TC}^0$ implies small $\mathbf{TC}^0$ circuits for compression by $s(n)$-size circuits, we need to make a few modifications we briefly describe at a high level (but require care to work). First, we allow Circuit-Min-Merge to take in an unbounded number of circuits on disjoint intervals, and its task is to "merge" all of them into one small circuit. Second, we build an $O(1)$-depth "tree" of Circuit-Min-Merge oracle gates, where each oracle gate in $\text{poly}(s(n))$ inputs (and outputs of "child" gates feed into the inputs of "parent" gates), whose leaves span the entire 2^n-length input, and where the output gate (the root of this tree) either prints a small circuit for the input or reports that none exists. Finally, applying the assumption $\mathbf{NP} \subset \mathbf{TC}^0$, we argue that inserting $\text{poly}(s(n))$-size $\mathbf{TC}^0$ circuits for Circuit-Min-Merge in this tree (in place of the oracle calls) would yield $2^n \cdot \text{poly}(s(n))$ size $\mathbf{TC}^0$ circuits for size-$s(n)$ MCSP on inputs of length 2^n.

4.1.3 What Do These Results Mean?

As is the case with other "amplification" and "magnification" results [83, 11, 67, 72, 76], the results described in this chapter have a strong "slippery slope" property: rather innocent-looking lower bounds on solving compression problems contained in a class $\mathcal{C}$ are in fact as hard as proving very strong complexity lower bounds for $\mathcal{C}$.

It seems obligatory to ask: ***What are we to make of such theorems?*** Should we seriously consider these results as suggesting an approach towards major lower bounds such as $\mathbf{EXP} \not\subset \mathbf{TC}^0$ and even $\mathbf{P} \neq \mathbf{NP}$? Or, if we accept the somewhat popular style of argument that "if A implies B, and B is hard to prove, then A is hard to prove", should we believe we have found a new kind of barrier for proving innocent lower bounds? In all honesty, we are not sure. However, two comments seem significant.

1. The aforementioned magnification and amplification results collected so far constitute a remarkable intuition-breaking phenomenon that demands closer attention. In the case of [76, 75, 70] and this chapter, the magnification results highlight the peculiar "weirdness" of MCSP, MKtP, and MKTP, showing how extremely weak-looking lower bounds for these problems turn out to already be as difficult as separating polynomially-strong complexity classes. In this regard, we believe such theorems to be noteworthy contributions towards a better understanding of these fundamental compression problems, and a better understanding of how close/far we are from proving major separations in complexity theory.
2. It seems strange to call an implied consequence B a "barrier" to establishing A, when everyone expects B to be true (as is the case for all lower bound consequences in this chapter). Certainly this is not the sort of barrier that arises with relativization [16], natural proofs [79], and algebrization [2], where we either have unconditionally true claims (a relativizing/algebrizing proof *cannot* resolve $\mathbf{P}$ vs $\mathbf{NP}$) or implied consequences we do not believe to be true (a natural proof of $\mathbf{NP} \not\subset \mathbf{P}/poly$ implies there are no strong one-way functions). It is intriguing to wonder whether other impediments to complexity lower bounds (or algorithms) follow from the algorithms and circuits constructed in this chapter. (Recall that our main results are not implications per se, but rather unconditional constructions of fast algorithms and circuits with short calls to $\Sigma_3\mathsf{SAT}$.)

4.2 Preliminaries

Here we recall and define some notions useful for this chapter.

Streaming Algorithms. We use a standard model for streaming algorithms. A *space-$s(n)$ streaming algorithm* with update (and reporting) time $u(n)$ on an input $x \in \{0,1\}^n$ has a working storage of $s(n)$ bits. At any point, the algorithm can either choose to perform one operation (from a standard palette of RAM operations) on $O(1)$ bits in storage, or it can choose to read the next bit x_i from the input (starting with x_1). The total time between two next-bit reads is at most $u(n)$, and the final output is reported in $O(u(n))$ time.

4.2.1 An Intermediate Problem

In order to prove our main hardness magnification result for circuits Theorem 4.1.1 (in Section 4.3), we define an intermediate problem $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$, and show that $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ can be solved efficiently using $\Sigma_3\mathsf{SAT}^A$ oracle gates. (A similar problem will be defined for proving Theorem 4.1.2, in Section 4.4.)

Problem: $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$
Given: Descriptions of A-oracle circuits $C_1, \ldots, C_t$ with n inputs and one output, a list of integers $(a_1, b_1), \ldots, (a_t, b_t) \in [2^n] \times [2^n]$ such that $a_i < b_i \leq a_{i+1}$ for all i.
Output: ***Either*** the string $1\langle C' \rangle$ where C' is the lexigraphically first minimum-size A-oracle circuit of size at most $s(n)$ such that for all $x \in \{0,1\}^n$ and all i, if $a_i \leq x < b_i$ then $C'(x) = C_i(x)$, ***or*** the all-0 string of length $100s(n)\log(s(n))$ when there is no such circuit.

(In the above, 100 is a placeholder for a sufficiently large constant c for encoding size $s(n)$ circuits in $cs(n)\log_2(s(n))$ bits; certainly 100 suffices.) Note that any bit of the output string $1\langle C' \rangle$ can be computed by a Σ_3^A machine in $\tilde{O}(m)$ time for inputs of length m, by guessing and checking the (unique) circuit C' in the output specification, then verifying that every circuit C'' that comes before C' (with respect to the natural

ordering) fails the specification. By a standard reduction (described further in the proof of Theorem 4.1.1), we can compute $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ with $100s(n)\log(n)$ parallel queries to $\Sigma_3\mathsf{SAT}^A$ (where each query computes a bit of the output). Because the queries can be done in parallel, we can then compute $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ with either a low depth circuit with $\Sigma_3\mathsf{SAT}^A$ oracle gates, or $\Sigma_3\mathsf{SAT}^A$ oracle queries in a streaming algorithm.

4.3 Efficient Oracle Circuit Family for MCSP

In this section, we give an efficient circuit family for $\mathsf{MCSP}^A[s(n)]$ with low fan-in $\Sigma_3\mathsf{SAT}^A$ gates.

Reminder of Theorem 4.1.1. *Let $s(n) \geq n$ and let $\ell(n) \geq s(n)^2$ for all n, where both are time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a uniform* **AC** *circuit family for $\mathsf{MCSP}^A[s(n)]$ on 2^n-bit inputs of $\tilde{O}(2^n \cdot s(n)^2)$ size and $O(n/\log \ell(n))$ depth with $\Sigma_3\mathsf{SAT}^A$ oracle gates, where each oracle gate takes only $\tilde{O}(\ell(n))$ bits of input.*

For example, for any $\varepsilon > 0$, if $s(n)$ is at most $O(2^{(1-\varepsilon)n/2})$, then we can let $\ell(n) = 2^{(1-\varepsilon)n}$ and our circuit will have constant depth.

To prove Theorem 4.1.1, we first show that there are small low-depth circuits using $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ oracle gates. (See the Preliminaries in Section 4.2.1 for the definition of $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$.)

Lemma 4.3.1. *Let $s(n) \geq n$ and let $\ell(n) \geq s(n)^2$ for all n, where both are time constructible. There is a uniform* **AC** *circuit family for $\mathsf{MCSP}^A[s(n)]$ of $\tilde{O}(2^n \cdot s(n)/\ell(n))$ size and $O(n/\log \ell(n))$ depth with $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ oracle gates, where each oracle gate takes only $\ell(n)$ bits of input.*

Proof. Suppose $s(n) \geq n$ and $\ell(n) \geq s(n)^2$ for all n, and both are time constructible. For the sake of clarity, we use the letter "G" to describe $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ gates used in the construction of our final circuit C_{2^n}, and the letter "D" to describe *bit-descriptions* of circuits. To simplify our description, we assume (without loss of

generality) that there is no "integer roundoff", i.e., quantities like $\log_d(2^n)$ are integers.

Construction. Let T be an input of 2^n bits. To make the circuit C_{2^n}, we start by taking 2^n *descriptions* of n-input constant-size circuits $D^0_0, D^0_1, \ldots, D^0_{2^n-1}$ such that $D^0_x(y) = T(x)$ for all $x, y \in \{0,1\}^n$, each circuit encoding one bit of the input truth table T. Since these are constant-size circuits, the length of each descriptions D_i is at most $s(n)$ for almost all input lengths n.

Let $d = \ell(n)/(100s(n)\log s(n))$. Our circuit C_{2^n} will include a d-ary tree of $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ gates G^i_j of fan-in $\ell(n)$. At each layer, we divide the previous layer of circuits into blocks of d circuits. Then for each block of d circuits, we use a single $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ gate to combine the block into one output circuit.

To demonstrate, start with the bottom layer. Taking the constant-size circuits D^0_j as input, we build a layer of $2^n/d$ circuits

$$G^1_0, \ldots, G^1_{(2^n/d)-1},$$

where $G^1_j = \mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)](D^0_{j\cdot d}, \ldots D^0_{(j+1)\cdot d}, (j\cdot d, j\cdot d+1), \ldots, ((j+1)\cdot d-1, (j+1)\cdot d))$. That is, each G^1_j takes a contiguous block of d descriptions from the bottom layer, and outputs one circuit of size at most $s(n)$ consistent with these d circuits (or reports that no such circuit exists).

We repeat this process on the *descriptions output by the circuits* G^1_j: at each new layer, we divide the previous layer of circuits into blocks of d circuits, using one $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ gate for each block. That is, for $i \in \{1, \ldots, \log_d(2^n)\}$ at layer i, the $\mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ gate G^i_j takes as input circuit descriptions $D^{i-1}_{j\cdot d}, \ldots, D^{i-1}_{(j+1)d-1}$ along with the ordered pairs

$$(jd^i, jd^i + d^{i-1}), (jd^i + d^{i-1}, jd^i + 2d^{i-1}), \ldots, ((j+1)d^i - d^{i-1}, (j+1)d^i)$$

and outputs the description of a new circuit D^i_j.

Finally, to get a circuit for MCSP, the output of the circuit C_{2^n} is the AND of the first output bit of each G^i_j in the circuit. To get a circuit for $\mathsf{search\text{-}MCSP}$, we

can AND the output gate of C_{2^n} with each bit of $G_0^{\log_d(2^n)}$. This circuit either prints the all-zeroes string when no circuit exists, or prints a size $s(n)$ circuit for the entire input.

Correctness. To prove that C_n computes $\mathsf{MCSP}^A[s(n)]$ on 2^n-bit inputs, we will first prove by induction that, assuming small circuits exist for the input truth table T, the circuit D_j^i output by gate G_j^i matches the input on bits jd^i through $(j+1)d^i - 1$. By construction, the $O(1)$-size circuit D_j^0 matches T on bit j for every j. For $i > 0$, suppose that all circuits D_j^{i-1} match T on bits jd^{i-1} through $(j+1)d^{i-1}-1$. The circuit D_j^i is the output of $G_j^i = \mathsf{Circuit\text{-}Min\text{-}Merge}^A[s(n)]$ on circuits $D_{j\cdot d}^{i-1}, \ldots, D_{(j+1)d-1}^{i-1}$ with ordered pairs

$$(jd^i, jd^i + d^{i-1}), (jd^i + d^{i-1}, jd^i + 2d^{i-1}), \ldots, ((j+1)d^i - d^{i-1}, (j+1)d^i).$$

This means that D_j^i matches D_{jd}^{i-1} on bits jd^i through $jd^i + d^{i-1} - 1$, and since D_{jd}^{i-1} matches T on bits $(jd)d^{i-1} = jd^i$ through $(jd+1)d^{i-1} - 1 = jd^i + d^{i-1} - 1$, we get that D_j^i matches T on these bits as well. Similarly, the bits on which D_j^i are forced to match a D_*^{i-1} circuit are exactly those bits on which the D_*^{i-1} circuit matches T, which means that the final circuit D_j^i matches T on all bits covered by D_{jd}^{i-1} through $D_{(j+1)d-1}^{i-1}$, which is jd^i through $(j+1)d^i - 1$, completing the induction.

So for all i, j, the output of G_j^i is then the description of a circuit D_j^i of size at most $s(n)$ whose truth table matches the desired truth table T on bits $j \cdot d^i$ through $(j+1) \cdot d^i - 1$ (or $0^{100s(n)\log s(n)}$ if such a circuit does not exist).

The final AND gate takes the first output bit of each of these G_j^i gates. We claim that all of these bits are 1 if and only if the input truth table T has a circuit of size at most $s(n)$. To prove this, we need only show

1. Each gate G_j^i and G' outputs 1 as its first bit if T has a circuit of size at most $s(n)$.

2. Some G_j^i or G' outputs 0 as its first bit if T does not have a circuit of size at most $s(n)$.

Suppose T has an A-oracle circuit of size at most $s(n)$. Then this oracle circuit serves as a witness for every $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ computation in the circuit; by construction, gate G^i_j outputs a circuit consistent with T on bits jd^i through $(j+1)d^i$ if such a circuit exists. However, the $s(n)$ size circuit which computes T will obviously be consistent with T on these bits (as well as every other bit of T), which means that the minimum circuit must have at most $s(n)$ gates. The minimum circuit may in fact be smaller (which is likely the case for the constant circuits D^0_j), but can never be larger than $s(n)$. So if all circuits G^i_j at layer i output a circuit D^i_j which is of size at most $s(n)$, then every circuit G^{i+1}_j at layer $i+1$ will output a circuit D^{i+1}_j which is of size at most $s(n)$. By induction, for all i, j, G^i_j will output a valid circuit, with 1 as its first bit, which means that the AND gate at the top will output 1 as well.

Now suppose that T has no A-oracle circuit of size $s(n)$. Then there should be some place in the circuit where $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ fails to combine the intervals into a single small circuit. Start at the top $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ gate $G_0^{\log_d(2^n)}$. If all d inputs to this gate are valid circuits, then the output of this circuit must still be $0^{100s(n)\log s(n)}$, since the output must be a size $s(n)$ circuit which matches T on all input bits, and we assumed that such a circuit does not exist. On the other hand, if one of the inputs to the gate is not a valid circuit then we can move to layer $\log_d(2^n) - 1$ and repeat this argument. Each time, either all inputs are valid circuits and the output is $0^{100s(n)\log s(n)}$ (the circuit does not exist for this interval), or we can recurse on one of the invalid inputs. Since there are constant-size circuits at the bottom, there must be some gate which outputs $0^{100s(n)\log s(n)}$ with valid circuits as input, which will force the AND to output 0 as well.

Recall the circuit C_n is a d-ary tree of $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ circuits with a single AND gate at the top. This gives a total depth of $\log_d(2^n) + 1 = n/\log_2(d) + 1 = n/(\log \ell(n) - \log s(n)) + 1 = O(n/\log \ell(n))$, and a size bound of $O(2^n/d)$ gates, which is a little suboptimal. To get the optimal bound, we can increase the number of circuits fed into the $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ circuits at the bottom layer to $O(\ell(n)/n)$, and maintain a fan-in of $\ell(n)$ to each $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ oracle gate. With this, the size of the circuit can be reduced to $O(2^n \cdot n/\ell(n)) \leq \tilde{O}(2^n/\ell(n))$. □

Finally, we explain why Theorem 4.1.1 follows (with the decision problem $\Sigma_3\mathsf{SAT}^A$ in place of $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$). First, recall that $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ is a function problem, outputting $\tilde{O}(s(n))$ bits, where each output bit is computable by a Σ_3 machine (making Σ_3-style alternations) in $\tilde{O}(n)$ time. By standard completeness results (see for example, in [44]) there is a simple $\mathbf{AC}^0$ reduction of size $\tilde{O}(\ell(n) \cdot s(n))$ from the $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ problem to $\tilde{O}(s(n))$ copies of $\Sigma_3\mathsf{SAT}^A$: we can directly map $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ instances with $\ell(n) > s(n)^2$ inputs and $\tilde{O}(s(n))$ outputs to $\tilde{O}(s(n))$ instances of $\Sigma_3\mathsf{SAT}^A$ where each instance has length $\tilde{O}(\ell(n))$. Hence each $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ oracle gate can be replaced by $\tilde{O}(\ell(n) \cdot s(n))$ extra gates plus $\tilde{O}(s(n))$ $\Sigma_3\mathsf{SAT}^A$ gates of fan-in $\tilde{O}(\ell(n))$. The size bound of Theorem 4.1.1 follows, and the replacement does not affect the depth of our $\mathbf{AC}$ circuit by more than a constant factor.

