Weizmann Institute of Science
Department of Computer Science and Applied Mathematics
Winter 2012/3

A taste of Circuit Complexity Pivoted at $\mathbf{NEXP} \not\subset \mathbf{ACC^0}$ (and more)

Gil Cohen

Preface

A couple of years ago, Ryan Williams settled a long standing open problem by showing that $\mathbf{NEXP} \not\subset \mathbf{ACC^0}$. To obtain this result, Williams applied an abundant of classical as well as more recent results from complexity theory. In particular, beautiful results concerning the tradeoffs between hardness and randomness were used. Some of the required building blocks for the proof, such as $\mathbf{IP} = \mathbf{PSPACE}$, Toda's Theorem and the Nisan-Wigderson pseduorandom generator, are well-documented in standard books on complexity theory, but others, such as the beautiful Impagliazzo-Kabanets-Wigderson Theorem, are not.

In this course we present Williams' proof assuming a fairly standard knowledge in complexity theory. More precisely, only an undergraduate-level background in complexity (namely, Turing machines, "standard" complexity classes, reductions and completeness) is assumed, but we also build upon several well-known and well-documented results such as the above in a black-box fashion. On the other hand, we allow ourselves to stray and discuss related topics, not used in Williams' proof. In particular, we cannot help but spending the last two lectures on matrix rigidity, which is related to a classical wide-open problem in circuit complexity.

I am thankful to all of the students for attending the course, conducting interesting discussions, and scribing the lecture notes (and for putting up with endless iterations): Sagie Benaim, Dean Doron, Anat Ganor, Elazar Goldenberg, Tom Gur, Rani Izsak, Shlomo Jozeph, Dima Kogan, Ilan Komargodski, Inbal Livni, Or Lotan, Yuval Madar, Ami Mor, Shani Nitzan, Bharat Ram Rangarajan, Daniel Reichman, Ron D. Rothblum, Uri Sherman, Igor Shinkar, Avishay Tal, Tal Wagner, Eylon Yogev and Asaf Ziv. I would like to thank Eli Ben-Sasson for allowing me to use his beautiful template for these lecture notes (crafted by Eyal Rozenberg). I would like to thank Oded Goldreich for coining the title of these notes and for his advice and guidance to publish them. Finally, I would like to thank my advisor, Ran Raz, for his encouragement.

Table of Contents

Lecture 1: The Limits of Diagonalization

1.1 Complexity Classes . . . 1–1
1.2 Universal Turing Machines . . . 1–2
1.3 Two Time-Hierarchy Theorems . . . 1–3
1.4 Oracles and Relativization . . . 1–6

Lecture 2: The Polynomial Hierarchy; Introduction to Circuit Complexity - Part 1

2.1 The Polynomial Hierarchy . . . 2–1
2.1.1 Complete Languages in the Polynomial Hierarchy . . . 2–5
2.2 Introduction to Circuit Complexity . . . 2–5
2.3 Circuit Lower Bounds and Separating **P** from **NP** . . . 2–9

Lecture 3: Introduction to Circuit Complexity - Part 2

3.1 Four Classical Theorems about Circuits . . . 3–1
3.2 Restricted Circuits . . . 3–6
3.3 Uniformly Generated Circuits . . . 3–9

Lecture 4: Razborov-Smolensky Theorem; Arithmetic Circuits

4.1 Razborov-Smolensky Theorem . . . 4–1
4.2 Introduction to Arithmetic Circuits . . . 4–6
4.2.1 The Determinant and the Permanent . . . 4–7
4.2.2 Bipartite Matching and the Determinant . . . 4–8

Lecture 5: Randomization in Computation

5.1 Complexity Classes for Randomized Computation . . . 5–1
5.2 Efficient Randomized Computation vs. Non-Determinism . . . 5–2
5.3 Efficient Randomized Computation vs. Non-Uniformity . . . 5–4
5.4 **P** vs. **BPP** . . . 5–7

5.5 Small-Bias Sets 5–8
5.6 Pseudorandom Generators 5–11

Lecture 6: Derandomization and Circuit Lower Bounds; Interactive Proof Systems

6.1 Interactive Proof Systems 6–3
6.2 Three Corollaries of **IP** = **PSPACE** 6–5

Lecture 7: Kabanets-Impagliazzo Theorem: Derandomization implies Circuit Lower Bounds

Lecture 8: Impagliazzo-Kabanets-Wigderson Theorem

8.1 Turing Machines that Take Advice 8–1
8.2 Infinitely Often 8–3
8.3 A Proof for the IKW Theorem 8–4

Lecture 9: NEXP $\not\subset$ ACC^0 - Part 1

9.1 A **NEXP**-Complete Language 9–1
9.2 Proof of Theorem 9.1 9–3
9.3 SYM+ Circuits and **ACC^0** Circuits. 9–5

Lecture 10:NEXP $\not\subset$ ACC^0 - Part 2

10.1 Yao-Beigel-Tarui Theorem 10–1
10.2 Non Trivial Satisfiability Algorithm for **ACC^0** 10–10

Lecture 11:Natural Proofs

11.1 More on the Largeness Condition 11–3

Lecture 12:Linear-Circuit Lower Bounds via Matrix Rigidity

12.1 Linear-Circuits 12–1
12.2 Matrix Rigidity 12–4
12.3 Valiant's Theorem 12–8

Lecture 13: Relations between Matrix Rigidity and Coding Theory

13.1 Basics of Error Correcting Codes 13–1
13.2 Rigid Matrices from Algebraic-Geometry Codes 13–3
13.3 Dvir's Approach to Matrix Rigidity 13–4
13.3.1 Local Decodability and Local Correction 13–4
13.3.2 Main Theorem 13–5
13.3.3 Discussion 13–7

References

Lecture 1
The Limits of Diagonalization

November 1st, 2012

Lecturer: Gil Cohen — Scribe: Daniel Reichman

In this lecture, we define basic complexity classes that play a central role in complexity theory. We will also touch upon a useful method for separating between complexity classes, called *diagonalization*. The use of diagonalization in complexity theory was inspired by the efficacy of this method in computability theory and in mathematical logic. Despite these success, we will demonstrate the limitation of this method in dealing with the very basic questions in complexity theory, such as the **P** vs. **NP** problem. These limitations are one of the reasons why complexity theorists study *circuits* - a major computational model in this course - instead of merely focusing on Turing machines.