4.3.1 Other Compression Problems

Other compression problems similar to MCSP are amenable to the same kind of circuit construction as Theorem 4.1.1. The most obvious example is $\mathsf{MKTP}^A[s(n)]$, which has an oracle circuit construction that is nearly identical to that of Theorem 4.1.1 using instead oracle calls to the following intermediate problem which can (as before) be reduced to $\Sigma_3\mathsf{SAT}^A$.

Problem: $\mathsf{KT\text{-}Min\text{-}Merge}^A[p(N)]$

Given: Descriptions of A-oracle machines $M_1, \ldots, M_s$ with $\lceil \log(n) \rceil$ inputs and one output, a list of integers $(a_1, b_1), \ldots, (a_s, b_s) \in [N] \times [N]$ such that $a_i < b_i \leq a_{i+1}$ for all i.

Output: ***Either*** a string $1\langle M' \rangle$ where M' is the lexigraphically first A-oracle Machine M' of minimum KT complexity such that for all $x \in [n]$ and all i, if $a_i \leq x < b_i$ then $M'(x) = M_i(x)$ ***or*** a string of length $p(N)$ containing only the character 0 when there is no such machine.

Using this, we get a theorem parallel to Theorem 4.1.1, replacing MCSP^A with

MKTP^A.

Theorem 4.3.2. *Let $p(N) \geq \log(N)$ and let $\ell(N) \geq p(N)^2$ for all N, where both are time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a uniform* **AC** *circuit family for $\mathsf{MKTP}^A[p(N)]$ of size $\tilde{O}(N \cdot p(N)^2)$ and depth $O(\log(N)/\log \ell(\log(N)))$ with $\Sigma_3\mathsf{SAT}^A$ oracle gates, where each oracle gate takes only $\ell(\log(N)) \cdot poly(\log\log(N))$ bits of input.*

Note that from Theorem 4.3.2, it follows that there are even analogous circuits computing the Kolmogorov complexity of a string, though this case is less interesting as we already know that the corresponding oracle will have to be undecidable. Another more interesting example is MKtP^A, which is constructed again as above but using oracles for the following intermediate problem.

Problem: Kt-Min-Merge$[p(N)]$

Given: A-oracle machines $M_1, \ldots, M_s$ with N inputs and one output, a list of integers $(a_1, b_1), \ldots, (a_s, b_s) \in [N] \times [N]$ such that $a_i < b_i \leq a_{i+1}$ for all i.

Output: ***Either*** a string $1\langle M' \rangle$ where M' is the lexigraphically first A-oracle machine M' of minimum Kt complexity such that for all $x \in [N]$ and all i, if $a_i \leq x < b_i$ then $M'(x) = M_i(x)$ ***or*** a string of length $p(N)$ containing only the character 0 when there is no such machine.

Observing that Kt-Min-Merge can naively be computed in time $2^{O(s(n))}$, we again find an analogous circuit for MKtP.

Theorem 4.3.3. *Let $p(N) \geq \log(N)$ and let $\ell(N) \geq p(N)^2$ for all N, where both are time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a language $B \in \mathbf{TIME}[2^{O(p(N))}]^A$ such that there is a uniform* **AC** *circuit family for $\mathsf{MKtP}^A[p(N)]$ of $\tilde{O}(N \cdot p(N)^2)$ size and $O(\log(N)/\log \ell(\log(N)))$ depth with B oracle gates, where each oracle gate takes only $\ell(\log(N)) \cdot poly(\log\log(N))$ bits of input.*

Note that the parameters of the circuits have to be changed in a straightforward way as an instance of MCSP^A has an input length of 2^n, while the input lengths to our other compression problems is simply N.

4.4 Streaming Algorithm for MCSP

We now turn to the streaming algorithm for search-MCSP.

Reminder of Theorem 4.1.2. *Let $s(n) \geq n$ for all n, where $s(n)$ is time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a (one-pass) streaming algorithm for search-MCSP$^A[s(n)]$ on 2^n-bit inputs running in $2^n \cdot \tilde{O}(s(n))$ time with $\tilde{O}(s(n)^2)$ update time and $\tilde{O}(s(n))$ space, using an oracle for Σ_3SAT^A with queries of length $\tilde{O}(s(n))$.*

Here we present an oracle reduction from the MCSP$^A[s(n)]$ problem to a problem related to Circuit-Min-Merge$^A[s(n)]$.

Let $x_1, \ldots, x_{2^n}$ be the list of n-bit strings in lexicographical order.

Problem: Stream-Merge$^A[s(n)]$

Given: A t-bit description of an A-oracle circuit C with n inputs and one output where $t = 100s(n)\log s(n)$, an $x_i \in \{0,1\}^n$, and a t-bit string $y = y_1 \cdots y_t$.

Output: ***Either*** the string $1\langle C'\rangle$ where C' is the lexicographically first minimum-size A-oracle circuit of size at most $s(n)$ such that for all strings $x < x_i$ we have $C'(x) = C(x)$, and for all $j = 1, \ldots, t$, $C'(x_{i+j-1}) = y_j$, ***or*** the all-0 string of length t when there is no such circuit C'.

In other words, Stream-Merge takes a circuit C as input, an input x_i, and t extra bits of a truth table, and tries to output a small circuit that agrees with C on all inputs less than x_i, and agrees with the t extra bits on the t inputs $x_i, \ldots, x_{i+t-1}$. As with Circuit-Min-Merge^A in Section 4.2.1 and Theorem 4.1.1, Stream-Merge$^A[s(n)]$ can be efficiently reduced to Σ_3SAT^A queries of length $\tilde{O}(s(n))$.

Proof. Algorithm 1 presents the description of the streaming algorithm. Let $t = 100s(n)\log s(n)$. We start by building a circuit C_0 of size at most $s(n)$ that is consistent with the first t bits of the given truth table T, using $O(s(n)\log s(n))$-length queries to the Stream-Merge^A oracle. We repeat this process for each successive block of t bits of the input, attempting to generate a new circuit C_{i+1} which is consistent

with both C_i and the new block of the input. If at any step we fail to generate a circuit C_i, we report that there is no circuit of size $s(n)$. Otherwise, at the end we print a circuit $C_{2^n/t}$ whose truth table is the input.

Algorithm 1: The Streaming Algorithm

```
Given a truth table T of size 2^n = N.
Let C be a trivial circuit for the constant function 0 on n bits.
for i ← 1, ..., 2^n/t do
    Let y_1, ..., y_t be the next t bits of T.
    C ← Stream-Merge^A[s(n)](C, x_{(i-1)·t+1}, y_1 ··· y_t).
    if C is not a valid circuit, then
        report that there is no A-oracle circuit of size s(n).
Report C as an A-oracle circuit of size at most s(n) computing T.
```

The proof that this algorithm computes MCSP^A is similar to the proof of correctness for the circuit of Theorem 4.1.1. The correctness of Algorithm 1 follows by induction. If T has an A-oracle circuit C of size $s(n)$, then C can serve as a witness for every $\mathsf{Stream\text{-}Merge}^A$ query in the algorithm: in iteration i of the algorithm, if C_{i-1} is a circuit of size $s(n)$ consistent with T on the first $(i-1)\cdot t$ bits, then the $\mathsf{Stream\text{-}Merge}^A$ call outputs a circuit C_i consistent with T on the first $i\cdot t$ bits, if such a circuit exists. A circuit C of $s(n)$-size for T will be consistent with T on these bits for any i, which means the minimum circuit C_{i+1} output by $\mathsf{Stream\text{-}Merge}^A$ has size at most $s(n)$. By induction, if T has an A-oracle circuit of size at most $s(n)$, then Algorithm 1 will produce this circuit and report that there is such a circuit.

On the other hand, if there is no such circuit for T, then there is some $i = 1, \ldots, 2^n/t$ such that there is a size-$s(n)$ circuit consistent with T on its first $(i-1)\cdot t$ bits, but there is no size-$s(n)$ circuit consistent with T on its first $i\cdot t$ bits. (Such a circuit exists trivially for $i = 1$, but by assumption no such circuit exists for $i = 2^n/t$.) Let i' be the first such index. For all iterations before this i', $\mathsf{Stream\text{-}Merge}^A$ successfully outputs a circuit of size $s(n)$, but in iteration i' $\mathsf{Stream\text{-}Merge}^A$ fails to find such a circuit, and thus outputs $0^{100s(n)\log s(n)}$. The algorithm will notice this output is not a valid circuit, and report there is no A oracle circuit of size $s(n)$.

It remains to show that Algorithm 1 uses small time and space. In each iteration

of the for-loop, we read t more bits of T, query the $\mathsf{Stream\text{-}Merge}^A$ oracle, and check that the t bits of query answer encodes a valid circuit. Each of these tasks can be done in $O(t)$ time. Therefore each iteration can be done in $O(t)$ time and space. Over $2^n/t$ iterations, the overall resource consumption is $O(2^n)$ time and $O(t) \leq O(s(n) \log s(n))$ space, when we have access to a $\mathsf{Stream\text{-}Merge}^A$ oracle.

When we only have access to a $\Sigma_3\mathsf{SAT}^A$ oracle instead, each $\mathsf{Stream\text{-}Merge}^A$ call of length t can be converted into $O(t)$ sequential calls of length $\tilde{O}(t)$ to $\Sigma_3\mathsf{SAT}^A$, computable in $\tilde{O}(t)$ time each. Thus when we have a $\Sigma_3\mathsf{SAT}^A$ oracle, the running time is $2^n \cdot \tilde{O}(t)$, the worst-case update time is $\tilde{O}(t^2)$ (between the reading of a bit from T in one iteration to the next, we have to make $O(t)$ calls of length $\tilde{O}(t)$), and the space usage is $\tilde{O}(t)$. □

Note the proof of Theorem 4.1.2 yields the following somewhat tighter result: a linear-time streaming algorithm with an appropriate oracle B.

Theorem 4.4.1. *Let $s(n) \geq n$ for all n, where $s(n)$ is time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. Let $t(n) = 100s(n) \log s(n)$. There is an oracle $B_n : \{0,1\}^{t(n)} \to \{0,1\}^{t(n)}$ whose output bits are computable in* **PH**, *and a (one-pass) streaming algorithm for $\mathsf{search\text{-}MCSP}^A[s(n)]$ on 2^n-bit inputs running in $O(2^n)$ time and $O(t(n))$ space, using an oracle for B_n with queries of length $t(n)$.*

4.4.1 Other Compression Problems

Theorem 4.1.2 readily applies to compression problems other than MCSP. For example, to model MKTP, we simply have to modify the definition of Stream-Merge in Theorem 4.1.2 so that, rather than taking as input a size-$s(n)$ circuit C representing the initial segment of a string, we take in a Turing machine M with KT complexity at most $p(N)$ (as in KT-Min-Merge in Theorem 4.3.2). Thus we can conclude Theorem 4.4.2:

Theorem 4.4.2. *Let $p(N) \geq \log(N)$ for all n, where $p(N)$ is time constructible. Let $A : \{0,1\}^\star \to \{0,1\}$ be an arbitrary oracle. There is a streaming algorithm for*

MKTP$^A[p(N)]$ *running in* $O(N \cdot p(N))$ *time and* $O(p(N))$ *space, using an oracle for* $\Sigma_3 SAT^A$ *with queries of length at most* $O(p(N))$.

Similarly, if we modify the definition of Stream-Merge so that the size-$s(n)$ circuit C is replaced by a Turing machine M with Kt complexity at most $p(N)$ (as in Kt-Min-Merge in Theorem 4.3.3), we obtain a similar streaming algorithm for $\mathsf{MKtP}^A[p(N)]$.

Theorem 4.4.3. *Let* $p(N) \geq \log(N)$ *for all* N, *where* $p(N)$ *is time constructible. Let* $A : \{0,1\}^\star \to \{0,1\}$ *be an arbitrary oracle. There is a streaming algorithm for* *MKtP*$^A[p(N)]$ *running in* $O(N \cdot p(N))$ *time and* $O(p(N))$ *space, using an oracle for* $\mathbf{TIME}[2^{O(p(N))}]^A$ *with queries of length at most* $O(p(N))$.

As in the case of the oracle circuit (Theorem 4.3.3), the parameters of Algorithm 1 have to be changed in a straightforward way to accommodate the fact that we have described the instances of MCSP^A as length 2^n strings, while we are describing instances of other compression problems as length n strings.

4.5 Consequences

We now present some consequences of the oracle streaming algorithm of Section 4.4 and the oracle circuits constructed in Section 4.3.

Reminder of Theorem 4.1.3. *Let* $s(n) \geq n$ *be time constructible. If there is an* $A \in \mathbf{PH}$ *such that* *search-MCSP*$^A[s(n)]$ *is not solvable by a poly*$(s(n))$*-space streaming algorithm with poly*$(s(n))$ *update time, then* $\mathbf{P} \neq \mathbf{NP}$.

Proof. We show that if $\mathbf{P} = \mathbf{NP}$, then such a streaming algorithm for $\mathsf{MCSP}^A[s(n)]$ exists for all $A \in \mathbf{PH}$.

Suppose $\mathbf{P} = \mathbf{NP}$. Then the entire polynomial hierarchy collapses to $\mathbf{P}$, which means that for any possible $A \in \mathbf{PH}$, $\Sigma_3\mathsf{SAT}^A$ (and by extension $\mathsf{Stream\text{-}Merge}^A$) can be solved in polynomial time. Taking our streaming algorithm from Theorem 4.1.2, every query to $\mathsf{Stream\text{-}Merge}^A[s(n)]$ can be replaced by some $\mathrm{poly}(s(n))$ time computation (all queries have length $O(s(n) \log s(n))$). As a result, we obtain (for any

$A \in \mathbf{PH}$) a poly$(s(n))$-space streaming algorithm that takes poly$(s(n))$ update time, completing the proof. □

Reminder of Theorem 4.1.4. *Let $s(n) \geq n$, and let $A \in \mathbf{PH}$.*

- *If there is an $\varepsilon > 0$ such that for every $c \geq 1$, the problem* search-MCSP$^A[2^{\varepsilon n/c}]$ *on inputs of length $N = 2^n$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{NP} \not\subset \mathbf{TC}^0$.*[6]
- *If* search-MCSP$^A[s(n)]$ *on inputs of length $N = 2^n$ does not have circuits of $N \cdot poly(s(n))$ size and $O(\log N)$ depth, then $\mathbf{NP} \not\subset \mathbf{NC}^1$.*
- *If* search-MCSP$^A[s(n)]$ *on inputs of length $N = 2^n$ does not have circuits of $N \cdot poly(s(n))$ size and $poly(s(n))$ depth, then $\mathbf{NP} \not\subset \mathbf{P}/poly$.*

Proof. Again, we prove these results by contrapositive. Let $\mathcal{C} \in \{\mathbf{TC}^0, \mathbf{NC}^1, \mathbf{P}/poly\}$. Since all circuit classes in $\mathcal{C}$ are closed under complement, $\mathbf{NP} \subset \mathcal{C}$ implies the entire polynomial hierarchy collapses to $\mathcal{C}$. This would imply Circuit-Min-Merge$^A[s(n)] \in \mathcal{C}$. As a result, each copy of the Circuit-Min-Merge gate in the circuit constructed in Lemma 4.3.1 can be replaced with a $\mathcal{C}$-circuit of poly$(\ell(n))$ size (where $\ell(n)$ is the length of the queries to Circuit-Min-Merge). The proper $s(n)$ or $\ell(n)$ will then yield a circuit of the desired size and depth computing MCSP$^A[s(n)]$.

• Let $\mathcal{C} = \mathbf{TC}^0$. For $s(n) = N^{\varepsilon/c}$ and $\ell(n) = N^{2\varepsilon/c}$, Lemma 4.3.1 gives a circuit of size $O(N^{1+\varepsilon/c})$ and depth $O(\log N / \log N^{2\varepsilon/c}) = O(c/\varepsilon)$; replacing the oracle gates with poly$(N^{2\varepsilon/c})$ size $\mathbf{TC}^0$ circuits gives a circuit of size $O(N \cdot \text{poly}(N^{\varepsilon/c}))$ and depth $O(c/\varepsilon)$. Since this circuit exists for every $c \geq 1$, set c large enough to obtain a $\mathbf{TC}^0$ circuit of $N^{1+\varepsilon}$-size and $O(1/\varepsilon)$-depth.

• For $\mathcal{C} = \mathbf{NC}^1$ and $\ell(n) = s(n)^3$, Lemma 4.3.1 gives an $O(N)$-size circuit of depth $O(\log N / \log s(n))$. Replacing the oracle gates with poly$(s(n))$ size $\mathbf{NC}^1$ circuits gives a circuit of size $N \cdot \text{poly}(s(n))$ and depth $O((\log N / \log s(n)) \cdot \log s(n)) = O(\log N)$.