1.1 Complexity Classes

A central and astonishing discovery in the study of computation is that although there are natural computational problems in the thousands, almost all of them can be classified into a small number of classes that capture the problems' computational hardness. In this section we remind the reader of some of these classes.
A function $f : \mathbb{N} \to \mathbb{N}$ is *time constructible* if for every $n, f(n) \geq n$ and there is a Turing machine that given 1^n as input, outputs $1^{f(n)}$ in time $O(f(n))$. All the functions that we will consider are time-constructible. Moreover, unless stated otherwise, all languages are subsets of $\{0,1\}^*$.

Definition 1.1. Let $T : \mathbb{N} \to \mathbb{N}$. $\mathbf{DTIME}\left(T(n)\right)$ is the class of all languages that are decidedable by a Turing machine halting within $O(T(n))$ steps.

Definition 1.2. The class **P** is defined as

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}\left(n^c\right) = \mathbf{DTIME}\left(n^{O(1)}\right).$$

In words, the class **P** captures all languages decidable in polynomial-time in the input length.

Definition 1.3. The class **EXP** is defined as

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}\left(2^{n^c}\right) = \mathbf{DTIME}\left(2^{n^{O(1)}}\right).$$

Definition 1.4. The class **E** is defined as $\mathbf{DTIME}\left(2^{O(n)}\right)$. That is, **E** is the class of all languages that can be solved in time 2^{cn} on inputs of length n, where c is some positive constant. This is sometimes referred to as *linear-exponential time.*

Similar definitions apply for non-deterministic complexity classes. For example,

$$\mathbf{NP} = \bigcup_{c \geq 1} \mathbf{NTIME}\left(n^c\right) = \mathbf{NTIME}\left(n^{O(1)}\right), \tag{1}$$

e.g., all languages that are decidable by *non-deterministic* Turing machines, whose longest computation path is polynomially bounded in the input length. In other words, **NP** is the class of all languages for which membership can be *verified* in polynomial-time (see also Definition 2.1). **NEXP** is the class of all languages decidable by a non-deterministic Turing machine whose longest computation path is bounded exponentially in the input length. Put differently, **NEXP** is the class of all languages for which membership can be verified in exponential time.

1.2 Universal Turing Machines

We will use the fact that every Turing machine M can be encoded by a finite string $x \in \{0,1\}^*$. When M is encoded by a string x, we denote this encoding by M_x to illustrate that the encoding is of M. A marvelous, yet simple to verify fact, is that there exists a *Universal Turing Machine U* which, when given an encoding M_x of M and a string $y \in \{0,1\}^*$, outputs $U(M_x, y) = M(y)$. That is, U simulates the computation of M on y and returns $M(y)$, if it is defined.
We make use of two special properties of the encoding of Turing machines that will be important for us. First, *every* string encodes some Turing machine. Second, *every* Turing machine is encoded by *infinitely* many strings. The latter, seemingly odd statement, is quite useful in proof by diagonalization. In fact, we will use this statement twice in this lecture (see Theorem 1.6 and Theorem 1.7).
We formalize the above in the following theorem, whose proof is omitted (see Chapter 1 of Goldreich [2008] or Arora and Barak [2009] for details).

Theorem 1.5. *There exists a Turing machine U, called the* Universal Turing Machine*, such that for every $x, y \in \{0,1\}^*$ it holds that $U(M_x, y) = M(y)$. Furthermore,* *if the running time of M on y is bounded by T, then the running time of U on M_x, y is bounded by $c \cdot T \log T$, where c is some positive constant independent of y.*

Note: the constant c in the above theorem depends only on the number of states and tapes in M. Clearly the running time of the universal Turing machine must somehow depend on the encoded machine M.
We also note that the name of this section is in plural (that is, the section is not titled "The Universal Turing Machine") because one can define a universal Turing machine in different models of computation. For example, a universal Turing machine for non-deterministic computation (which we will use in the proof of Theorem 1.7).

1.3 Two Time-Hierarchy Theorems

In complexity theory, one is usually interested in the tradeoff between different resources such as time, space, non-determinism, etc. For example, the famous **P** vs. **NP** problem asks whether non-determinism adds computational power to polynomial-time computation. A harsh lesson we learned in the past few decades is that such results are usually extremely hard to obtain.
An easier problem is whether more of the *same* resource adds to the computational power. Such theorems are called hierarchy theorems. A key technique in proving such theorems is *diagonalization* - a proof technique that was introduced by Cantor in showing that there are real numbers that are irrational. Diagonalization is widely used in computability theory (for example, showing that the Halting Problem is undecidable). It is also used in mathematical logic in proving impossibilty results such as Gödel incompleteness theorem (see Chapter 6 of Papadimitriou [2003] for details). We now state the deterministic time-hierarchy theorem.

Theorem 1.6 (Hartmanis and Stearns [1965]). *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two time constructible functions. Assume that $f(n)\log(f(n)) = O(g(n))$. Then*

$$\mathbf{DTIME}\,(f(n)) \subsetneq \mathbf{DTIME}\,(g(n)).$$

Proof. Following Arora and Barak [2009], for simplicity, we will prove this theorem for the special case $f(n) = n$ and $g(n) = n^{1.5}$. The general case can be easily derived. We define now a Turing machine D (stands for diagonalization) as follows. Given x as input, D simulates M_x on input x for $|x|^{1.2}$ steps ($|x|$ is the length of the string x). If M_x outputs a bit b and halts then D output $1 - b$ (and, of course, halts as well). Otherwise, D outputs some arbitrary value, say 0, and halts. We note that the language L, decided by D, belongs to $\mathbf{DTIME}\,(n^{1.5})$. Indeed, by Theorem 1.5, the

simulation of M_x incurs only a multiplicative factor of $O(\log n)$ to the running time, and $n^{1.2} \cdot \log n = o(n^{1.5})$.
We now prove that $L \notin \mathbf{DTIME}(n)$. The proof is by contradiction. Assume there is a Turing machine M such that M decides L and halts in time $O(n)$ on all inputs of length n. When we ran the universal Turing machine U on the pair (M_x, x), the running time is bounded by $c|x| \log |x|$ for some fixed positive constant c (the value of the constant c depends on the hidden constant in the running time of M and on the constant in the running time of the Universal Turing machine U).
Take $m \in \mathbb{N}$ large enough such that $m^{1.2} > cm \log m$ and consider an encoding y of M such that $|y| > m$ (such an encoding exists as we know every Turing machine is encoded by infinitely many strings). When receiving y as input, D will compute $M(y)$ after at most $|y|^{1.2}$ steps and will output a bit different from $M(y)$. Thus $D(y) \neq M(y)$ contradicting the definition of D. This contradiction concludes the proof of the theorem. □

The proof of Theorem 1.6 is very similar to the proof by diagonalization of the Halting Problem being undecidable. The following is a hierarchy theorem for non-determinism. Its proof is also based on diagonalization, but requires a new idea. Besides having a beautiful proof, we will use this theorem in the future (e.g., in Theorem 7.1 and in Williams' theorem, Theorem 9.1).

Theorem 1.7 (Cook [1973]). *If $f, g : \mathbb{N} \to \mathbb{N}$ are time-constructible functions such that $f(n+1) = o(g(n))$, we have that*

$$\mathbf{NTIME}(f(n)) \subsetneq \mathbf{NTIME}(g(n)).$$

Proof. As in Arora and Barak [2009], for simplicity, we focus on the case $f(n) = n$, $g(n) = n^{1.5}$. The proof for the general case is similar. As a first attempt to prove this theorem, one might try and use a similar diagonalization argument as in the deterministic time-hierarchy theorem, using a universal Turing machine for non-deterministic machines (such a universal machine does exist. We will take this fact for grunted in this course). Unfortunately, a naive implementation of this idea does not quite work. The difficulty is the lack of symmetry with respect to accepting and rejecting an input in the non-deterministic model. For example, the fact that a language L is in $\mathbf{NTIME}(n)$ is not known to imply that the *complement* of L belongs to $\mathbf{NTIME}(n^{1.5})$, as for a given x, to decide $x \notin L$ requires, at least naively, going over *all* computational branches of the non-deterministic machine deciding L, which does not seem to be possible to accomplish in non-deterministic polynomial-time. For similar reasons, it is not clear whether $\mathbf{NP}$ is closed under taking complements and in fact, it is widely believed $\mathbf{NP} \neq \mathbf{coNP}$.

We define recursively a function $h : \mathbb{N} \to \mathbb{N}$ as follows.

$$h(1) = 2$$
$$h(i+1) = 2^{h(i)^{1.1}} \quad \forall i \geq 1.$$

It can be verified that for any integer n, one can find the integer i such that $h(i) < n \leq h(i+1)$ in, say, $O(n^{1.5})$ time.
The machine D is defined as follows.

1. For input x, if $x \notin 1^*$ reject. Otherwise, $x = 1^n$. Compute i such that $h(i) < n \leq h(i+1)$, and let M_i be the machine encoded by the binary representation of i.

2. If $n \in (h(i), h(i+1))$, using a non-deterministic universal Turing machine, simulate M_i on 1^{n+1} for $n^{1.2}$ steps and return an identical answer to M_i (if M_i fails to halt then halt and accept).

3. If $n = h(i+1)$ then simulate M_i for $(h(i)+1)^{1.2}$ steps. If M_i fails to halt then accept. Otherwise accept 1^n if and only if M_i rejects $1^{h(i)+1}$.

Observe that Part 3 requires going over all the computational branches of length $(h(i)+1)^{1.2}$, which can be implemented to run in time $2^{(h(i)+1)^{1.2}} = O(h(i+1)^{1.5})$. Hence, the language decided by D belongs to $\mathbf{NTIME}\,(n^{1.5})$ (as Parts 1,2 can also be computed in this time limit).
Suppose towards a contradiction that the language decided by D belongs to $\mathbf{NTIME}\,(n)$ and let M be a non-deterministic Turing machine deciding L in time cn for some positive constant c. As in the deterministic case, every non-deterministic machine is represented by infinitely many strings. In particular, M is represented as the binary encoding of a large enough i, such that for all inputs of length n larger than $h(i)$, simulating M_i takes less time than $n^{1.2}$. By the way we constructed D, for *every* $h(i) < n < h(i+1)$ it holds that

$$D(1^n) = M_i(1^{n+1}). \tag{2}$$

By the definition of M_i, for *every* $h(i) < n \leq h(i+1)$,

$$D(1^n) = M_i(1^n). \tag{3}$$

Combining Equation 2 and Equation 3 (many times) we have that $M_i(1^{h(i)+1}) = D(1^{h(i+1)})$. On the other hand, as M_i halts on $1^{h(i)+1}$ on every branch after less than

$(h(i)+1)^{1.2}$ time, we get by the way D was defined that

$$M_i(h(i)+1) \neq D(h(i+1)).$$

This contradiction concludes the proof. □

We end this section with the following corollary.

Corollary 1.8.

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}.$$

Moreover, at least one of the containments is strict.

Proof. We give only a sketch of the proof (see Chapter 7 of Papadimitriou [2003]). The first containment is trivial. As for the second one, every non-deterministic polynomial-time machine can be simulated in polynomial space (simply go over all computation branches, reusing space). The third containment holds since every polynomial space machine that halts in fact halts in exponential time (the possible number of configurations of a machine using polynomial space is exponential). However, by the Deterministic Time-Hierarchy Theorem (Theorem 1.5) $\mathbf{P} \neq \mathbf{EXP}$. Thus, one of the containments must be strict. □

While it is strongly believed that all containments in the above theorem are strict, it is not known how to prove that a *single* containment is strict. Proving it (or even proving $\mathbf{P} \neq \mathbf{PSPACE}$) would be a major breakthrough.

1.4 Oracles and Relativization

The success of diagonalization techniques in separating between complexity classes revealed in hierarchy theorems may lead one to believe that such methods can be used to settle the **P** vs. **NP** problem. However, in the 70's researchers have shown that this is not the case - provably, new ideas are required.
Informally, these kind of results, where one shows that a natural and useful technique is provably not enough to settle some problem, is referred to as a *barrier*. In this section we show a barrier for proofs by diagonalization.
The key ingredient behind the formal proof of the limitation of diagonalization lies in the use of *oracles*. Let $O \subseteq \{0,1\}^*$. Roughly speaking, an oracle Turing machine M^O is a Turing machine that can test membership to O in a single computational step. Namely, while running on an input, the machine can send a string to an oracle that decides whether this string belongs to O and return an answer to M accordingly.