• Let $\mathcal{C} = \mathbf{P}/poly$. For $\ell(n) = s(n)^3$, Lemma 4.3.1 again gives a circuit of size $O(N)$ and depth $O(\log N / \log s(n))$. Replacing the oracle gates with poly$(s(n))$ size circuits

[6] Note that $\mathbf{TC}^0$ could be substituted with any other constant-depth circuit family, such as $\mathbf{AC}^0$[6].

gives a circuit of size $N \cdot \text{poly}(s(n))$ and depth $O((\log N / \log s(n)) \cdot \text{poly}(s(n))) = \text{poly}(s(n))$.

This completes the proof. □

Now we turn to proving hardness magnification consequences for (harder) oracle versions of MCSP.

Reminder of Theorem 4.1.5. *[Magnifying Streaming Lower Bounds for Harder MCSP Versions] Let $\mathcal{C}$ be in $\{\mathbf{NP}, \mathbf{PP}, \mathbf{PSPACE}\}$. Suppose there is a constructible $s(n)$ and oracle $A \in \mathcal{C}$ such that for all $c \geq 1$, search-MCSP$^A[s(n)]$ on inputs of length 2^n has no $s(n)^c$-space streaming algorithm with update time $s(n)^c$. Then $\mathbf{P} \neq \mathcal{C}$.*

Proof. Suppose $\mathbf{P} = \mathcal{C}$. To prove the contrapositive, we wish to show that for all constructible functions $s(n)$ and oracles $A \in \mathcal{C}$, there exists a $c \geq 1$ such that MCSP$^A[s(n)]$ has an $s(n)^c$-space streaming algorithm with update time $s(n)^c$.

Since $\mathbf{NP} \subseteq \mathcal{C}$, we know that $\mathbf{P} = \mathbf{NP}$, so again the polynomial hierarchy collapses to $\mathbf{P}$, and since $A \in \mathbf{P}$ as well Stream-Merge^A can be solved in polynomial time. Similar to Theorem 4.1.3, we can replace the oracle in the streaming algorithm with a deterministic algorithm running in $s(n)^c$ time for some constant c on inputs of length $O(s(n) \log s(n))$. So Theorem 4.1.2 gives us an $s(n)^c$-space streaming algorithm with update time $s(n)^c$. □

Reminder of Theorem 4.1.6. *[Magnifying Low-Depth Circuit Lower Bounds for Harder MCSP] Let $\mathcal{C}$ be in $\{\mathbf{NP}, \mathbf{PP}, \mathbf{PSPACE}, \mathbf{EXP}\}$. Suppose there is some $s(n)$ and oracle $A \in \mathcal{C}$ such that for all $c \geq 1$, search-MCSP$^A[s(n)]$ on inputs of length 2^n has no circuits of depth $O(n)$ and $2^n \cdot s(n)^c$ size. Then $\mathcal{C}$ does not have polynomial-size formulas (i.e., $\mathcal{C} \not\subset \mathbf{NC}^1$).*

Proof. Suppose $\mathcal{C}$ has polynomial-size formulas. To prove the contrapositive, we construct MCSP$^A[s(n)]$ circuits of depth $O(n)$ and $2^n \cdot \text{poly}(s(n))$ size.

Since $\mathbf{NP} \subseteq \mathcal{C}$, we can construct polynomial-size formulas for any problem in $\mathbf{PH}^A$, including Circuit-Min-Merge^A. Take the oracle circuit family constructed in Lemma 4.3.1 of $O(2^n \cdot s(n)/\ell(n))$ size and $O(n/\log \ell(n))$ depth. Then, replace each

$\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ gate with a $\text{poly}(\ell(n))$ size formula, which will blow up both the size and depth, but not by much. There are $O(2^n)$ copies of this $\mathsf{Circuit\text{-}Min\text{-}Merge}^A$ formula, so the size of the resulting circuit is about $O(2^n \cdot \text{poly}(\ell(n))$. Each of these formulas will have depth $O(\log(\ell(n)))$ as well, which will increase the depth of the circuit to $O(n/\log \ell(n) \cdot \log \ell(n)) = O(n)$. So if we set $\ell(n) = \text{poly}(s(n))$, we obtain a circuit of depth $O(n)$ and size $2^n \cdot \text{poly}(s(n))$ size computing $\mathsf{MCSP}^A[s(n)]$ on 2^n-bit inputs, which completes the proof. □

4.5.1 Other Compression Problems

Most of the above theorems are consequences of the existence of the circuits and streaming algorithm as given by Theorems 4.1.1 and 4.1.2. Because these circuits and streaming algorithms exist for MKTP and MKtP as per Theorems 4.3.2, 4.3.3, 4.4.2, and 4.4.3, we can conclude analogues of many of the same results as above.

Reminder of Theorem 4.1.7. *[Consequences for MKTP and MKtP] Let the function $p(N) \geq \log(N)$ be time constructible.*

- *If $\mathsf{MKTP}[p(N)]$ is not solvable by a $\text{poly}(p(N))$-space streaming algorithm with $\text{poly}(p(N))$ update time, then $\mathbf{P} \neq \mathbf{NP}$.*
- *If there is an $\varepsilon > 0$ such that for all $c \geq 1$, $\mathsf{MKTP}[N^{\varepsilon/c}]$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{NP} \not\subset \mathbf{TC}^0$.*
- *If $\mathsf{MKTP}[p(N)]$ does not have $N \cdot \text{poly}(p(N))$-size $O(\log(N))$-depth circuits, then $\mathbf{NP} \not\subset \mathbf{NC}^1$.*
- *If $\mathsf{MKTP}[p(N)]$ does not have $N \cdot \text{poly}(p(N))$-size $\text{poly}(p(N))$-depth circuits, then $\mathbf{NP} \not\subset \mathbf{P}/poly$.*
- *If $\mathsf{MKtP}[p(N)]$ does not have $N \cdot \text{poly}(p(N))$-size $\text{poly}(p(N))$-depth circuits, then $\mathbf{EXP} \not\subset \mathbf{P}/poly$.*
- *If $\mathsf{MKtP}[p(N)]$ does not have $N \cdot \text{poly}(p(N))$-size $O(\log(N))$-depth circuits, then $\mathbf{EXP} \not\subset \mathbf{NC}^1$.*

- *If there is an $\varepsilon > 0$ such that for all $c \geq 1$, $\mathsf{MKtP}[N^{\varepsilon/c}]$ does not have $N^{1+\varepsilon}$-size $O(1/\varepsilon)$-depth $\mathbf{TC}^0$ circuits, then $\mathbf{EXP} \not\subset \mathbf{TC}^0$.*

We omit the proof of Theorem 4.1.7, as each claim follows easily from the arguments given above for the analogous claims about MCSP^A in Theorems 4.1.4, 4.1.5, and 4.1.6 citing instead Theorems 4.3.2, 4.3.3, 4.4.2, and 4.4.3 for the existence of efficient oracle circuits and streaming algorithms for MKTP and MKtP. Relativized versions of Theorem 4.1.7 also hold for MKTP^A and MKtP^A with an oracle A; we leave the details to the interested reader.

4.6 Conclusion

We conclude this chapter with a few related open problems.

- **What is the moral of this story?** What lessons do we draw from these results? As we stated in the introduction, it does not seem right to call our results a "barrier" to proving weak time-space lower bounds, because we believe all of the consequential lower bounds of this chapter! Still, there ought to be more consequences of the fact that certain "weak-looking" lower bound problems are deceptive, and in fact are much stronger than they appear.

- **Bounded Nondeterminism Problems?** A key property of $\mathsf{MCSP}[s(n)]$ (and related compression problems) is that the nondeterminism needed to solve the problem is only $O(s(n) \log s(n))$ bits: the number of bits needed to write down a circuit of size $s(n)$. When $s(n) \leq 2^{o(n)}$, $\mathsf{MCSP}[s(n)]$ is a problem with *sub-linear nondeterminism*. It is natural to ask whether hardness magnification holds for similar problems. For example, consider the SAT problem with $s(n)$ variables and 2^n clauses. Are there interesting consequences of proving weak time lower bounds for such SAT problems, for algorithms using $\mathrm{poly}(s(n))$ space?

- **Truth tables presented differently?** Our main results for MCSP rely on the input truth table of f being presented in a canonical way, namely as a 2^n-bit

string

$$f(x_1) \cdots f(x_{2^n}),$$

where $x_1, \ldots, x_{2^n}$ is the list of n-bit strings in lexicographical order. Our results can also extend to the case of other efficiently-computable orderings on strings. What about when the truth table is presented in an *arbitrary* order, as a list of pairs

$$(x_1, f(x_1)), \ldots, (x_{2^n}, f(x_{2^n}))$$

where $x_1, \ldots, x_{2^n}$ is an arbitrary permutation of $\{0,1\}^n$? Can similar hardness magnification results be proved for this version of the problem? Note that, in this representation, each Boolean function corresponds to $(2^n)!$ distinct strings of length $\Theta(n2^n)$, so the underlying language $\mathsf{MCSP}[s(n)]$ is no longer as sparse as it used to be.

Chapter 5

An Equivalence Between Fixed-Polynomial Circuit Size Lower Bounds and Karp-Lipton Style Theorems

5.1 Introduction

Let $\mathcal{C}$ be a complexity class containing $\mathbf{NP}$. A longstanding method for proving fixed-polynomial circuit lower bounds for functions in $\mathcal{C}$, first observed by Kannan [62], applies versions of the classical Karp-Lipton Theorem by splitting the proof into two cases:

1. If $\mathbf{NP} \not\subset \mathbf{P}/poly$, then $\mathsf{SAT} \in \mathbf{NP} \subset \mathcal{C}$ does not have polynomial-size circuits.

2. If $\mathbf{NP} \subset \mathbf{P}/poly$, then by a "collapse" theorem, we have $\mathbf{PH} \subseteq \mathcal{C}$. But for every k, there is an $f \in \mathbf{PH}$ that does not have n^k-size circuits, so we are also done in this case.

Such collapse theorems are called *Karp-Lipton Theorems*, as they were first discovered by Karp and Lipton [63] in their pioneering work on complexity classes with

advice. The general theme of such theorems is a connection between non-uniform and uniform complexity:

> "$\mathcal{C}$ has (non-uniform) polynomial-size circuits implies a collapse of (uniform) complexity classes."

Over the years, Karp-Lipton Theorems have been applied to prove circuit lower bounds for the complexity classes $\mathbf{NP}^{\mathbf{NP}}$ [62], $\mathbf{ZPP}^{\mathbf{NP}}$ [26, 64], $\mathsf{S}_2\mathsf{P}$ [30, 31], **PP** [89, 1][1], and Promise-**MA** and **MA**/1 [81].[2] Other literature on Karp-Lipton Theorems include [92, 33, 34].

When one first encounters such a lower bound argument, the non-constructivity of the result (the two uncertain cases) and the use of a Karp-Lipton Theorem look strange.[3] It appears obvious that one ought to be able to prove circuit lower bounds in a fundamentally *different way*, without worrying over any collapses of the polynomial hierarchy. It is easy to imagine the possibility of a sophisticated combinatorial argument establishing a lower bound for $\mathbf{P}^{\mathbf{NP}}$ functions (one natural next step in such lower bounds) which has nothing to do with simulating **PH** more efficiently, and has no implications for it.

$\mathbf{P}^{\mathbf{NP}}$ Circuit Lower Bounds are Equivalent to Karp-Lipton Collapses to $\mathbf{P}^{\mathbf{NP}}$. We show that, in a sense, the above intuition is **false**: any fixed-polynomial-size circuit lower bound for $\mathbf{P}^{\mathbf{NP}}$ would imply a Karp-Lipton Theorem collapsing **PH** all the way to $\mathbf{P}^{\mathbf{NP}}$. (There are some technicalities: the $\mathbf{P}^{\mathbf{NP}}$ simulation uses small advice and only works infinitely often, but we believe these conditions can potentially be removed, and they do not change the moral of our story.) We find this result surprising; it shows that *in order to prove a circuit lower bound for* $\mathbf{P}^{\mathbf{NP}}$*, one cannot avoid proving a Karp-Lipton Theorem for* $\mathbf{P}^{\mathbf{NP}}$ *in the process.* A Karp-Lipton Theorem is both necessary and sufficient for such lower bounds.

[1] Both Vinodchandran and Aaronson's proofs of $\mathbf{PP} \not\subset \mathbf{SIZE}[n^k]$ use the Karp-Lipton-style theorem "$\mathbf{PP} \subset \mathbf{P}/poly$ then $\mathbf{PP} = \mathbf{MA}$", which follows from [68]. Aaronson shows further that "$\mathbf{PP} \subset \mathbf{P}/poly$ then $\mathbf{P}^{\mathbf{PP}} = \mathbf{MA}$". From there, one can directly construct a function in $\mathbf{P}^{\mathbf{PP}}$ without n^k-size circuits.

[2] Santhanam used the Karp-Lipton-style theorem "$\mathbf{PSPACE} \subset \mathbf{P}/poly$ implies $\mathbf{PSPACE} = \mathbf{MA}$" to prove lower bounds against Promise-**MA** and **MA** with one bit of advice.

[3] Note Cai and Watanabe [32] found a constructive proof for $\mathbf{NP}^{\mathbf{NP}}$.

Reminder of Theorem 1.0.7 ($\mathbf{P^{NP}}$ Circuit Lower Bounds are Equivalent to a Karp-Lipton Collapse to $\mathbf{P^{NP}}$).
$\mathbf{P^{NP}} \not\subset \mathbf{SIZE}[n^k]$ *for all* k *if and only if* ($\mathbf{NP} \subset \mathbf{P}/poly \implies \mathbf{PH} \subset$ *i.o.*-$\mathbf{P^{NP}}/n$).

One direction of Theorem 1.0.7 follows immediately from the classical lower bound paradigm described above. In particular, assuming $\mathbf{P^{NP}} \subset \mathbf{SIZE}[n^k]$ for some k and assuming $\mathbf{NP} \subset \mathbf{P}/poly \implies \mathbf{PH} \subset$ i.o.-$\mathbf{P^{NP}}/n$ we have

$$\mathbf{PH} \subset \text{i.o.-}\mathbf{P^{NP}}/n \subseteq \text{i.o.-}\mathbf{SIZE}[O(n)^k],$$

which contradicts known fixed-polynomial lower bounds for **PH**. The interesting direction is the converse, showing that *proving lower bounds against* $\mathbf{P^{NP}}$ **implies** *proving a Karp-Lipton collapse to* $\mathbf{P^{NP}}$ *that is sufficient for the lower bound.*

NP Circuit Lower Bounds Imply Better Karp-Lipton Collapses. After observing Theorem 1.0.7, a natural question is whether such a theorem holds for **NP** circuit lower bounds as well:

Does $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ *for all* k *imply a Karp-Lipton Collapse to* **NP**?

While we have not yet been able to prove this under the hypothesis $\mathbf{NP} \subset \mathbf{P}/poly$ as above, we can show it for stronger hypotheses. Another class of Karp-Lipton Theorems (used in circuit lower bounds for **PP** [89, 1] and Promise-**MA** [81]) give stronger collapses under hypotheses like $\mathbf{PSPACE} \subset \mathbf{P}/poly$: for any class $\mathcal{C}$ which is one of **NEXP** [60], $\mathbf{EXP^{NP}}$ ([27] and [15]), **EXP** and **PSPACE** [15], **PP** [68] and $\oplus\mathbf{P}$ [59], we have:

If $\mathcal{C} \subset \mathbf{P}/poly$ then $\mathcal{C} \subseteq \mathbf{MA}$.

We show how **NP** circuit lower bounds can be used to derandomize **MA**. In fact, under the hypothesis $\mathbf{NP} \subset \mathbf{P}/poly$, we prove an equivalence between **NP** circuit lower bounds, fast Arthur-Merlin simulations of **NP**, and nondeterministic derandomization of Arthur-Merlin protocols.

To state our results, we first define a variation of the "robust simulation" which was originally introduced in [45]. For a complexity class $\mathcal{C}$ and a language L, we say

L is in c-r.o.-$\mathcal{C}$ for a constant c, if there is a language $L' \in \mathcal{C}$ such that there are infinitely many m's such that for all $n \in [m, m^c]$, L' agrees with L on inputs of length n.[4] (See Section 5.2.1 for formal definitions.)