Definition 1.9. An *Oracle Turing Machine* M with oracle access to a language O is a Turing machine with a special auxiliary tape (named the *oracle tape*) as well as three special states $q_{\text{yes}}, q_{\text{no}}, q_{\text{query}}$. When the machine is in q_{query}, it queries whether the string written on the oracle tape is in O by moving to q_{yes} if the string belongs to O and to q_{no} otherwise. A query step is counted as a single computational step. We denote the output of M with oracle access to O on input x, by $M^O(x)$. A similar definition applies to non-deterministic Turing machines.

Definition 1.10. Consider $O \subseteq \{0,1\}^*$. The class $\mathbf{P}^O$ is the class of languages that can be decided by an oracle Turing machine with oracle access to O that runs in polynomial-time. Similarly, the class $\mathbf{NP}^O$ is the class of all languages that can be decided by a non-deterministic Turing machine with oracle access to O, with all computational paths having polynomial length.

A close inspection on diagonalization-based proofs, such as those used in the hierarchy theorems presented at the previous section, reveals that they rely on the encoding of Turing machines that allows a universal Turing machine to simulate any other machine M' using the encoding of M' without much increase of the time complexity of M'. As it turns out, it is possible to encode oracle Turing machines such that a universal Turing machine can efficiently simulate oracle Turing machines.
The implication is clear yet powerful: any complexity result proved for ordinary Turing machines using only the existence of an encoding of Turing machines and a universal Turing machine that simulates them in a black-box manner, carries over to oracle machines as well. In this case, we say that the result *relativize*, because the same result holds relative to any oracle. If we are able to find, for two complexity classes C_1 and C_2, two oracles O_1 and O_2 such that ${C_1}^{O_1} = {C_2}^{O_1}$ but ${C_1}^{O_2} \neq {C_2}^{O_2}$, then equality and inequality of C_1 and C_2 do not relativize. In other words, plain diagonalization cannot be used to prove or disprove $C_1 = C_2$.
As we see next, Baker et al. [1975] proved that this is the situation for **P** and **NP**. We note that there are many more examples of similar flavor. For example, for the polynomial hierarchy **PH** that will be defined in Lecture 2 (see Section 2.1), there exist oracles O_1, O_2, such that $\mathbf{PH}^{O_1} = \mathbf{PSPACE}^{O_1}$ whereas $\mathbf{PH}^{O_2} \neq \mathbf{PSPACE}^{O_2}$ (see Yao [1985]).

Theorem 1.11 (Baker et al. [1975])**.** *There exist oracles* $A, B \subseteq \{0,1\}^*$ *such that*

$$\mathbf{P}^A = \mathbf{NP}^A \quad \textit{and yet} \quad \mathbf{P}^B \neq \mathbf{NP}^B.$$

Proof. We first show how to construct an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$. The idea is to construct a powerful oracle that will "cancel" the non-determinism advantage **NP** machines have over deterministic polynomial-time computation.

Take A to be an arbitrary **EXP**-complete language (in this course, completeness is always with respect to polynomial-time reductions). Such a language exists. An oracle access to A enables a Turing machine to solve any problem in **EXP** in a single query. Hence $\mathbf{EXP} \subseteq \mathbf{P}^A$. On the other hand, one can simulate in deterministic exponential-time a non-deterministic polynomial-time Turing machine, with oracle access to A. This is done by going over all computation branches (in exponential time) and answer A queries (where every query can be answered in exponential time). Thus $\mathbf{NP}^A \subseteq \mathbf{EXP}$. As clearly $\mathbf{P}^A \subseteq \mathbf{NP}^A$ for any oracle A, the result follows.
As for the second part of the theorem, for a given language B we define U_B to be the unary language

$$U_B = \{1^n | \text{ there exists a string of length } n \text{ in } B\}.$$

Clearly $U_B \in \mathbf{NP}^B$, as a non-deterministic polynomial-time machine, can guess a string s in B (if such exists) and use the oracle to verify whether $s \in B$. We show how to construct a language B such that $U_B \notin \mathbf{P}^B$.
Given $i \in \mathbb{N}$, let M_i be the oracle Turing machine encoded by the binary representation of i. B is constructed iteratively (initially B is set to be the empty language), such that M_i will not decide U_B in less than $2^n/10$ time steps, when running on inputs of length n.
At the i^{th} stage, for a finitely many strings a decision has been made whether they belong to B or not. Call strings that are decided to be in B - *black*, strings that are decided not to be in B - *white* and strings for which no decision has been made yet - *gray*. Take n to be larger than the length of all strings that are either black or white. Run M_i on input 1^n for exactly $2^n/10$ steps. When M_i queries strings that are known to belong (or not to belong) to B (e.g., black or white), the oracle answers consistently with B. If M_i uses the oracle to query strings that are not yet decided to belong to B (that is, gray strings) then these strings are declared white.
Our goal is to make sure that the answer of M_i on 1^n will lead to a contradiction. By now, the decision whether to include a string to B or not was made for at most $2^n/10$ strings in $\{0,1\}^n$. Every string of length n we have encountered thus far is white, by the way our iterative construction work. If M_i accepts 1^n then we declare all of the gray strings in $\{0,1\}^n$ as *not* belonging to B (that is, white). Otherwise, we choose a gray string of length n (note that as the number of strings of length n that we have encountered thus far is bounded by $2^n/10$, there must be at least one gray string) and declare it to be in B, that is, set it to black.
This completes the description of how B is constructed. Our construction ensures that *every* machine M with oracle access to B, running in polynomial-time, will err on 1^n for some n. Thus $U_B \notin P^B$. Even more strongly, the above proof implies that

$U_B \notin \mathbf{DTIME}\left(f(n)^B\right)$ for every time-constructible f with $f = o(2^n)$. $\square$

A neat aspect of the proof of Theorem 1.11, which shows that diagonalization is in some sense "weak", is that it uses diagonalization!