Theorem 5.1.1. *Theorem Assuming* $\mathbf{NP} \subset \mathbf{P}/poly$*, the following are equivalent:*

1. $\mathbf{NP}$ *is not in* $\mathbf{SIZE}[n^k]$ *for all* k*.*

2. $\mathbf{AM}/1$ *is in c-r.o.-*$\mathbf{NP}/n^\varepsilon$ *for all* $\varepsilon > 0$ *and integers* c*.*
   That is, Arthur-Merlin games with $O(1)$ *rounds and small advice can be simulated "c-robustly often" in* $\mathbf{NP}$ *with modest advice, for all constants* c*.*[5]

3. $\mathbf{NP}$ *does not have* n^k*-size witnesses for all* k*.*
   That is, for all k*, there is a language* $L \in \mathbf{NP}$*, a poly-time verifier* V *for* L*, and infinitely many* $x_n \in L$ *such that* $V(x_n, \cdot)$ *has no witness of circuit complexity at most* n^k*.*

4. *For all* k *and* d*, there is a polynomial-time nondeterministic PRG with seed-length* $O(\log n)$ *and* n *bits of advice against* n^k*-size circuits* d*-robustly often.*[6]

5. $\mathbf{NP}$ *is not in* $\mathbf{AMTIME}(n^k)$ *for all* k*.*

6. $(\mathbf{NP} \cap \mathbf{coNP})/n^\varepsilon$ *is not in* $\mathbf{SIZE}[n^k]$ *for all* k *and all* $\varepsilon > 0$*.*

7. $(\mathbf{AM} \cap \mathbf{coAM})/1$ *is in c-r.o.-*$(\mathbf{NP} \cap \mathbf{coNP})/n^\varepsilon$ *for all* $\varepsilon > 0$ *and all integers* c*.*

That is, under $\mathbf{NP} \subset \mathbf{P}/poly$, the tasks of fixed-polynomial lower bounds for $\mathbf{NP}$, lower bounds for $(\mathbf{NP} \cap \mathbf{coNP})/n^\varepsilon$, uniform lower bounds on simulating $\mathbf{NP}$ within $\mathbf{AM}$, and derandomizing $\mathbf{AM}$ in $\mathbf{NP}$ are all equivalent.

We recall another type of Karp-Lipton collapse was shown by [14]: $\mathbf{NP} \subset \mathbf{P}/poly$ implies $\mathbf{AM} = \mathbf{MA}$. An intriguing corollary of Theorem 5.1.1 is that fixed-polynomial

[4] The original definition of $L \subseteq$ r.o.-$\mathcal{C}$ requires that there is a *single* language $L' \in \mathcal{C}$ such that for all c there are infinitely many m's such that for all $n \in [m, m^c]$, L' agrees with L on inputs of length n.

[5] See the preliminaries section of this chapter (Section 5.2) for a definition of "c-robustly often". Intutively, it is a mild strengthening of "infinitely often".

[6] See the preliminaries section of this chapter (Section 5.2) for formal definitions.

lower bounds for **NP** would improve this collapse, from **MA** to c-r.o.-**NP**/n^ε for all c:

Reminder of Corollary 1.0.9 (NP Circuit Lower Bounds Equivalent to a Karp-Lipton Collapse of AM to NP). **NP** $\not\subset$ **SIZE**$[n^k]$ *for all* k *if and only if* (**NP** $\subset$ **P**/*poly* $\implies$ **AM** *is in* c*-r.o.-***NP**/n^ε *for all* c*).*

Another consequence of Theorem 5.1.1 is that **NP** circuit lower bounds imply better Karp-Lipton collapses from **MA** down to **NP**:

Theorem 5.1.2 (NP Circuit Lower Bounds Imply Better Karp-Lipton Collapses)**.** *Let* $\mathcal{C} \in \{\oplus\mathbf{P}, \mathbf{PSPACE}, \mathbf{PP}, \mathbf{EXP}\}$*. Suppose* **NP** $\not\subset$ **SIZE**$[n^k]$ *for all* k*. Then for all* $\varepsilon > 0$*,* ($\mathcal{C} \subset$ **P**/*poly* $\implies$ $\mathcal{C} \subset$ *i.o.-***NP**/n^ε)*. In particular, polynomial-size circuits for any* $\mathcal{C}$*-complete language* L *can be constructed in* **NP** *infinitely often, with* n^ε *advice.*

Remark 5.1.3. *By "circuits for* L *can be constructed in* **NP** *infinitely often", we mean that there is a nondeterministic polynomial-time algorithm* A *such that, for infinitely many* n*,* A *on input* 1^n *outputs a circuit* C_n *for* L_n *on at least one computation path, and on all paths where such a* C_n *is not output,* A *outputs* reject*.*

Consequences of Weak Circuit Lower Bounds for Sparse Languages in NP. Theorem 5.1.1 shows that assuming **NP** $\subset$ **P**/*poly*, fixed-polynomial lower bounds for **NP** imply **AM** = **MA** $\subseteq$ i.o.-**NP**/n^ε. This is also the reason that we can only show collapses to i.o.-**NP**/n^ε in Theorem 5.1.2. It is interesting to ask whether the n^ε advice in the simulation can be eliminated or reduced. In the following, we show that an $n^{1.00001}$-size circuit lower bound for a polynomially-sparse language in **NP** would imply an advice reduction, along with other interesting consequences.

Theorem 5.1.4 (Consequences of Weak Circuit Lower Bounds for Polynomially-Sparse **NP** Languages)**.** *Suppose there is an* $\varepsilon > 0$*, a* $c \geq 1$*, and an* n^c*-sparse* $L \in$ **NP** *without* $n^{1+\varepsilon}$*-size circuits. Then* **MA** $\subset$ *i.o.-***NP**/$O(\log n)$*,* **MA** $\subseteq$ *i.o.-*$\mathbf{P}^{\mathbf{NP}[O(\log n)]}$*, and* **NE** $\not\subset$ **SIZE**$[2^{\delta \cdot n}]$ *for some* $\delta > 0$ *(which implies* **NP** $\not\subset$ **SIZE**$[n^k]$ *for all* k*).*

One step in the proof of Theorem 5.1.4 is a form of *hardness condensation* (as termed by Impagliazzo [57]) for sparse **NP** languages. The goal of hardness condensation [28, 58] is that, given a function f on n input bits with complexity S, we want to construct a function $\widetilde{f}$ on $\ell \ll n$ input bits that still has complexity roughly S. We show how a hard $S(n)$-sparse language in $\mathbf{NTIME}[T(n)]$ can be "condensed" in a generic way, based on the sparsity $S(n)$. We can efficiently build a PRG from the harder condensed function.

Theorem 5.1.4 shows how a very weak lower bound ($n^{1+\varepsilon}$) for a sparse language $L \in \mathbf{NP}$ would imply an *exponential-size* lower bound for **NE** (note, the converse is easy to show). This is reminiscent of a recent line of work [76, 75, 70] (and the previous chapter) on "hardness magnification" phenomena, showing that seemingly weak circuit lower bounds for certain problems can in fact imply strong circuit lower bounds which are out of reach of current proof techniques.

At a high level, the hardness magnification results in the above-cited papers show how weak lower bounds on "compression problems" can imply strong complexity class separations. These compression problems have the form: *given a string, does it have a small efficient representation?* As an example, in the Minimum Circuit Size Problem for size $S(m) \ll 2^m$, denoted as $\mathsf{MCSP}[S(m)]$, we are given a truth table of length $N = 2^m$ and want to know if the function has a circuit of size at most $S(m)$. As an example of hardness magnification, Theorem 4.1.4 of Chapter 4 shows that, if there is an $\varepsilon > 0$ such that $\mathsf{MCSP}[2^{m/\log^\star m}]$ is not in $\mathbf{SIZE}[N^{1+\varepsilon}]$, then $\mathbf{NP} \not\subset \mathbf{P}/poly$. Thus a very weak circuit size lower bound for $\mathsf{MCSP}[2^{m/\log^\star m}]$ would imply a super-polynomial lower bound for SAT!

Sparsity Alone Implies a Weak Hardness Magnification. We identify a simple property of all efficient compression problems which alone implies a (weak) form of hardness magnification: the *sparsity of the underlying language*. For any compression problem on length-N strings where we ask for a length-$\ell(N)$ representation (think of $\ell(N) \leq n^{o(1)}$), there are at most $2^{\ell(N)}$ strings in the language. Scaling up the sparsity of Theorem 5.1.4, we show that non-trivial circuit lower bounds for *any* **NP** problem

with subexponential sparsity already implies longstanding circuit lower bounds. In fact, we have an equivalence:

Reminder of Theorem 1.0.11. **NEXP** $\not\subset$ **P**/*poly if and only if there exists an* $\varepsilon > 0$ *such that for every sufficiently small* $\beta > 0$*, there is a* 2^{n^β}*-sparse language* $L \in \mathbf{NTIME}[2^{n^\beta}]$ *without* $n^{1+\varepsilon}$*-size circuits.*

It follows that an $n^{1+\varepsilon}$-size circuit lower bound for $\mathsf{MCSP}[2^{m/\log^\star m}]$ would imply **NEXP** $\not\subset$ **P**/*poly* (but we got **NP** $\not\subset$ **P**/*poly* above). We remark while the lower bound consequence here is *much weaker* than the consequences of prior work [76, 75, 70] (only **NEXP** $\not\subset$ **P**/*poly*, instead of **NP** $\not\subset$ **P**/*poly*), the hypothesis has much more flexibility: Theorem 1.0.11 allows for any sparse language in $\mathbf{NTIME}[2^{n^{o(1)}}]$, while the MCSP problem is in $\mathbf{NTIME}[n^{1+o(1)}]$.[7]

Finally, we observe that Theorem 1.0.11 is similar in spirit to the Hartmanis-Immerman-Sewelson theorem [48] which states that there is a polynomially-sparse language in **NP** $\setminus$ **P** if and only if **NE** $\neq$ **E**. Theorem 1.0.11 can be interpreted as a certain optimized, non-uniform analogue of Hartmanis-Immerman-Sewelson theorem, in a different regime of sparsity.

Organization of the Chapter. In Section 5.2, we introduce the necessary preliminaries for this chapter. In Section 5.3, we prove that fixed-polynomial circuit lower bounds for $\mathbf{P}^{\mathbf{NP}}$ is equivalent to a (weak) Karp-Lipton theorem for **P**. In Section 5.4, we prove our equivalence theorem for **NP** circuit lower bounds, fast simulations of **NP**, and nondeterministic polynomial-time derandomization, under the hypothesis **NP** $\subset$ **P**/*poly*. In Section 5.5, we show how our equivalence theorem implies that fixed polynomial circuit lower bounds for **NP** implies better Karp-Lipton theorems for higher complexity classes. In Section 5.6, we prove the consequences of weak circuit lower bounds for sparse **NP** languages. In Section 5.7, we prove the key Lemma 5.4.1 showing almost everywhere lower bounds against $\mathbf{MA} \cap \mathbf{coMA}/1$, finishing the proof of Theorem 5.1.1 from Section 5.4. Finally, in Section 5.8, we discuss some interesting open questions stemming from this work.

[7] We remark that these results are not directly related to hardness magnification for $\mathbf{NC}^1$-complete problems [12, 38], as the problems studied in these works are clearly not sparse.

5.2 Preliminaries

Here we review some notation and concepts that are of particular interest for this chapter.

5.2.1 Infinitely Often and Robust Simulations

In this section, let $\mathscr{C}$ be a class of languages. Here we recall infinitely often and robust simulations, the latter of which was first defined and studied in [45]. Robust simulations expand on the notion of "infinitely often" simulations. A language $L \in$ i.o.-$\mathscr{C}$ (*infinitely often* $\mathscr{C}$), if there is a language L' in $\mathscr{C}$ such that there are infinitely many n such that $L_n = L'_n$. A language $L \in$ r.o.-$\mathscr{C}$ (*robustly often* $\mathscr{C}$), if there is a language L' in $\mathscr{C}$ such that for all $k \geq 1$, there are infinitely many n such that $L_m = L'_m$ for all $m \in [n, n^k]$. In this case, we say L' *r.o.-computes* L.

c-Robust Simulations. We consider a parameterized version of the robust simulation concept which is useful for stating our results. Let $c \geq 1$ be an integer constant. We say a language $L \in$ c-r.o.-$\mathscr{C}$ (c-robustly often $\mathscr{C}$) if there is an $L' \in \mathscr{C}$ and infinitely many n such that $L_m = L'_m$ for all $m \in [n, n^c]$. In this case, we say L' c-r.o.-computes L. Note that $L \in$ r.o.-$\mathscr{C}$ implies $L \in$ c-r.o.-$\mathscr{C}$ for all c, but the converse is not necessarily true.

More generally, a property $P(n)$ holds c-robustly often (c-r.o.-) if for all integers k, there are infinitely many m's such that $P(n)$ holds for all $n \in [m, m^c]$.

5.2.2 Non-deterministic Pseudo-Random Generators

Let $w(n), s(n) : \mathbb{N} \to \mathbb{N}$, and let $\mathscr{C}$ be a class of functions. We say a function family G, specified by $G_n : \{0,1\}^{w(n)} \times \{0,1\}^{s(n)} \to \{0,1\}^* \cap \{\bot\}$, is a *nondeterministic PRG against* $\mathscr{C}$ if for all sufficiently large n and all $C \in \mathscr{C}$, the following hold:

- For all $y \in \{0,1\}^{w(n)}$, either $G_n(y,z) \neq \bot$ for all z's (such a y is called *good*), or $G_n(y,z) = \bot$ for all z's (a *bad* y).

- There is at least one good $y \in \{0,1\}^{w(n)}$.
- Suppose $y \in \{0,1\}^{w(n)}$ is good, C has m input bits, and $|G_n(y,z)| \geq m$ for all z. Then

$$\left| \Pr_{z \in \{0,1\}^{s(n)}}[C(G_n(y,z)) = 1] - \Pr_{z \in \{0,1\}^m}[C(z) = 1] \right| < 1/n.$$

  As usual, if C takes less than $|G_n(y,z)|$ inputs, $C(G_n(y,z))$ corresponds to feeding C with the first m bits of $G_n(y,z)$.

Usually we are only interested in the seed length parameter $s(n)$ and the running time $T(n)$ of the PRG G_n as a function of n. To be concise, we say G is a $T(n)$-*time NPRG of seed length* $s(n)$ *against* $\mathscr{C}$.

We say G is a i.o.-NPRG or r.o.-NPRG, if it only fools functions in $\mathscr{C}$ infinite often or robustly often.

5.2.3 Pseudorandom Generators from Strings of High Circuit Complexity

We will use the following strong construction of pseudorandom generators from hard functions:

Theorem 5.2.1 (Umans [87])**.** *There is a constant g and a function $G : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that, for all s and Y satisfying $\mathsf{CircuitComplexity}(Y) \geq s^g$, and for all circuits C of size s,*

$$\left| \Pr_{x \in \{0,1\}^{g \log |Y|}}[C(G(Y,x)) = 1] - \Pr_{x \in \{0,1\}^s}[C(x) = 1] \right| < 1/s.$$

Furthermore, G is computable in poly$(|Y|)$ time.

Fortnow-Santhanam-Williams [46]. A work related to this chapter is that of Fortnow, Santhanam, and Williams, who proved the equivalences $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ for all $k \iff \mathbf{P}^{\mathbf{NP}[n^k]} \not\subset \mathbf{SIZE}[n^c]$ for all k, c and $\mathbf{AM} \not\subset \mathbf{SIZE}[n^k]$ for all $k \iff$

$\mathbf{MA} \not\subset \mathbf{SIZE}[n^k]$ for all k. We use intermediate results of theirs in our equivalence theorems (see the citations).

5.3 $\mathbf{P^{NP}}$ Circuit Lower Bounds Equivalent to Karp-Lipton Collapses to $\mathbf{P^{NP}}$

In this section we prove Theorem 1.0.7 (restated below).

Reminder of Theorem 1.0.7. $\mathbf{P^{NP}} \not\subset \mathbf{SIZE}[n^k]$ *for all* k *if and only if* ($\mathbf{NP} \subset \mathbf{P}/poly \implies \mathbf{PH} \subset i.o.\text{-}\mathbf{P^{NP}}/n$).

We begin with a lemma on the simulation of poly-time functions with an **NP** oracle. Essentially the lemma says that if functions with an **NP** oracle always output strings of low circuit complexity, then we can simulate $\mathbf{P^{NP}}$ extremely efficiently in the polynomial hierarchy. This is similar in spirit to Fortnow, Santhanam, and Williams' result that $\mathbf{P^{NP}} \subset \mathbf{SIZE}[n^k]$ implies $\mathbf{NP} \subseteq \mathbf{MATIME}(n^{O(k)})$ [46]; our result is more complex in that we simulate all of $\mathbf{P^{NP}}$.