Lecture 2

The Polynomial Hierarchy; Introduction to Circuit Complexity - Part 1

November 8th, 2012

Lecturer: Gil Cohen Scribe: Tom Gur

In Lecture 1 we talked about diagonalization. We discussed the seminal result by Baker et al. [1975] (see Theorem 1.11), that shows that resolving the **P** versus **NP** problem, in some sense, cannot be accomplished using only black-box simulations of Turing machines, a property of diagonalization-based proofs. This result gives a central motivation for the study of circuits, which, as we will see, are more amendable to white-box inquiry than Turing machines.
In this lecture we will complete some background and cover the basics of the *Polynomial Hierarchy* (see Section 2.1). We then start studying the theory of circuits (see Section 2.2).

2.1 The Polynomial Hierarchy

The *Polynomial Hierarchy*, denoted by **PH**, introduced by Meyer and Stockmeyer [1972], is a hierarchy of complexity classes that generalize the classes **P**, **NP** and **coNP**.[*] Before we give the formal definition, let us begin with a couple of motivating examples. Consider the language

$$\mathsf{CLIQUE} = \{(G,k) \mid G=(V,E) \text{ has a clique of size at least } k\}.$$

Note that $(G,k) \in \mathsf{CLIQUE}$ if and only if $\exists S \subseteq V$ such that $|S|=k$ is a clique. Hence, the condition can be expressed as an existential (*First Order Logic*) formula, where the quantified condition (i.e., $|S|=k$ is a clique) can be computed in polynomial time. In general, recall the definition of **NP** (the definition for **NP** below is, of course, equivalent to the somewhat less formal definition in Equation 1 from Lecture 1).

Definition 2.1. A language L is in **NP** if and only if there exist polynomials p and q, and a deterministic Turing machine M, such that

- $\forall x, w \in \{0,1\}^*$, the machine M runs in time $p(|x|+|w|)$ on input (x,w).

[*]One can view the *Polynomial Hierarchy* as the resource-bounded counterpart to the *Arithmetical Hierarchy* (the *Kleene-Mostowski Hierarchy*) from mathematical logic, which classifies certain sets based on the complexity of formulas that define them.

- $x \in L \iff$ there exists a proof string (witness) w of length $q(|x|)$ such that $M(x, w) = 1$.

According to Definition 2.1, we see that $\mathsf{CLIQUE} \in \mathbf{NP}$. Now consider a natural variant of CLIQUE, namely,

$$\mathsf{EXACT-CLIQUE} = \{(G, k) \mid \text{The largest clique in } G = (V, E) \text{ has size } \textit{exactly } k\}.$$

Note that $(G, k) \in \mathsf{EXACT-CLIQUE}$ if and only if $\exists S \subseteq V$ such that $|S| = k$ is a clique and $\forall T \subseteq V$, $|T| > k$ is not a clique. Hence, the condition can be expressed by a formula with one existential quantifier and one universal quantifier. For the $\mathsf{EXACT-CLIQUE}$ language, the order of the quantifiers does not matter, as the conditions on S, T are independent. However, this is not always the case; e.g., consider the language

$$\mathsf{MIN-EQ-DNF} = \{(\phi, k) \mid \exists \text{ DNF } \psi \text{ of size } k \text{ that is equivalent to } \phi\}.$$

Here, $(\phi, k) \in \mathsf{MIN-EQ-DNF}$ if and only if $\exists$ DNF ψ of size k such that $\forall x\ \phi(x) = \psi(x)$. The class of all languages that can be expressed by an existential quantifier followed by a universal quantifier is denoted by $\mathbf{\Sigma_2^p}$. [*] It is known Umans [1998] that $\mathsf{MIN-EQ-DNF}$ is a complete language for Σ_2^p.

Definition 2.2. A language L is in $\mathbf{\Sigma_2^p}$ if and only if there exist polynomials p, q_1, q_2, and a deterministic Turing machine M, such that

- $\forall x, w, z \in \{0,1\}^*$, the machine M runs in time $p(|x|+|w|+|z|)$ on input (x, w, z).
- $x \in L \iff \exists w \in \{0,1\}^{q_1(|x|)}\ \forall z \in \{0,1\}^{q_2(|x|)}\ M(x, w, z) = 1$.

If we change the order of the quantifiers, we get the class $\mathbf{\Pi_2^p}$. Formally,

Definition 2.3. A language L is in $\mathbf{\Pi_2^p}$ if and only if there exist polynomials p, q_1, q_2, and a deterministic Turing machine M, such that

- $\forall x, w, z \in \{0,1\}^*$, the machine M runs in time $p(|x|+|w|+|z|)$ on input (x, w, z).
- $x \in L \iff \forall w \in \{0,1\}^{q_1(|x|)}\ \exists z \in \{0,1\}^{q_2(|x|)}\ M(x, w, z) = 1$.

In a similar fashion, for every $i \in \mathbb{N}$ we define the class $\mathbf{\Sigma_i^p}$ as the extension of $\mathbf{\Sigma_2^p}$ for formulas with i alternating quantifiers, starting with an existential quantifier, and $\mathbf{\Pi_i^p}$ as the extension of $\mathbf{\Pi_2^p}$ for formulas with i alternating quantifiers, starting with an universal quantifier. Note that $\mathbf{\Sigma_0^p} = \mathbf{\Pi_0^p} = \mathbf{P}$, $\mathbf{\Sigma_1^p} = \mathbf{NP}$, and $\mathbf{\Pi_1^p} = \mathbf{coNP}$.

[*] In the language of mathematical logic, $\mathbf{\Sigma_2^p}$ is a class of *Second Order Logic* formulas.