Lemma 5.3.1. *Suppose there is a* k *such that for all* $\mathbf{FP^{NP}}$ *functions* f*, the circuit complexity of* $f(x)$ *is at most* $|x|^k$ *for all but finitely many* x*. Then* $\mathbf{P^{NP}} \subseteq \Sigma_3\mathbf{TIME}[n^{O(k)}]$.

Proof. Let $L \in \mathbf{P^{NP}}$ be a language which can be computed by a 3SAT oracle machine M in n^c time, for a constant c. Without loss of generality, we may assume M is a single-tape machine.

The $\mathbf{FP^{NP}}$ Function f_{sol}. Consider the following $\mathbf{FP^{NP}}$ function f_{sol}:

> **$\mathbf{FP^{NP}}$ function f_{sol} for printing assignments to all satisfiable oracle queries**
>
> - Given an input x, simulate the 3SAT oracle machine M running on the input x.

- On the i-th step of the simulation, if M makes an oracle query ψ (ψ is a $\mathsf{3SAT}$ instance) and ψ is satisfiable, call the $\mathbf{NP}$ oracle multiple times to construct a satisfying assignment for ψ, and print it. Letting m be the length of the assignment (note that $m \leq n^c$), we print $n^c + 1 - m$ additional ones.
- Otherwise, print $n^c + 1$ zeros on the i-th step.

In the following we always assume n is sufficiently large. For all x with $|x| = n$, by assumption we know the string $f_{\mathsf{sol}}(x)$ has an n^k size circuit. Let ψ be a $\mathsf{3SAT}$ query made on i-th step which is satisfiable; ψ has a satisfying assignment corresponding to a sub-string of $f_{\mathsf{sol}}(x)$ starting from the position $(i-1)\cdot(n^c+1)+1$, and therefore has circuit complexity at most $O(n^k) \leq n^{k+1}$. In particular, we can define a circuit $E_i(j) := f_{\mathsf{sol}}(x)((i-1)\cdot(n^c+1)+j)$ whose truth table encodes a SAT assignment to ψ.

The $\mathbf{FP^{NP}}$ Function f_{history}. Next, we define a function $\mathbf{FP^{NP}}$ function f_{history}, which prints the computation history of M. More precisely, we can interpret $f_{\mathsf{history}}(x)$ as a matrix $\mathsf{cell}(x) \in \Sigma^{n^c \times n^c}$, such that $\mathsf{cell}(i,j)$ represents the state of the j-th cell of the working tape before the i-th step, and Σ is a constant-size alphabet which represents all possible states of a cell. From our assumption, for an x with $|x| = n$, we know that $f_{\mathsf{history}}(x)$ has an n^k-size circuit.

The Algorithm. Now we are ready to describe a Σ_3 algorithm for L running in $n^{O(k)}$ time. At a high level, the algorithm first guesses two circuits C_{history} and C_{sol}, whose truth-tables are supposed to represent $f_{\mathsf{history}}(x)$ and $f_{\mathsf{sol}}(x)$, it tries to verify that these circuits correspond to a correct accepting computation of M on x. The whole verification can be done in $\Pi_2\mathbf{TIME}[n^{O(k)}]$, utilizing the fact that M is making $\mathsf{3SAT}$ queries. The formal description of the algorithm is given below.

A $\Sigma_3\mathbf{TIME}[n^{O(k)}]$ algorithm for L

(1) Given an input x, guess two n^k-size circuits C_{history} and C_{sol} where the truth-table of C_{history} is intended to be $f_{\mathsf{history}}(x)$), and the truth-table of C_{sol} is intended to be $f_{\mathsf{sol}}(x)$. Let $\mathsf{cell} \in \Sigma^{n^c \times n^c}$ be the matrix (tableau) corresponding to the truth-table of C_{history}.

(2) We check that C_{history} is consistent and accepting, assuming its claimed answers to oracle queries are correct. In particular, we universally check over all $(i,j) \in [n^c] \times [n^c]$ that $\mathsf{cell}(i,j)$ is consistent with the contents of $\mathsf{cell}(i-1,j-1), \mathsf{cell}(i-1,j), \mathsf{cell}(i,j+1)$ when $i > 1$, whether it agrees with the initial configuration when $i = 1$, and whether M is in an accept state when $i = n^c$.

(3) We check that the claimed answers to oracle queries in C_{history} are correct. For convenience, we assume the query string always starts at the leftmost position on the tape. We universally check over all step $i \in [n^c]$:

If there is no query at the i-th step, we *accept*.

(A) Let ψ be the $\mathsf{3SAT}$ query. If the claimed answer in C_{history} for ψ is *yes*, we examine the corresponding sub-string of $\mathsf{TruthTable}(C_{\mathsf{sol}})$, and check universally over all clauses in ψ that it is satisfied by the corresponding assignment in $\mathsf{TruthTable}(C_{\mathsf{sol}})$ (accepting if the check passes and rejecting if it fails).

(B) If the claimed answer in C_{history} for ψ is *no*, we universally check over all n^{k+1}-size circuits D that $\mathsf{TruthTable}(D)$ is not an assignment to ψ, by existentially checking that there is a clause in ψ which is not satisfied by $\mathsf{TruthTable}(D)$.

Running Time. It is straightforward to see that the above is a $\Sigma_3\mathbf{TIME}[n^{O(k)}]$ algorithm.

Correctness. When $x \in L$, there are C_{sol} and C_{history} such that $\mathsf{TruthTable}(C_{\mathsf{sol}})$ and $\mathsf{TruthTable}(C_{\mathsf{history}})$ correspond to $f_{\mathsf{sol}}(x)$ and $f_{\mathsf{history}}(x)$, so all of the checks pass and the above algorithm accepts x.

Let $x \notin L$. We want to show that all possible n^k-size circuits for C_{history} and C_{sol} will be rejected. Assume for contradiction that there are circuits C_{history} and C_{sol} that can pass the whole verification. By our checks in step (2) of the algorithm, C_{history} is consistent and ends in accept state; therefore, at least one answer to its oracle queries is not correct. Suppose the first incorrect answer occurs on the i-th step. Since C_{history} is consistent and all queries made before the i-th are correctly answered, the i-th query ψ is actually the correct i-th query made by machine M on the input x.

Therefore, if the correct answer to ψ is yes but C_{history} claims it is no, case (B) will not be passed, as there is always a satisfying assignment that can be represented by the truth-table of an n^{k+1}-size circuit. Similarly, if C_{history} incorrectly claims the answer is yes, then case (A) cannot be passed, as ψ is unsatisfiable. □

We are now ready to prove Theorem 1.0.7.

Proof of Theorem 1.0.7. Suppose (1) $\mathbf{P}^{\mathbf{NP}}$ does not have $\mathbf{SIZE}[n^k]$ circuits for any fixed k and (2) $\mathbf{NP} \subset \mathbf{P}/poly$. By assumption (2), we have that for every c, $\Sigma_3\mathbf{TIME}[n^c] \subset \mathbf{SIZE}[n^{O(c)}]$. Therefore, applying (1), $\mathbf{P}^{\mathbf{NP}} \not\subseteq \Sigma_3\mathbf{TIME}[n^c]$ for every c. By the contrapositive of Lemma 5.3.1, for every k there is a $\mathbf{P}^{\mathbf{NP}}$ function B that for infinitely many x of length n, the circuit complexity of $B(x)$ is greater than n^k. In other words, $B(x)$ outputs the truth tables of hard functions on infinitely many x.

Assumption (2) also implies a collapse of the polynomial hierarchy to $\mathbf{ZPP}^{\mathbf{NP}}$ [64]. By (2), we also have $\mathbf{ZPP}^{\mathbf{NP}} \subset \mathbf{P}/poly$, so every $\mathbf{ZPP}^{\mathbf{NP}}$ algorithm A has polynomial-size circuits. Thus, by standard hardness-to-PRG constructions (e.g., Theorem 5.2.1) there is a fixed k such that a string of circuit complexity at least n^k can be used to construct a PRG that fools algorithm A on inputs of length n. As shown above,

there is a function B in $\mathbf{P}^{\mathbf{NP}}$ that can produce such strings on infinitely many inputs x. If the inputs x that make B produce high complexity strings are given as advice, then the $\mathbf{ZPP}^{\mathbf{NP}}$ algorithm A can be simulated in $\mathbf{P}^{\mathbf{NP}}/n$: first, call B on the advice x to generate a hard function, produce a PRG of seed length $O(\log n)$ with the hard function, then simulate A on the input and the pseudorandom strings output by the PRG, using the $\mathbf{NP}$ oracle to simulate the $\mathbf{NP}$ oracle of A. Thus we have $\mathbf{ZPP}^{\mathbf{NP}} \subset$ i.o.-$\mathbf{P}^{\mathbf{NP}}/n$.

Finally, we note that the n bits of advice can be reduced to n^ε bits for any desired $\varepsilon > 0$. For every $k > 0$, we can find an $\mathbf{FP}^{\mathbf{NP}}$ function that outputs a string of circuit complexity greater than n^k. Setting $k' = k/\varepsilon$, we can use an n^ε-length input as advice, and still get a function that is hard enough to derandomize $((n^\varepsilon)^{k'} = (n^\varepsilon)^{k/\varepsilon} = n^k)$. □

5.4 An Equivalence Theorem Under NP $\subset$ P/*poly*

In this section we prove Theorem 5.1.1 together with several applications.

First, we need a strong size lower bound for a language in $(\mathbf{MA} \cap \mathbf{coMA})/1$. The proof is based on a similar lemma in a recent work [35] (which further builds on [73, 81]). We present a proof in Section 5.7 for completeness.

Lemma 5.4.1 (Implicit in [35])**.** *For all constants k, there is an integer c, and a language $L \in (\mathbf{MA} \cap \mathbf{coMA})/1$, such that for all sufficiently large $\tau \in \mathbb{N}$ and $n = 2^\tau$, either*

- $\mathsf{SIZE}(L_n) > n^k$*, or*
- $\mathsf{SIZE}(L_m) > m^k$*, for an $m \in (n^c, 2 \cdot n^c) \cap \mathbb{N}$.*

We also need the following two simple lemmas.

Lemma 5.4.2. $\mathbf{NP}$ *is not in* $\mathbf{SIZE}[n^k]$ *for all k iff* $\mathbf{NP}/n$ *is not in* $\mathbf{SIZE}[n^k]$ *for all k.*

Proof. The $\Rightarrow$ direction is trivial. For the $\Leftarrow$ direction, suppose **NP** is in $\mathbf{SIZE}[n^k]$ for an integer k. Let $L \in \mathbf{NP}/n$, and M and $\{\alpha_n\}_{n\in\mathbb{N}}$ be its corresponding nondeterministic Turing machine and advice sequence. Let $p(n)$ be a polynomial running time upper bound of M on inputs of length n.

Now, we define a language L' such that a pair $(x, \alpha) \in L'$ if and only if $|x| = |\alpha|$ and M accepts x with advice bits set to α in $p(|x|)$ steps. Clearly, $L' \in \mathbf{NP}$ from the definition, so it has an n^k-size circuit family. Fixing the advice bits to the actual α_n's in the circuit family, we have an $n^{O(k)}$-size circuit family for L as well. This completes the proof. □

Lemma 5.4.3 (Theorem 14 [46])**.** *Let k be an integer. If* **NP** $\subset$ **P**$/poly$ *and all* **NP** *verifiers have n^k-size witnesses, then* **NP** $\subseteq$ $\mathbf{MATIME}[n^{O(k)}] \subset \mathbf{SIZE}[n^{O(k)}]$.

Proof. Assume all **NP** verifiers have n^k-size witnesses. By guessing circuits for the witnesses to PCP verifiers, it follows that **NP** $\subseteq$ $\mathbf{MATIME}[n^{O(k)}]$ [46]. Furthermore, we have $\mathbf{MATIME}[n^{O(k)}] \subset \mathbf{NTIME}[n^{O(k)}]/n^{O(k)} \subset \mathbf{SIZE}[n^{O(k)}]$. The last step follows from the assumption that **NP** $\subset$ **P**$/poly$ (and therefore $\mathsf{SAT} \in \mathbf{SIZE}[n^c]$ for a constant c). □

Now, we are ready to prove our equivalence theorem (restated below).

Reminder of Theorem 5.1.1. *Assuming* **NP** $\subset$ **P**$/poly$*, the following are equivalent:*

1. **NP** *is not in* $\mathbf{SIZE}[n^k]$ *for all k.*
2. $\mathbf{AM}/1$ *is in c-r.o.-*$\mathbf{NP}/n^\varepsilon$ *for all $\varepsilon > 0$ and integers c.*
3. **NP** *does not have n^k-size witnesses for all k.*[8]
4. *For all k and d, there is a poly-time nondeterministic PRG with n bits of advice against n^k-size circuits d-robustly often.*[9]

[8] See the statement of Theorem 5.1.1 in the introduction for the definition of n^k-size witnesses.

[9] See the preliminaries section of this chapter (Section 5.2) for a full definition of nondeterministic PRG and d-robustly often.

5. $\mathbf{NP}$ *is not in* $\mathbf{AMTIME}(n^k)$ *for all* k.

6. $(\mathbf{NP} \cap \mathbf{coNP})/n^\varepsilon$ *is not in* $\mathbf{SIZE}[n^k]$ *for all* k *and all* $\varepsilon > 0$.

7. $(\mathbf{AM} \cap \mathbf{coAM})/1$ *is in* c*-r.o.-*$(\mathbf{NP} \cap \mathbf{coNP})/n^\varepsilon$ *for all* $\varepsilon > 0$ *and all integers* c.

Proof. We prove the following directions to show equivalence.

(2) ⇒ (1). Suppose (2) holds. For all k, let L be the $\mathbf{MA}/1$ language and let c be the corresponding constant c guaranteed by Lemma 5.4.1. By (2) and the fact that $\mathbf{MA}/1 \subseteq \mathbf{AM}/1$, there is an $\mathbf{NP}/n$ language L' such that for infinitely many n's, L' agrees with L on inputs with length in $[n, n^{2c}]$.

Let $\tau = \lceil \log(n) \rceil$. By the condition of Lemma 5.4.1, we know that for at least one $\ell \in [n, n^{2c}]$, we have $\mathsf{SIZE}(L'_\ell) \geq \ell^k$. Since there are infinitely many such n, we conclude that L' is not in $\mathbf{SIZE}[n^k]$. Since k can be an arbitrary integer, it further implies that $\mathbf{NP}/n$ is not in $\mathbf{SIZE}[n^k]$ for all k, and hence also $\mathbf{NP}$ is not in $\mathbf{SIZE}[n^k]$ for all k by Lemma 5.4.2.

(1) ⇒ (3). We prove the contrapositive. Suppose $\mathbf{NP}$ has n^k-size witnesses for an integer k. Then, by Lemma 5.4.3, $\mathbf{NP} \subset \mathbf{SIZE}[n^{O(k)}]$.

(3) ⇒ (4). This follows directly from standard hardness-to-pseudorandomness constructions [87], more or less. Specifically, for all integers k and d and $\varepsilon > 0$, there is a language $L \in \mathbf{NP}$ without $n^{gkd/\varepsilon}$-size witnesses. Equivalently, there is a poly-time verifier V for L, such that there are infinitely many $x \in L$ such that for all y with $V(x, y) = 1$, it follows $\mathsf{CircuitComplexity}(y) \geq |x|^{gkd/\varepsilon}$.

For such an $x \in L$ with $|x| = m$, we can guess a y such that $V(x, y) = 1$ and apply Theorem 5.2.1 to construct a poly-time nondeterministic PRG with seed length $O(\log m)$, which works for input length $n \in [m^{1/\varepsilon}, m^{d/\varepsilon}]$ and against n^k-size circuits. Note that advice length is $|x| = m \leq n^\varepsilon$.

(4) ⇒ (2). First, under the assumption that $\mathbf{NP} \subset \mathbf{P}/poly$, we have the collapse $\mathbf{AM}/1 = \mathbf{MA}/1$ [14]. So it suffices to show that $\mathbf{MA}/1 \subset$ c-r.o.-$\mathbf{NP}/n^{\varepsilon}$ for all $\varepsilon > 0$ and integers d.

Let $L \in \mathbf{MA}/1$. That is, for some constant k, there is an n^k-time algorithm $A(x, y, z, \alpha)$ with one bit of advice α_n, such that

- $x \in L \Rightarrow$ there is a y of $|x|^k$ length such that $\Pr_z[A(x, y, z, \alpha_n) = 1] \geq 2/3$.
- $x \notin L \Rightarrow$ for all y of $|x|^k$ length, $\Pr_z[A(x, y, z, \alpha_n) = 1] \leq 1/3$.