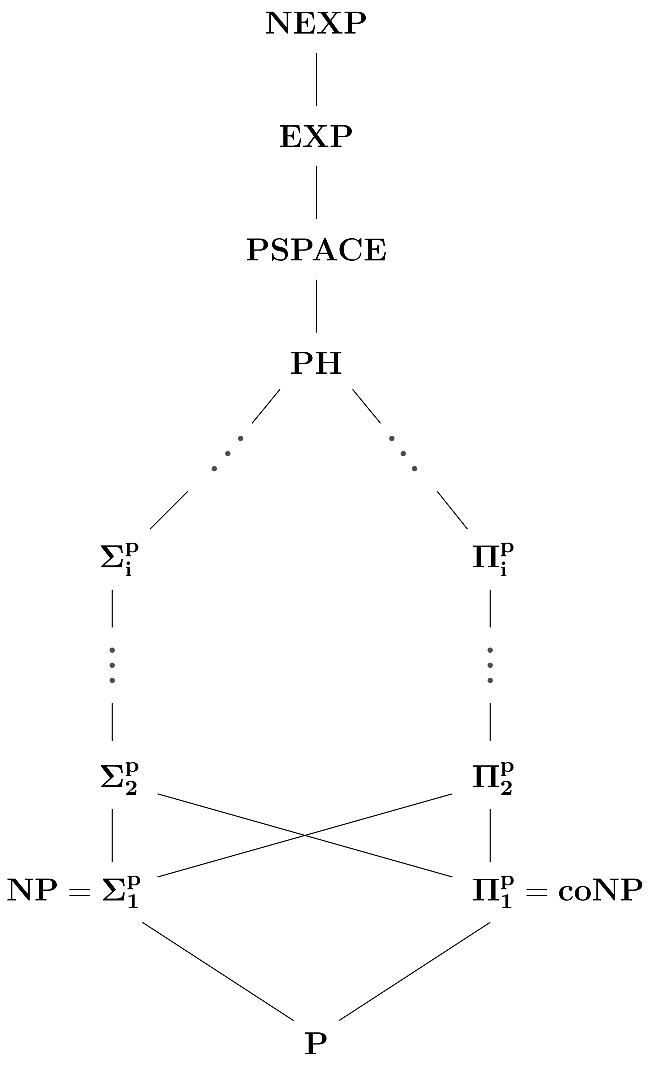


Figure 1: The Polynomial Hierarchy.

We can also formulate these classes, known as the *levels* of the Polynomial Hierarchy, in terms of *oracles*: For example, it can be shown that $\mathbf{\Sigma_2^p} = \mathbf{NP^{NP}}$. More generally, for $i \geq 0$ it holds that

$$\mathbf{\Sigma_{i+1}^p} = \mathbf{NP}^{\mathbf{\Sigma_i^p}},$$
$$\mathbf{\Pi_{i+1}^p} = \mathbf{coNP}^{\mathbf{\Sigma_i^p}}.$$

See Theorem 5.12 in Arora and Barak [2009] for a proof of the equivalence between the two definitions.

Definition 2.4. The *Polynomial Hierarchy* is defined by

$$\mathbf{PH} = \bigcup_{i \geq 1} \mathbf{\Sigma_i^p}.$$

It is easy to show that **PH** is contained within **PSPACE**, but it is not known whether the two classes are equal.[*] The following lemma states that if a level of the Polynomial Hierarchy is closed under complement then the Polynomial Hierarchy collapses to that level.

Lemma 2.5. *For any $i \in \mathbb{N}$, $\mathbf{\Sigma_i^p} = \mathbf{\Pi_i^p} \implies \mathbf{PH} = \mathbf{\Sigma_i^p}$. In this case we say that the Polynomial Hierarchy collapses to its i^{th} level.*

Proof. We start by making the following observation (see Proposition 3.9 in Goldreich [2008] for a full proof):

Observation 2.6 (rephrased). For every $k \geq 0$, a language L is in $\mathbf{\Sigma_{k+1}^p}$ if and only if there exists a polynomial p and a language $L' \in \mathbf{\Pi_k^p}$ such that

$$L = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in L'\}.$$

With this observation in mind, assume that $\mathbf{\Sigma_i^p} = \mathbf{\Pi_i^p}$. We start by showing that $\mathbf{\Sigma_{i+1}^p} = \mathbf{\Sigma_i^p}$. For any language $L \in \mathbf{\Sigma_{i+1}^p}$, by the aforementioned observation, there exists a polynomial p and a language $L' \in \mathbf{\Pi_i^p}$ such that

$$L = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in L'\}.$$

By the hypothesis $L' \in \mathbf{\Sigma_i^p}$, and so (using the observation and $i \geq 1$) there exists a polynomial p' and a language $L'' \in \mathbf{\Pi_{i-1}^p}$ such that

$$L' = \{x' : \exists y' \in \{0,1\}^{p'(|x'|)} \text{ s.t. } (x',y') \in L''\}.$$

Hence,

$$L = \{x : \exists y \in \{0,1\}^{p(|x|)} \exists z \in \{0,1\}^{p'(p(|x|))} \text{ s.t. } ((x,y),z) \in L''\}.$$

By collapsing the two adjacent existential quantifiers and using the aforementioned observation, we conclude that $L \in \mathbf{\Sigma_i^p}$. Then, we note that $\mathbf{\Sigma_{i+1}^p} = \mathbf{\Sigma_i^p}$ implies $\mathbf{\Sigma_{i+2}^p} = \mathbf{\Sigma_{i+1}^p}$ (again, by the aforementioned observation), and likewise $\mathbf{\Sigma_{j+2}^p} = \mathbf{\Sigma_{j+1}^p}$ for every $j \geq i$. Hence $\mathbf{PH} = \mathbf{\Sigma_i^p}$. □

[*]One useful reformulation of this problem is that **PH** = **PSPACE** if and only if second-order logic over finite structures gains no additional power from the addition of a transitive closure operator.

As an immediate corollary of Lemma 2.5 we get:

Corollary 2.7. $\mathbf{P} = \mathbf{NP} \implies \mathbf{P} = \mathbf{PH}$.

Namely, if one quantifier "doesn't help" then no constant number of them will.

2.1.1 Complete Languages in the Polynomial Hierarchy

We observe that if $\mathbf{PH} = \cup_{i \geq 1} \mathbf{\Sigma_i^P}$ has any complete language L then there exists an i such that L is complete for $\mathbf{\Sigma_i^P}$. As a consequence, the Polynomial Hierarchy would collapse to its i^{th} level. Thus, we believe that **PH** does not have a complete language. Since there are **PSPACE**-complete languages, we infer that if $\mathbf{PSPACE} = \mathbf{PH}$ then the Polynomial Hierarchy must collapse. This is served as an evidence for $\mathbf{PH} \neq \mathbf{PSPACE}$.