Fixing the x, y, α_n, we can construct a circuit $C_{x,y,\alpha_n}(z) := A(x, y, z, \alpha_n)$ of size n^{2k} in n^{2k} time.

Now, by (4), for all d, there is a poly-time NPRG G with seed length $O(\log n)$ and advice length n^{ε} such that there are infinitely many m's such that for all $n \in [m, m^d]$, G_n fools n^{2k}-size circuits.

Applying G_n to fool C_{x,y,α_n} directly, we obtain a language $L' \in \mathbf{NP}/n^{\varepsilon}$ such that there are infinitely many m such that L' agrees with L on all input lengths in $[m, m^d]$. This completes the proof since d can be made arbitrarily large.

(5) ⇒ (3). We prove the contrapositive. Suppose $\mathbf{NP}$ has n^k-size witnesses for an integer k. By Lemma 5.4.3, it follows that

$$\mathbf{NP} \subseteq \mathbf{MATIME}[n^{O(k)}] \subseteq \mathbf{AMTIME}[n^{O(k)}].$$

(1) ⇒ (5). Again, we prove the contrapositive. We have

$$\mathbf{NP} \subseteq \mathbf{AMTIME}[n^{O(k)}] \subset \mathbf{NTIME}[n^{O(k)}]/n^{O(k)} \subset \mathbf{SIZE}[n^{O(k)}].$$

The last step follows from the assumption that $\mathbf{NP} \subseteq \mathbf{P}/poly$ (and therefore $\mathsf{SAT} \in \mathbf{SIZE}[n^c]$ for a constant c).

(6) ⇒ (1). $(\mathbf{NP} \cap \mathbf{coNP})/n^{\varepsilon}$ is not in $\mathbf{SIZE}[n^k]$ for all k and $\varepsilon > 0$ implies $\mathbf{NP}/n$ is not in $\mathbf{SIZE}[n^k]$ for all k, which in turn implies $\mathbf{NP}$ is not in $\mathbf{SIZE}[n^k]$ for all k by Lemma 5.4.2.

(4) $\Rightarrow$ (7). This follows similarly as the direction from (4) to (2).

(7) $\Rightarrow$ (6). This follows similarly as the direction from (2) to (1). Note that [14] also implies $(\mathbf{MA} \cap \mathbf{coMA})/1 = (\mathbf{AM} \cap \mathbf{coAM})/1$ under the assumption $\mathbf{NP} \subset \mathbf{P}/poly$. □

5.5 NP Circuit Lower Bounds Imply Better Karp-Lipton Collapses

Now we show a corollary of Theorem 5.1.1 that **NP** circuit lower bounds imply better Karp-Lipton collapses.

Reminder of Theorem 5.1.2. *Let $\mathcal{C} \in \{\oplus\mathbf{P}, \mathbf{PSPACE}, \mathbf{PP}, \mathbf{EXP}\}$. Suppose $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ for all k. Then for all $\varepsilon > 0$, $(\mathcal{C} \subset \mathbf{P}/poly \implies \mathcal{C} \subset i.o.\text{-}\mathbf{NP}/n^{\varepsilon})$. In particular, polynomial-size circuits for any $\mathcal{C}$-complete language can be constructed in* **NP** *on infinitely many input lengths with n^{ε} advice.*

Proof of Theorem 5.1.2. We first prove it for $\oplus\mathbf{P}$. Suppose for all k, $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ and $\oplus\mathbf{P} \subset \mathbf{P}/poly$.

First, note that $\mathbf{BPP}^{\oplus\mathbf{P}} \subset \mathbf{P}/poly$, implying $\mathbf{PH} \subset \mathbf{P}/poly$ by Toda's theorem [84]. Therefore, by Theorem 5.1.1 together with our assumption, we have $\mathbf{MA} \subset$ c-r.o.-$\mathbf{NP}/n^{\varepsilon}$ for all $\varepsilon > 0$ and integers c. In particular, $\mathbf{MA} \subset$ i.o.-$\mathbf{NP}/n^{\varepsilon}$ for all $\varepsilon > 0$. Now it suffices to show that $\oplus\mathbf{P} \subset \mathbf{P}/poly \implies \oplus\mathbf{P} \subseteq \mathbf{MA}$.

Let Π be the random self-reducible and downward self-reducible $\oplus\mathbf{P}$-complete language in [59]. By our assumption that $\oplus\mathbf{P} \subset \mathbf{P}/poly$, Π has a poly-size circuit family.

Then we can guess-and-verify these circuits in **MA**. We first existentially guess a circuit C_k for Π on every input length $k = 1, \ldots, n$. C_1 can be verified in constant time, and each successive circuit can be verified via random downward self-reducibility: given a circuit of length m that computes Π_m exactly, a circuit of length $m+1$ can be checked on random inputs to verify (with high probability) its consistency with

Π_{m+1} (which is computable using the downward self-reducibility and the circuit for Π_m). Then we can apply the random self-reducibility to construct an exact circuit for Π_{m+1} from C_{m+1} with high probability, as we already know C_{m+1} approximates Π_{m+1} very well. Therefore, with high probability, we can guess-and-verify a circuit for Π_n via a poly-time **MA** computation. This puts $\oplus\mathbf{P} \subseteq \mathbf{MA}$. Combining that with $\mathbf{MA} \subset$ i.o.-$\mathbf{NP}/n^{\varepsilon}$ for all $\varepsilon > 0$, we can conclude that $\oplus\mathbf{P} \subset$ i.o.-$\mathbf{NP}/n^{\varepsilon}$ for all $\varepsilon > 0$.

To construct a circuit for Π_n in i.o.-$\mathbf{NP}/n^{\varepsilon}$, note that by Theorem 5.1.4, for all k, we have an i.o.-NPRG fooling n^k-size circuits. We can pick k to be a sufficiently large integer, and use the i.o.-NPRG to derandomize the above process. This turns out to be more subtle than one might expect.

Construction of poly-size circuits of Π_n in i.o.-$\mathbf{NP}/n^{\varepsilon}$. Let d be a sufficiently large constant. Since we only aim for an i.o.-construction, we can assume that our i.o.-NPRG works for the parameter n, and fools all n^d-size circuits. Also, suppose we have $\mathsf{SIZE}(\Pi_n) \leq n^c$ for all n and a constant c.

We say a circuit C γ-approximates a function f, if $C(x) = f(x)$ for at least a γ fraction of the inputs.

Again, suppose we already constructed the circuits $C_1, \ldots, C_k$ for $\Pi_1, \ldots, \Pi_k$. This time we cannot guarantee C_i exactly computes Π_i. Instead, we relax the condition a bit and ensure that C_i $(1 - 4/n)$-approximates Π_i for all $i \in [k]$. Clearly, we can check $C_1 \equiv \Pi_1$ directly so this can be satisfied when $k = 1$.

We now show how to construct an approximate circuit for Π_{k+1}. First, using the random self-reducibility of Π and the circuit C_k approximating Π_k, there is an oracle circuit E of size $\mathrm{poly}(n)$, which takes two inputs x with $|x| = k$ and r with $|r| = \mathrm{poly}(n)$, such that for all x,

$$\Pr_r \left[E^{C_k}(x, r) = \Pi_k(x) \right] \geq 1 - 1/2^n.$$

Also, by the downward self-reducibility of Π, there is an oracle machine D of $\mathrm{poly}(k)$ size, such that $D^{\Pi_k}(z) = \Pi_{k+1}(z)$ for all z.

Now, consider the following circuit $G(x,r)$ for computing Π_{k+1}: the circuit simulates D^{Π_k}, while answering all queries w to Π_k using $E^{C_k}(w,r)$. For each input $x \in \{0,1\}^{k+1}$, let $w_1, w_2, \ldots, w_{\mathrm{poly}(n)}$ be all queries to Π_k made by running D on the input x assuming all answers are correct, we can see that if $E^{C_k}(w_j,r) = \Pi_k(w_j)$ for all these w_j's, then $G(x,r) = \Pi_{k+1}(x)$. Therefore, we have

$$\Pr_r[G(x,r) = \Pi_{k+1}(x)] \geq 1 - \mathrm{poly}(n)/2^n,$$

for all $x \in \{0,1\}^{k+1}$.

Now, we guess a circuit C_{k+1} of size $(k+1)^c$ which is supposed to compute Π_{k+1}. By an enumeration of all possible seeds to our NPRG, we can estimate the probability

$$p_{\mathsf{good}} := \Pr_{x \in \{0,1\}^{k+1}} \Pr_r[G(x,r) = C_{k+1}(x)].$$

within $1/n$ in $\mathrm{poly}(n)$ time, as the expression $[G(x,r) = C_{k+1}(x)]$ has a $\mathrm{poly}(n)$ size circuit with inputs being x and r. Let our estimation be p_{est}. We have $|p_{\mathsf{good}} - p_{\mathsf{est}}| \leq 1/n$.

Putting the above together, we have

$$\left| \Pr_{x \in \{0,1\}^{k+1}}[\Pi_{k+1}(x) = C_{k+1}(x)] - p_{\mathsf{good}} \right| \leq \mathrm{poly}(n)/2^n.$$

We reject immediately if our estimation $p_{\mathsf{est}} < 1 - 2/n$ (note that if C_{k+1} is the correct circuit, p_{good} would be larger than $1 - \mathrm{poly}(n)/2^n > 1 - 1/n$, and therefore $p_{\mathsf{est}} > 1 - 2/n$). So after that, we can safely assume that C_{k+1} $(1-4/n)$-approximates Π_{k+1}.

Therefore, at the end we have an n^c-size circuit C_n which $(1-4/n)$-approximates Π_n, and we try to recover an exact circuit for Π_n from C_n by exploiting the random self-reducibility of Π_n again. Note that there is an oracle circuit $E(x,r)$, which takes two inputs x with $|x| = n$ and r with $|r| = \mathrm{poly}(n)$ such that for all x,

$$\Pr_r[E^{C_n}(x,r) = \Pi_n(x)] \geq 2/3.$$

Now, we generate $\ell = n^{O(1)}$ strings $r_1, r_2, \ldots, r_\ell$ by enumerating all seeds to the NPRG. We construct our final circuit C to be the majority of $E^{C_n}(x, r_j)$ for all $j \in [\ell]$. It is not hard to see that C computes Π_n exactly, as our inputs $\{r_j\}_{j\in[\ell]}$ fool the expression $\left[E^{C_n}(x, r) = \Pi_n(x)\right]$ for all $x \in \{0,1\}^n$.

For the case of **PP** and **PSPACE**, one can implement the above procedure in the same way, using the corresponding random self-reducible and downward self-reducible #**P**-complete and **PSPACE**-complete problems (Permanent and the **PSPACE**-complete language in [86]).

For the case of **EXP**, note that $\mathbf{EXP} \subset \mathbf{P}/poly \implies \mathbf{EXP} = \mathbf{PSPACE}$, so we can proceed the same way as for **PSPACE** (since **EXP** = **PSPACE**, **PSPACE**-complete languages are also **EXP**-complete). □

5.6 Consequence of Weak Circuit Lower Bounds for Sparse Languages in NP

Now, we are ready to prove the consequences of weak circuit lower bounds for sparse **NP** languages. We first need the following lemma.

Lemma 5.6.1 (Hardness Verification from Circuit Lower Bounds for Sparse Languages in $\mathbf{NTIME}[T(n)]$)**.** *Let $S_{\mathsf{ckt}}(n), S_{\mathsf{sparse}}(n), T(n) : \mathbb{N} \to \mathbb{N}$ be time constructible functions. Suppose there is an $S_{\mathsf{sparse}}(n)$-sparse language $L \in \mathbf{NTIME}[T(n)]$ without $(n \cdot S_{\mathsf{ckt}}(n))$-size circuits. Then there is a procedure V such that:*

- *V takes a string z of length $n \cdot S_{\mathsf{sparse}}(n)$ as input and an integer $\ell \leq S_{\mathsf{sparse}}(n)$ as advice.*
- *V runs in $O(S_{\mathsf{sparse}}(n) \cdot T(n))$ nondeterministic time.*
- *For infinitely many n, there is an integer $\ell_n \leq S_{\mathsf{sparse}}(n)$ such that $V(z, \ell_n)$ accepts exactly one string z, and z has circuit complexity $\Omega(S_{\mathsf{ckt}}(n)/\log S_{\mathsf{sparse}}(n))$.*

Proof. Let L be the $\mathbf{NTIME}[T(n)]$ language in the assumption. Let $N = n \cdot S_{\mathsf{sparse}}(n)$. We define a string $\mathsf{List}_{L_n} \in \{0,1\}^N$ as the concatenation of all $x \in L_n$ in lexicograph-

ical order, together with additional zeros at the end to make the string have length exactly N.

Now define a function f_n on $m = \log\lceil N+1 \rceil$ bits, with truth-table $\mathsf{List}_{L_n} 10^{2^m - N}$.

We claim that $\mathsf{SIZE}(L_n) \leq O(\mathsf{SIZE}(f_n) \cdot n \cdot \log(S_{\mathsf{sparse}}(n)))$. To determine whether $x \in L_n$, it would suffice to perform a binary search on the list List_{L_n}. We construct a circuit for L_n which performs binary search using f_n. First, we hard-wire the length of the list $\ell := |L_n| \leq S_{\mathsf{sparse}}(n)$ into our circuit for L_n so that the binary search can begin with the correct range. A binary search on List_{L_n} takes $O(\log S_{\mathsf{sparse}}(n))$ comparisons, and each comparison requires $O(n)$ calls to f_n (to print the appropriate string). It is easy to see that the circuit size required for the binary search is dominated by the total cost of the comparisons; this proves the claim.

From the assumption, we know that for infinitely many n, L_n has no circuit of size $n \cdot S_{\mathsf{ckt}}(n)$. By our upper bound on the circuit size of L_n, it follows that on the same set of n, the function f_n has circuit complexity at least $\Omega(S_{\mathsf{ckt}}(n)/\log S_{\mathsf{sparse}}(n))$.

Now, we construct an algorithm V that only accepts the string $f_n = \mathsf{List}_{L_n} 10^{2^m - N}$. We first need the integer $\ell = |L_n|$ as the advice. Given a string Y of length N, we check that Y contains exactly ℓ distinct inputs in $\{0,1\}^n$ in lexicographical order with the correct format, and we guess an $O(T(n))$-length witness for each input to verify they are indeed all in L. It is easy to see that V runs in $O(S_{\mathsf{sparse}}(n) \cdot T(n))$ nondeterministic time, which completes the proof. □

Remark 5.6.2. *Note that the advice integer ℓ can be calculated directly with an* **NP** *oracle by doing a binary search for ℓ, which takes $O(\log S_{\mathsf{sparse}}(n))$* **NP***-oracle calls. That is, one can also use a $\mathbf{P}^{\mathbf{NP}[O(\log S_{\mathsf{sparse}}(n))]}$ verifier without advice bits in the statement of Lemma 5.6.1.*

Remark 5.6.3. *As mentioned in the introduction, the above proof can be seen as a type of* hardness condensation *for all sparse* **NTIME**$[T(n)]$ *languages. The goal of hardness condensation [28, 58] is that, given a hard function f on n input bits with complexity S, we want to construct a function $\widetilde{f}$ on $\ell \ll n$ input bits that still has complexity roughly S. The above proof shows any hard sparse language in* **NTIME**$[T(n)]$

can be "condensed" into a function representing its sorted yes-instances.

Combining Lemma 5.6.1 with Theorem 5.2.1, we obtain a construction of an i.o.-NPRG.

Corollary 5.6.4 (NPRG from lower bounds against sparse $\mathbf{NTIME}[T(n)]$ languages)**.** *Under the circuit lower bound assumption of Lemma 5.6.1, there is an i.o.-NPRG G with the properties:*

- *G has $O(\log S_{\mathsf{sparse}}(n) + \log n)$ seed length.*
- *G takes $O(\log S_{\mathsf{sparse}}(n))$ bits of advice.*
- *G runs in $S_{\mathsf{sparse}}(n) \cdot T(n) + poly(n \cdot S_{\mathsf{sparse}}(n))$ time.*
- *G fools circuits of size at most $(S_{\mathsf{ckt}}(n)/\log S_{\mathsf{sparse}}(n))^{\Omega(1)}$.*

Now we are ready to prove Theorem 5.1.4.