On the other hand for every $i \geq 1$ there exists a complete languages for $\mathbf{\Sigma_i^P}$ as well as for $\mathbf{\Pi_i^P}$. These complete languages are based on *Totally Quantified Boolean Formulas* (hereafter TQBF). A TQBF is a Boolean formula wherein all of the variables are quantified. We notice that since there are no free variables, a TQBF is either true or false. For example,

$$\forall x \exists y \ \ (x \wedge y) \vee (\bar{x} \wedge \bar{y})$$

is a TQBF that is always true, as indeed, for every x there exists a y that equals to x. On the other hand, the TQBF

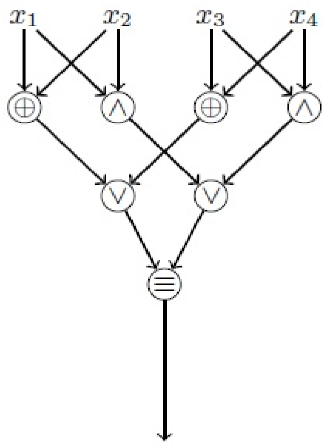
$$\exists x \forall y \ \ (x \wedge y) \vee (\bar{x} \wedge \bar{y})$$

is false (well, unless the set from which x, y are drawn is a singleton). For every $i \in \mathbb{N}$ we define TQBF_i as the set of all true TQBF that contain i quantifiers (starting with an existential quantifier). For every $i \in \mathbb{N}$, TQBF_i is a $\mathbf{\Sigma_i^P}$-complete language (where completeness is, as always in these notes, defined by polynomial time reductions), while TQBF is **PSPACE**-complete Stockmeyer and Meyer [1973] (for a proof, see e.g., Theorem 5.15 in Goldreich [2008]).

2.2 Introduction to Circuit Complexity

An n-input *Boolean circuit* is a directed acyclic graph with n *sources* and one *sink*. All non-source vertices are called *gates* and are labeled with one of $\{\vee, \wedge, \neg\}$, i.e., disjunction (logical OR), conjunction (logical AND) and negation gates. The in-degree of the negation gates is always 1. In this lecture, we consider disjunction and conjuction with in-degree 2.

The *depth* of the circuit C, denote by depth(C) is the number of edges in the longest path between the sink and a source. The *fan-in* is the maximum in-degree of the graph. The *fan-out* is the maximum out-degree of the gates in the graph. The size of a circuit C, denoted by size(C), is the number of wires in the graph.
In order to evaluate a circuit on an input $x = (x_1, \dots, x_n)$, for every vertex of the circuit we give a value as follows: if the vertex is the i^{th} source, then its value is the i^{th} bit of the input (that is, x_i). Otherwise the value is defined recursively by applying the vertex's logical operation on the values of the vertices connected to it. The output of the circuit is the value of the sink.
For example, consider the following circuit (taken from Williams [2011a]).



This circuit is the smallest possible circuit (found by Kojevnikov et al. [2009] using a SAT solver) that computes the MOD$_3$ function, where MOD$_3 : \{0,1\}^4 \to \{0,1\}$ is defined by

$$\text{MOD}_3(x_1, x_2, x_3, x_4) = 1 \iff x_1 + x_2 + x_3 + x_4 \pmod 3 = 0.$$

Let us consider Boolean circuits that compute functions $f : \{0,1\}^n \to \{0,1\}$. We show the following basic facts regarding the representation of Boolean functions by Boolean circuits.

Theorem 2.8. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a Boolean circuit of size $O(n \cdot 2^n)$.*[*] *Moreover, "most" (an arbitrary constant fraction) functions require circuits of size $\Omega(2^n/n)$.*

[*]In fact we can do even better: Lupanov [1958] proved that every Boolean function on n variables can be computed by a circuit with size $(1 + \alpha_n)\frac{2^n}{n}$, where $\alpha_n \sim \frac{\log n}{n}$ (see Theorem 1.15 in Jukna [2012]).

Proof. Given a function $f : \{0,1\}^n \to \{0,1\}$, we can write down its truth table in *Conjunction Normal Form* (CNF). Then we can build a logarithmic depth circuit (note that we can implement an AND gate with n inputs using a tree of fan-in 2 AND gates, with size $O(n)$ and depth $O(\log n)$) that expresses the aforementioned CNF. Since the size of the truth table is 2^n, the size of the circuit will be $O(n \cdot 2^n)$.
For the "moreover" part, let us count and compare the number of possible circuits of size s, and Boolean functions with n variables. On one hand, we can bound the size of the representation of a size s Boolean circuit by $O(s \log s)$ (simply by listing each gate and a pointer to its 2 neighbors, each described by $\log s$ bits). On the other hand, the size of the description of the truth table of a Boolean function on n bits is 2^n. If $O(s \log s) < 2^n$, then there are not enough Boolean circuits to compute all functions on n variables. Thus, an arbitrary fraction of the Boolean functions on the hypercube requires size that is proportional to $2^n/n$. □

While a Turing machine can handle inputs of every length, Boolean circuits can only get inputs of a fixed-length. Hence, the computational model of Boolean circuits is defined as a family of circuits $\mathcal{C} = \{C_n\}_{n=1}^{\infty}$, where the circuit C_n has n inputs. Given x, the computation is done by applying $C_{|x|}$ to x. This kind of computational model is called *non-uniform*, since it allows a different treatment for inputs of varying length, or infinite number of algorithms, if you will.
The non-uniformity of Boolean circuits gives a surprisingly strong power to the model. In fact, it can actually solve undecidable languages. To see this, consider an undecidable language encoded in unary. Since we can design a circuit per input length, in the case of unary languages (where there is only one input of each length) we can fit a degenerate circuit (that simply returns the right answer) for each input.
This irksome strength of circuits does not bother us so much, mainly because our concern will be to prove that small circuits cannot accomplish certain tasks, and so, we do not care if very small circuits can accomplish tasks we are not interested in to begin with. Let us now formalize what we mean by "small circuits".

Definition 2.9. Given a function $s : \mathbb{N} \to \mathbb{N}$, a language L is in $\mathsf{SIZE}(s(n))$ if and only if there exists a family of Boolean circuits $\mathcal{C} = \{C_n\}_{n=1}^{\infty}$ that decides L, such that $\text{size}(C_n) \le s(n)$ for all n.

Definition 2.10. The class **P/poly**,[*] is defined as the class of languages decided by

[*]The name **P/poly** comes from an equivalent complexity class: the class of languages recognized by a polynomial-time Turing machine with a polynomial-bounded advice string. We will address this definition later in the course (see Section 8.1).

families of circuits of polynomial size, namely,

$$\mathbf{P/poly} = \bigcup_{c \geq 1} \mathsf{SIZE}(n^c).$$

Similarly to the time hierarchy theorems (Theorem 1.6, Theorem 1.7), we have a *size hierarchy theorem.*

Theorem 2.11. *If $n \leq s(n) \leq \frac{2^n}{4n}$, then $\mathsf{SIZE}(s(n)) \subsetneq \mathsf{SIZE}(4 \cdot s(n))$.*

For simplicity we prove a somewhat weaker theorem, namely, that for large enough n, $\mathsf{SIZE}(n^2) \subsetneq \mathsf{SIZE}(n^3)$. The proof of Theorem 2.11 can be found in Jukna [2012], Chapter 1.