Reminder of Theorem 5.1.4. *Suppose there is an $\varepsilon > 0$, a $c \geq 1$, and an n^c-sparse $L \in \mathbf{NP}$ without $n^{1+\varepsilon}$-size circuits. Then $\mathbf{MA} \subset$ i.o.-$\mathbf{NP}/O(\log n)$, $\mathbf{MA} \subseteq$ i.o.-$\mathbf{P}^{\mathbf{NP}[O(\log n)]}$, and $\mathbf{NE} \not\subset \mathbf{SIZE}[2^{\delta \cdot n}]$ for some $\delta > 0$ (which implies $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ for all k).*

Proof. First, by Corollary 5.6.4 and setting $S_{\mathsf{ckt}}(n) = n^{\varepsilon}$, $S_{\mathsf{sparse}}(n) = n^c$ and $T(n) = \mathrm{poly}(n)$, there is an i.o.-NPRG with seed length $O(\log n)$ which takes $O(\log n)$ bits of advice, runs in $\mathrm{poly}(n)$ time, and fools circuits of size $n^{\Omega(\varepsilon)} = n^{\Omega(1)}$. Note that we can simply scale it up to make it fool circuits of size n^k for any k, with only a constant factor blowup on seed length and advice bits and a polynomial blowup on the running time.

Applying the i.o.-NPRG to arbitrary Merlin-Arthur computations, we conclude $\mathbf{MA} \subset$ i.o.-$\mathbf{NP}/O(\log n)$. Similarly, we conclude $\mathbf{MA} \subseteq$ i.o.-$\mathbf{P}^{\mathbf{NP}[O(\log n)]}$ from Remark 5.6.2.

Now we show $\mathbf{NE} \not\subset \mathbf{SIZE}[2^{\delta \cdot n}]$ for some $\delta > 0$. By Lemma 5.6.1, there is a nondeterministic algorithm running in $\mathrm{poly}(n)$ time, given $\alpha_n = c \log n$ bits of advice,

guess and verify a string of length n^{c+1} which has circuit complexity at least $n^{\varepsilon/2}$, for infinitely many n. We say these infinitely many n are *good* n.

Next, we define the following language $L \in \mathbf{NE}$: Given an input of length m, our nondeterministic algorithm treats the first $\ell = m/4c$ bits a binary encoded integer $n \leq 2^\ell$. Then it treats the next $c \log n$ input bits a as the advice, and tries to guess-and-verify a string z which passes the verification procedure in Lemma 5.6.1 with advice a and parameter n. Finally, it treats the next $(c+1) \cdot \log n$ input bits as an integer $i \in [n^{c+1}]$, and accepts if and only $z_i = 1$.

First, it is easy to verify $L \in \mathbf{NE}$, as the algorithm runs in $\mathrm{poly}(n) \leq 2^{O(\ell)} \leq 2^{O(m)}$ nondeterministic time. For the circuit complexity of L, we know that for the "good" values on n, on inputs of length of $m = 4 \cdot c \cdot \lceil \log n \rceil$, if we fix the first $m/4c$ bits to represent the integer n, and the next $c \log n$ bits to the actual advice α_n, L would compute the hard string of length n^{c+1} on the next $(c+1) \cdot \log n$ bits. Therefore, $\mathsf{SIZE}(L_m) \geq n^\varepsilon \geq 2^{\Omega(m)}$ for infinitely many m, which completes the proof. □

Finally, we prove Theorem 1.0.11.

Reminder of Theorem 1.0.11. **NEXP** $\not\subset$ **P**/*poly* *if and only if there is an* $\varepsilon > 0$ *such that for all sufficiently small* $\beta > 0$, *there is a* 2^{n^β}*-sparse language* $L \in$ **NTIME**$[2^{n^\beta}]$ *without* $n^{1+\varepsilon}$*-size circuits.*

Proof. ($\Rightarrow$) This direction is easy to prove using standard methods. Suppose that **NEXP** $\not\subset$ **P**/*poly*; this implies **NE** $\not\subset$ **P**/*poly*. Therefore, there is a language $L \in$ **NTIME**$[2^n]$ that does not have $n^{2/\beta}$-size circuits. Define a padded language $L' = \{x10^{|x|^{1/\beta}-1} | x \in L\}$. It is easy to see that $L' \in$ **NTIME**$[2^{m^\beta}]$, by running the **NE** algorithm for L on its first $n = O(m^\beta)$ input bits. From the circuit lower bound on L, it follows that L' does not have $n^{2/\beta} = m^2$-size circuits.

($\Leftarrow$) First, it follows from Impagliazzo-Kabanets-Wigderson [60] that, if for every ε and integer k, there is an i.o.-NPRG with seed length n^ε, n^ε advice bits, and 2^{n^ε} running time that fools n^k-size circuits, then **NEXP** $\not\subset$ **P**/*poly*.

Setting $S_{\mathsf{ckt}}(n) = n^\varepsilon$, $S_{\mathsf{sparse}}(n) = 2^{n^\beta}$ and $T(n) = 2^{n^\beta}$ in Corollary 5.6.4, there is an i.o.-NPRG with seed length $O(n^\beta)$, takes $O(n^\beta)$ bits of advice, and runs in $2^{O(n^\beta)}$

time that fools circuits of size $n^{\Omega(\varepsilon/\beta)} = n^{\varepsilon'}$ for $\varepsilon' > 0$. By setting $m = n^{\varepsilon'/k}$, we obtain an i.o.-NPRG with seed/advice length $O(m^{\beta \cdot k/\varepsilon'})$ and running time $2^{O(m^{\beta \cdot k/\varepsilon'})}$, which fools circuits of size m^k. Therefore, by [60], it follows that $\mathbf{NEXP} \not\subset \mathbf{P}/poly$.

□

5.7 Almost Almost-everywhere $(\mathbf{MA} \cap \mathbf{coMA})/1$ Circuit Lower Bounds

Here we provide a proof for Lemma 5.4.1 for completeness. The proof is based on a similar lemma from [35].

Preliminaries

A crucial ingredient of the proof is a **PSPACE**-complete language [86] satisfying strong reducibility properties, which is also used in the fixed-polynomial lower bounds for $\mathbf{MA}/1$ and promise$\mathbf{MA}$ [81], and the recent new witness lemmas for **NQP** and **NP** [73].

We first define these reducibility properties.

Definition 5.7.1. *Let $L : \{0,1\}^* \to \{0,1\}$ be a language, we define the following properties:*

- *L is* downward self-reducible *if there is a constant c such that for all sufficiently large n, there is an n^c size uniform oracle circuit A such that for all $x \in \{0,1\}^n$, $A^{L_{n-1}}(x) = L_n(x)$.*

- *L is* paddable*, if there is a polynomial time computable projection Pad (that is, each output bit is either a constant or only depends on 1 input bit), such that for all integers $1 \leq n < m$ and $x \in \{0,1\}^n$, we have $x \in L$ if and only if $\mathsf{Pad}(x, 1^m) \in L$, where $\mathsf{Pad}(x, 1^m)$ always has length m.*

- *L is* same-length checkable *if there is a probabilistic polynomial-time oracle Turing machine M with output in $\{0,1,?\}$, such that, for any input x,*

– *M asks its oracle queries only of length $|x|$.*

– *If M is given L as an oracle, then M outputs $L(x)$ with probability 1.*

– *M outputs $1 - L(x)$ with probability at most $1/3$ no matter which oracle is given to it.*

We call M an instance checker *for L.*

Remark 5.7.2. *Note that the paddable property implies $\mathsf{SIZE}(L_n)$ is non-decreasing.*

The following **PSPACE**-complete language is given by [81] (modifying a construction of Trevisan and Vadhan [86]).

Theorem 5.7.3 ([81, 86])**.** *There is a* **PSPACE***-complete language $L^{\mathbf{PSPACE}}$ which is paddable, downward self-reducible, and same-length checkable.*

We also need the following folklore theorem which is proved by a direct diagonalization against all small circuits.

Theorem 5.7.4. *Let $n \leq s(n) \leq 2^{o(n)}$ be space-constructible. There is a universal constant c and a language $L \in \mathbf{SPACE}[s(n)^c]$ that $\mathsf{SIZE}(L_n) > s(n)$ for all sufficiently large n.*

Definitions

We need the following convenient definition of an $\mathbf{MA} \cap \mathbf{coMA}$ algorithm, which simplifies the presentation.

Definition 5.7.5. *A language L is in $\mathbf{MA} \cap \mathbf{coMA}$, if there is a deterministic algorithm $A(x, y, z)$ (which is called the predicate) such that:*

- *A takes three inputs x, y, z such that $|x| = n$, $|y| = |z| = poly(n)$ (y is the witness while z is the collection of random bits), runs in $O(T(n))$ time, and outputs an element from $\{0, 1, ?\}$.*
- *(Completeness) There exists a y such that*

$$\Pr_z[A(x, y, z) = L(x)] \geq 2/3.$$

- *(Soundness) For all y,*

$$\Pr_z[A(x, y, z) = 1 - L(x)] \leq 1/3.$$

Remark 5.7.6. ($\mathbf{MA} \cap \mathbf{coMA}$) *languages with advice are defined similarly, with A being an algorithm with the corresponding advice.*

Note that by above definition, the semantic of $(\mathbf{MA} \cap \mathbf{coMA})/1$ is different from $\mathbf{MA}/1 \cap \mathbf{coMA}/1$. A language in $(\mathbf{MA} \cap \mathbf{coMA})/1$ has both an $\mathbf{MA}/1$ algorithm and a $\mathbf{coMA}/1$ algorithm, and *their advice bits are the same*. While a language in $\mathbf{MA}/1 \cap \mathbf{coMA}/1$ can have an $\mathbf{MA}/1$ algorithm and a $\mathbf{coMA}/1$ algorithm with different advice sequences.

Proof for Lemma 5.4.1

Now we are ready to prove Lemma 5.4.1 (restated below).

Reminder of Lemma 5.4.1. *For all constants k, there is an integer c, and a language $L \in (\mathbf{MA} \cap \mathbf{coMA})/1$, such that for all sufficiently large $\tau \in \mathbb{N}$ and $n = 2^\tau$, either*

- $\mathsf{SIZE}(L_n) > n^k$, *or*
- $\mathsf{SIZE}(L_m) > m^k$, *for an $m \in (n^c, 2 \cdot n^c) \cap \mathbb{N}$.*

Proof. Let $L^{\mathbf{PSPACE}}$ be the language specified by Theorem 5.7.3. By Theorem 5.7.4, there is an integer c_1 and a language L^{diag} in $\mathbf{SPACE}(n^{c_1})$, such that $\mathsf{SIZE}(L_n^{\mathsf{diag}}) \geq n^k$ for all sufficiently large n. By the fact that $L^{\mathbf{PSPACE}}$ is $\mathbf{PSPACE}$-complete, there is a constant c_2 such that L_n^{diag} can be reduced to $L^{\mathbf{PSPACE}}$ on input length n^{c_2} in n^{c_2} time. We set $c = c_2$.

The Algorithm. Let $\tau \in \mathbb{N}$ be sufficiently large. We also let b to be a constant to be specified later. Given an input x of length $n = 2^\tau$ and let $m = n^c$, we first provide an informal description of the $(\mathbf{MA} \cap \mathbf{coMA})/1$ algorithm which computes the language L. There are two cases:

1. When $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) \leq n^b$. That is, when $L_m^{\mathbf{PSPACE}}$ is *easy*. In this case, on inputs of length n, we guess-and-verify a circuit for $L_m^{\mathbf{PSPACE}}$ of size n^b and use that to compute L_n^{diag}.

2. Otherwise, we know $L_m^{\mathbf{PSPACE}}$ is *hard*. Let ℓ be the largest integer such that $\mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \leq n^b$. On inputs of length $m_1 = m + \ell$, we guess-and-verify a circuit for $L_\ell^{\mathbf{PSPACE}}$ and compute it (that is, compute $L_\ell^{\mathbf{PSPACE}}$ on the first ℓ input bits while ignoring the rest).

Intuitively, the above algorithm computes a hard function because either it computes the hard language L_n^{diag} on inputs of length n, or it computes the hard language $L_\ell^{\mathbf{PSPACE}}$ on inputs of length m_1. A formal description of the algorithm is given in Algorithm 2, while an algorithm for setting the advice sequence is given in Algorithm 3. It is not hard to see that a y_n can only be set once in Algorithm 3.

The Algorithm Satisfies the MA∩coMA Promise. We first show the algorithm satisfies the $\mathbf{MA} \cap \mathbf{coMA}$ promise (Definition 5.7.5). The intuition is that it only tries to guess-and-verify a circuit for $L^{\mathbf{PSPACE}}$ when it exists, and the properties of the instance checker (Definition 5.7.1) ensure that in this case the algorithm satisfies the $\mathbf{MA} \cap \mathbf{coMA}$ promise. Let $y = y_n$, there are three cases:

1. $y = 0$. In this case, the algorithm computes the all zero function, and clearly satisfies the $\mathbf{MA} \cap \mathbf{coMA}$ promise.

2. $y = 1$ and n is a power of 2. In this case, from Algorithm 3, we know that $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) \leq n^b$ for $m = n^c$. Therefore, at least one guess of the circuit is the correct circuit for $L_m^{\mathbf{PSPACE}}$, and on that guess, the algorithm outputs $L_n^{\mathsf{diag}}(x) = L_m^{\mathbf{PSPACE}}(z)$ with probability at least $2/3$, by the property of the instance checker (Definition 5.7.1).

   Again by the property of the instance checker, on all possible guesses, the algorithm outputs $1 - L_n^{\mathsf{diag}}(x) = 1 - L_m^{\mathbf{PSPACE}}(z)$ with probability at most $1/3$. Hence, the algorithm correctly computes L_n^{diag} on inputs of length n, with respect to Definition 5.7.5.

Algorithm 2: The $\mathbf{MA} \cap \mathbf{coMA}$ algorithm

Given an input x with input length $n = |x|$;
Given an advice bit $y = y_n \in \{0, 1\}$;
Let $m = n^c$;
Let $n_0 = n_0(n)$ be the largest integer such that $n_0^c \leq n$;
Let $m_0 = n_0^c$;
Let $\ell = n - m_0$;
if $y = 0$ **then**
    Output 0 and terminate
if n *is a power of* 2 **then**
    (We are in the case that $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) \leq n^b$.);
    Compute z in n^c time such that $L_n^{\mathsf{diag}}(x) = L_m^{\mathbf{PSPACE}}(z)$;
    Guess a circuit C of size at most n^b;
    Let M be the instance checker for $L_m^{\mathbf{PSPACE}}$;
    Flip an appropriate number of random coins, let them be r;
    Output $M^C(z, r)$;
else
    (We are in the case that $\mathsf{SIZE}(L_{m_0}^{\mathbf{PSPACE}}) > n_0^b$ and ℓ is the largest integer such that $\mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \leq n_0^b$.);
    Let z be the first ℓ bits of x;
    Guess a circuit C of size at most n_0^b;
    Let M be the instance checker for $L_\ell^{\mathbf{PSPACE}}$;
    Flip an appropriate number of random coins, let them be r;
    Output $M^C(z, r)$;

Algorithm 3: The algorithm for setting advice bits

All y_n's are set to 0 by default;
for $\tau = 1 \to \infty$ **do**
    Let $n = 2^\tau$;
    Let $m = n^c$;
    if $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) \leq n^b$ **then**
        Set $y_n = 1$;
    else
        Let $\ell = \max\{\ell : \mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \leq n^b\}$;
        Set $y_{m+\ell} = 1$;

3. $y = 1$ and n is not a power of 2. In this case, from Algorithm 3, we know that $\mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \leq n_0^b$. Therefore, at least one guess of the circuit is the correct circuit for $L_\ell^{\mathbf{PSPACE}}$, and on that guess, the algorithm outputs $L_\ell^{\mathbf{PSPACE}}(z)$ ($z = z(x)$ is the first ℓ bits of x) with probability at least $2/3$, by the property of the instance checker (Definition 5.7.1).

   Again by the property of the instance checker, on all possible guesses, the algorithm outputs $1 - L_\ell^{\mathbf{PSPACE}}(z)$ with probability at most $1/3$. Hence, the algorithm correctly computes $L_\ell^{\mathbf{PSPACE}}(z(x))$ on inputs of length n, with respect to Definition 5.7.5.

The Algorithm Computes a Hard Language. Next we show that the algorithm indeed computes a hard language as stated. Let τ be a sufficiently large integer, $n = 2^\tau$, and $m = n^c$. According to Algorithm 3, there are two cases:

- $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) \leq n^b$. In this case, Algorithm 3 sets $y_n = 1$. And by previous analyses, we know that L_n computes the hard language L_n^{diag}, and therefore $\mathsf{SIZE}(L_n) > n^k$.

- $\mathsf{SIZE}(L_m^{\mathbf{PSPACE}}) > n^b$. Let ℓ be the largest integer such that $\mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \leq n^b$. By Remark 5.7.2, we have $0 < \ell < m$.

  Note that $\mathsf{SIZE}(L_{\ell+1}^{\mathbf{PSPACE}}) \leq (\ell+1)^d \cdot \mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}})$ for a universal constant d, because $L^{\mathbf{PSPACE}}$ is downward self-reducible. Therefore,

$$\mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \geq \mathsf{SIZE}(L_{\ell+1}^{\mathbf{PSPACE}})/(\ell+1)^d \geq n^b/m^d \geq n^{b-c\cdot d}.$$

  Now, on inputs of length $m_1 = m + \ell$, we have $y_{m_1} = 1$ by Algorithm 3 (note that $m_1 \in (m, 2m)$ as $\ell \in (0, m)$). Therefore, L_{m_1} computes $L_\ell^{\mathbf{PSPACE}}$, and

$$\mathsf{SIZE}(L_{m_1}) = \mathsf{SIZE}(L_\ell^{\mathbf{PSPACE}}) \geq n^{b-c\cdot d}.$$

  We set b such that $n^{b-c\dot{d}} \geq (2m)^k \geq m_1^k$ (we can set $b = cd + 3 \cdot ck$), which completes the proof.

□

5.8 Open Problems

We conclude this chapter with three interesting open questions stemming from this work.

1. *Are fixed-polynomial circuit lower bounds for* **NP** *equivalent to a Karp-Lipton collapse of* **PH** *to* **NP***?*

   Formally, is $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ for all k equivalent to ($\mathbf{NP} \subset \mathbf{P}/poly \implies \mathbf{PH} \subset$ i.o.-$\mathbf{NP}/n$)? Recall we showed that similar Karp-Lipton-style collapses do occur, assuming **NP** circuit lower bounds (e.g., ($\mathbf{PSPACE} \subset \mathbf{P}/poly \implies \mathbf{PSPACE} \subset$ i.o.-$\mathbf{NP}/n$)), and we showed that $\mathbf{NP} \not\subset \mathbf{SIZE}[n^k]$ implies a type of collapse of **AM** into **NP**.

2. It is also a prominent open problem to prove that $\mathbf{ZPP}^{\mathbf{NP}}_{tt} \not\subset \mathbf{SIZE}[n^k]$ for some constant k [41] (that is, prove lower bounds for **ZPP** with nonadaptive queries to an **NP** oracle). *Is this lower bound equivalent to a Karp-Lipton collapse of* **PH***?*

   The difficulty is that, assuming $\mathbf{ZPP}^{\mathbf{NP}}_{tt} \not\subset \mathbf{SIZE}[n^k]$, it appears that we may obtain a good simulation of $\mathbf{BPP}^{\mathbf{NP}}_{tt}$, but we presently have no Karp-Lipton Theorem collapsing **PH** to $\mathbf{BPP}^{\mathbf{NP}}_{tt}$ (indeed, lower bounds for this class are also open). Furthermore, [41] observe that $\mathbf{NP} \subset \mathbf{P}/poly$ does imply the (small) collapse $\mathbf{BPP}^{\mathbf{NP}}_{tt} = \mathbf{ZPP}^{\mathbf{NP}}_{tt}$; it is unclear how a circuit lower bound against $\mathbf{ZPP}^{\mathbf{NP}}_{tt}$ would aid a further collapse.

3. In light of our Theorem 1.0.11, *is it possible to show interesting hardness magnification results for non-sparse versions of* MCSP *(say,* $\mathsf{MCSP}[2^m/m^2]$*)?*

   Currently, we only know hardness magnification results when the circuit size parameter is $2^{o(m)}$ [76, 75, 70].

Chapter 6

Concluding Thoughts – Lower Bounds and Hardness for MCSP

In this thesis, we have presented two kinds of results which seem quite harmonious.

- Nearly-quadratic lower bounds on the product of time and space needed for solving various problems, including a few circuit analysis problems.
- Strong consequences of much weaker lower bounds for another circuit analysis problem ($\mathsf{MCSP}[n^{10}]$).

If only MCSP had found its way into Chapter 3, we may have ended up with a much catchier thesis title (like "$\mathbf{P} \neq \mathbf{NP}$"). Unfortunately, our lower bounds remain intermediate for the time being.

Proving lower bounds against MCSP. There is an inherent dissonance that exists between Chapters 3 and 4. In Chapter 3, the problems for which we would really like to prove intermediate lower bounds are *compression problems*. Consider the following restatement of Corollary 1.0.5.

Proposition 6.0.1 (Strengthening of Corollary 1.0.5)**.** *If* search-$\mathsf{MCSP}[n^{10}]$ *does not have streaming algorithms with update time and space* $\log^{O(1)}(N)$*, then* $\mathbf{P} \neq \mathbf{NP}$*.*

When we proved NOE requires a time-space product of $\Omega(n^2)$, we very crucially relied on the fact that a *random* instance of NOE requires a *long* output. While

this is true for the general search-MCSP problem, it is not true for the versions of search-MCSP like search-MCSP$[n^{10}]$. In fact, random inputs fall into the "no such circuit exists" category with overwhelmingly high probability. Moreover, we crucially use the fact that the object we are searching for in search-MCSP$[n^{10}]$ is small, in our proof of Proposition 6.0.1.

However, this path may still hold some use for us. It is still open an open problem to prove the general search-MCSP (the problem of finding a minimum size circuit of a given truth table) requires a time-space product of at least $n^2/\log^{O(1)}(n)$. As mentioned above, random inputs are extremely likely to produce long outputs for search-MCSP. There may be difficulty in reproducing the remainder of the analysis in showing that short branching programs cannot output a large portion of the output, but there does not seem to be any barrier standing in our way. Nevertheless, we still do not know hardness magnification results (along the lines of Chapter 4) for *general* search-MCSP.

Giving reductions to MCSP. A significant ongoing venture is to relate MCSP to other well-studied problems, especially via reductions. Before recently, the common wisdom among complexity theorists was that finding reductions other natural problems to MCSP, especially for the goal of proving MCSP is **NP**-hard, is not a task worth attempting (at least as a long term venture). After all, Levin famously delayed publishing his original work on **NP**-completeness while he attempted to prove MCSP was **NP**-hard. There are even results showing that certain types of reductions from SAT to MCSP would give new circuit lower bounds [61, 74]. It is even open to give reduction from MKTP to MCSP, despite their tremendous similarities!

However, recent work by Ilango [55] and Ilango, Loff, and Oliviera [56] might inspire some researchers to pursue reductions to MCSP. We now examine what might come of trying to give new reductions to MCSP, even from simple problems like NOE or Sort.

Unfortunately, much like trying to prove lower bounds against search-MCSP$[n^{10}]$ directly, we run into the same annoying concern we ran into before. The reductions we gave in Chapter 3 required that our target problems also required long outputs! We

took an instance of NOE and produced an instance of Print-3SAT or #SAT which had an equally long output. In order to overcome this, we would need to use a different size parameter, or we could try coming up with a style of reduction that skirts this issue.

There is another option, though. When we fail to find reductions or algorithms, this often gives us the opportunity to consider impossibility results! Could we prove that there are no super-efficient reductions from NOE or Sort to search-MCSP? It might be quite easy, considering that NOE and Sort are both not affected by permutations on their inputs, but truth tables are not at all robust to this behavior (certain if reductions are required to be local, this intuition might be a path to the answer). What might it mean if there are no super-efficient reductions from NOE or Sort to search-MCSP?

There may additionally be an avenue here to find new results of the flavor of Murray and Williams [74], in which they conclude that efficient (logtime uniform $\mathbf{AC}^0$) **NP**-hardness reductions to MCSP would lead to strong circuit lower bounds. Perhaps we simply note the fact that sufficiently strong reductions from any of our problems for which we have nearly quadratic time-space lower bounds to versions of MCSP and other compression problems would give breakthroughs. With enough finagling, there may be a gem yet to be discovered.

Bibliography

[1] Scott Aaronson. Oracles are subtle but not malicious. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 340–354, 2006.

[2] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *TOCT*, 1(1):2:1–2:54, 2009.

[3] Karl Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987.

[4] Karl Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *Journal of Computer and System Sciences*, 43(2):269 – 289, 1991.

[5] Miklós Ajtai. Determinism versus nondeterminism for linear time rams with memory restrictions. *Journal of Computer and System Sciences*, 65(1):2–37, 2002.

[6] Miklós Ajtai. A non-linear time lower bound for boolean branching programs. *Theory of Computing*, 1(8):149–176, 2005.

[7] Eric Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *31th IARCS Conf. Found. of Software Tech. and Theoret. Comput. Sci. (FSTTCS'01)*, volume 2245, pages 1–15, 2001.

[8] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006. Preliminary version in FOCS'02.

[9] Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. In *MFCS*, pages 54:1–54:14, 2017.

[10] Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. *Computational Complexity*, 26(2):469–496, 2017. Preliminary version in STACS'15.

[11] Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *JACM*, 57(3), 2010.

[12] Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3):14:1–14:36, 2010.

[13] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[14] Vikraman Arvind, Johannes Köbler, Uwe Schöning, and Rainer Schuler. If NP has polynomial-size circuits, then ma=am. *Theor. Comput. Sci.*, 137(2):279–282, 1995.

[15] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991.

[16] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[17] David A Mix Barrington and Pierre McKenzie. Oracle branching programs and Logspace versus P. *Information and Computation*, 95(1):96–115, 1991.

[18] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, 1968.

[19] Paul Beame. A general sequential time-space tradeoff for finding unique elements. In *STOC*, pages 197–203. ACM, 1989.

[20] Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.

[21] Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *FOCS*, pages 290–299. IEEE, 2013.

[22] Paul Beame, Thathachar S Jayram, and Michael Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, 2001.

[23] Paul Beame, Michael Saks, Xiaodong Sun, and Erik Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM (JACM)*, 50(2):154–195, 2003.

[24] Paul Beame and Erik Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. In *STOC*, pages 688–697. ACM, 2002.

[25] Allan Borodin and Stephen Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, 1982.

[26] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.

[27] Harry Buhrman and Steven Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, pages 116–127, 1992.

[28] Joshua Buresh-Oppenheim and Rahul Santhanam. Making hard problems harder. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 73–87, 2006.

[29] Samuel R. Buss and Ryan Williams. Limits on alternation trading proofs for time-space lower bounds. *Computational Complexity*, 24(3):533–600, 2015.

[30] Jin-yi Cai. $\mathsf{S}_2^{\mathsf{p}}$ is subset of $\mathsf{ZPP}^{\mathsf{np}}$. *J. Comput. Syst. Sci.*, 73(1):25–35, 2007.

[31] Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved karp-lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005.

[32] Jin-yi Cai and Osamu Watanabe. On proving circuit lower bounds against the polynomial-time hierarchy. *SIAM J. Comput.*, 33(4):984–1009, 2004.

[33] Venkatesan T Chakaravarthy and Sambuddha Roy. Oblivious symmetric alternation. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 230–241. Springer, 2006.

[34] Venkatesan T. Chakaravarthy and Sambuddha Roy. Arthur and merlin as oracles. *Computational Complexity*, 20(3):505–558, 2011.

[35] Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:31, 2019.

[36] Lijie Chen, Ce Jin, and R Ryan Williams. Hardness magnification for all sparse np languages. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1240–1255. IEEE, 2019.

[37] Lijie Chen, Dylan M McKay, Cody D Murray, and R Ryan Williams. Relations and equivalences between circuit lower bounds and karp-lipton theorems. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[38] Lijie Chen and Roei Tell. Bootstrapping results for threshold circuits "just beyond" known lower bounds. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 34–41, 2019.

[39] Timothy Y. Chow. Almost-natural proofs. *J. Comput. Syst. Sci.*, 77(4):728–737, 2011.

[40] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[41] Peter Dixon, Aduri Pavan, and N. V. Vinodchandran. On pseudodeterministic approximation algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 61:1–61:11, 2018.

[42] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[43] Lance Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space: Time-space tradeoffs for satisfiability. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 52–60. IEEE, 1997.

[44] Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *JACM*, 52(6):835–865, 2005.

[45] Lance Fortnow and Rahul Santhanam. Robust simulations and significant separations. *Inf. Comput.*, 256:149–159, 2017.

[46] Lance Fortnow, Rahul Santhanam, and Ryan Williams. Fixed-polynomial size circuit bounds. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 19–26, 2009.

[47] Etienne Grandjean. Linear time algorithms and np-complete problems. *SIAM J. Comput.*, 23(3):573–597, 1994.

[48] Juris Hartmanis, Neil Immerman, and Vivian Sewelson. Sparse sets in NP-P: EXPTIME versus NEXPTIME. *Information and Control*, 65(2/3):158–181, 1985.

[49] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:138, 2018.

[50] Shuichi Hirahara, Igor Carboni Oliveira, and Rahul Santhanam. Np-hardness of minimum circuit size problem for OR-AND-MOD circuits. In *CCC*, pages 5:1–5:31, 2018.

[51] Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In *CCC*, pages 7:1–7:20, 2017.

[52] Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In *CCC*, volume 50, pages 18:1–18:20, 2016. Available at ECCC.

[53] John M. Hitchcock and Aduri Pavan. On the NP-completeness of the minimum circuit size problem. In *35th IARCS Conf. Found. of Software Tech. and Theoret. Comput. Sci. (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 236–245, 2015.

[54] Christian Hoffmann. Exponential time complexity of weighted counting of independent sets. *CoRR*, abs/1007.1146, 2010.

[55] Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and acˆ0[p]. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 34:1–34:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[56] Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. Np-hardness of circuit minimization for multi-output functions. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 22:1–22:36. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[57] Russell Impagliazzo. Personal Communication, 2018.

[58] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM J. Comput.*, 39(4):1637–1665, 2010.

[59] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. The power of natural properties as oracles. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 7:1–7:20, 2018.

[60] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[61] Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000.

[62] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

[63] Richard Karp and Richard Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[64] Johannes Köbler and Osamu Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998.

[65] Richard E Ladner and Nancy A Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10(1):19–32, 1976.

[66] Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.

[67] Richard J. Lipton and Ryan Williams. Amplifying circuit lower bounds against polynomial time with applications. In *CCC*, pages 1–9, 2012.

[68] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[69] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107(1):121 – 133, 1993.

[70] Dylan M McKay, Cody D Murray, and R Ryan Williams. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1215–1225, 2019.

[71] Dylan M McKay and Richard Ryan Williams. Quadratic time-space lower bounds for computing natural functions with a random oracle. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[72] Moritz Müller and Ján Pich. Feasibly constructive proofs of succinct weak circuit lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:144, 2017.

[73] Cody Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for NP and NQP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 890–901, 2018.

[74] Cody D. Murray and R. Ryan Williams. On the (non) NP-hardness of computing circuit complexity. In *CCC*, volume 33 of *LIPIcs*, pages 365–380. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[75] Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *34th Computational Complexity Conference*, 2019.

[76] Igor Carboni Oliveira and Rahul Santhanam. Hardness magnification for natural problems. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 65–76. IEEE, 2018.

[77] Jakob Pagter. On Ajtai's lower bound technique for R-way branching programs and the hamming distance problem. *Chicago J. Theor. Comput. Sci.*, 2005.

[78] Wolfgang J. Paul, Nicholas Pippenger, Endre Szemerédi, and William T. Trotter. On determinism versus non-determinism and related problems (preliminary version). In *FOCS*, pages 429–438, 1983.

[79] Alexander Razborov and Steven Rudich. Natural proofs. *JCSS*, 55(1):24–35, 1997.

[80] Rahul Santhanam. Lower bounds on the complexity of recognizing SAT by Turing machines. *Inf. Process. Lett.*, 79(5):243–247, 2001.

[81] Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

[82] Martin Sauerhoff and Philipp Woelfel. Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In *STOC*, pages 186–195. ACM, 2003.

[83] Aravind Srinivasan. On the approximability of clique and related maximization problems. *J. Comput. Syst. Sci.*, 67(3):633–651, 2003.

[84] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[85] B. A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, Oct 1984.

[86] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[87] Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[88] Dieter Van Melkebeek et al. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends® in Theoretical Computer Science*, 2(3):197–303, 2007.

[89] N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347(1-2):415–418, 2005.

[90] R. Ryan Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2):179–219, 2008.

[91] Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1):2, 2014.

[92] Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.

[93] Yaacov Yesha. Time-space tradeoffs for matrix multiplication and the discrete fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29(2):183–197, 1984.