Proof. By Theorem 2.8, there exists a constant c such that for every ℓ there exists a function $h_\ell : \{0,1\}^\ell \to \{0,1\}$ that cannot be computed by a circuit of size $2^\ell/(c \cdot \ell)$. Let $\ell = \ell(n)$ be the smallest number such that $2^\ell/(c \cdot \ell) > n^2$. Consider the function $f_n : \{0,1\}^n \to \{0,1\}$ that applies h_ℓ to the first ℓ (out of n) inputs.
By Theorem 2.8, there exists a constant d for which h_ℓ can be computed by a circuit C_ℓ of size at most $d \cdot 2^\ell \ell$. Thus, f_n can be computed by a circuit C_n of size at most $d \cdot 2^\ell \ell \leq n^3$, where the inequality holds for large enough n (as a function of the two constants c, d).
Thus, the language L, decided by the family of circuits $\{C_n\}_n$, is in $\mathbf{SIZE}(n^3) \setminus \mathbf{SIZE}(n^2)$, which completes the proof. □

An important fact concerning polynomial-size Boolean circuits is their ability to emulate any (deterministic) polynomial-time Turing machine. This is captured by the following theorem.

Theorem 2.12. $\mathbf{P} \subset \mathbf{P/poly}$.

Proof. We follow the proof in Arora and Barak [2009]. Note that in this discussion we allow using circuits with many outputs. Recall (by the proof of the Cook-Levin theorem) that one can simulate every time $O(T(n))$ Turing machine M by an oblivious Turing machine $\tilde{M}$ (a machine whose head movement depends only on the input *length*) running in time $O(T(n)^2)$ (or even $O(T(n) \log T(n))$ if we try harder). Thus, it suffices to show that for every oblivious $T(n)$-time Turing machine M, there exists an $O(T(n))$-sized circuit family $\{C_n\}_{n \in \mathbb{N}}$ such that $C_{|x|}(x) = M(x)$ for every $x \in \{0,1\}^*$.
Let M be such an oblivious Turing machine, that has k tapes. Let $x \in \{0,1\}^*$ be some input for M and define the transcript of M's execution on x to be the sequence $z_1, \ldots, z_{T(n)}$ of snapshots (the machine's state and symbols read by all heads) of the

execution at each step in time. We can encode each such snapshot z_i by a constant-size binary string, and furthermore, we can compute the string z_i based on k bits from the input x, the previous snapshot z_{i-1} and the snapshots $z_{i_1}, \dots, z_{i_k}$, where z_{i_j} denotes the last step that M's j^{th} head was in the same position as it is in the i^{th} step. Because these are only a constant number of strings of constant length, this means that we can compute z_i from these previous snapshots using a constant-size circuit (essentially due to Theorem 2.8, which promise us we can compute any function).
The composition of all these constant-size circuits gives rise to a circuit that computes from the input x the snapshot $z_{T(n)}$ of the last step of M's execution on x. There is a simple constant-size circuit that, given $z_{T(n)}$, outputs 1 if and only if $z_{T(n)}$ is an accepting snapshot (in which M outputs 1 and halts). Thus, there is an $O(T(n)^2)$-sized circuit C_n such that $C_n(x) = M(x)$ for every $x \in \{0,1\}^n$. $\square$

Note that the proof of Theorem 2.12 shows that not only there exists a family of polynomial-size circuits that decides any language in **P**, but in fact, given the machine M and $n \in \mathbb{N}$, the circuit C_n can be computed by a Turing machine in poly(n) time. In this case we say that $\{C_n\}_{n\in\mathbb{N}}$ can be *uniformly generated* (see Section 3.3 for some more information about the circuits uniformity.)

2.3 Circuit Lower Bounds and Separating P from NP

Although Theorem 2.12 is quite simple to prove, it yields a new route for attacking the **P** vs **NP** problem, or more precisely, to try and separate **P** from **NP**. Indeed, by Theorem 2.12, in order to show that $\mathbf{P} \neq \mathbf{NP}$, it suffices to show that there exists a language in **NP** that is not in **P/poly**, that is, $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. This type of result is called *circuit lower bounds*, as we are interested in proving a lower bound on the size of a circuit (or circuit family to be more precise) for some language (in **NP** in this case).
The current state of our knowledge of circuit lower bounds is very meagre. We can't even show that **NEXP** (and even $\mathbf{EXP}^{\mathbf{NP}}$) is not contained in **P/poly**. The smallest class we do know of that is not contained in **P/poly** is $\mathbf{MA}_{\mathbf{EXP}}$ Buhrman et al. [1998]; we will prove this statement later during the course (see Theorem 6.13). In terms of things we have learned so far, we can only show that $\mathbf{NEXP}^{\mathbf{NP}}$ is not contained in **P/poly**. In fact, an easier exercise is to show that $\mathbf{EXPSPACE} \not\subset \mathbf{P/poly}$.
Williams's result (Theorem 9.1), which is the main topic of this course, is a circuit lower bound. It shows that some type of circuits (which we will define in the next lecture) cannot compute some language in **NEXP**.

LECTURE 3

INTRODUCTION TO CIRCUIT COMPLEXITY - PART 2

NOVEMBER 15TH, 2012

LECTURER: Gil Cohen SCRIBE: Anat Ganor, Elazar Goldenberg

3.1 Four Classical Theorems about Circuits

In this section we prove four classical theorems concerning circuits. These theorems were chosen out of many beautiful and useful theorems. The chosen four give further insights regarding the computational power of non-uniformity, and we will actually make use of the first two later in the course.

In Lecture 2 we saw that $\mathbf{P} \subset \mathbf{P/poly}$ (see Theorem 2.12). Therefore, one way to prove that $\mathbf{P} \neq \mathbf{NP}$ is to show that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. Is this a reasonable route for separating $\mathbf{P}$ from $\mathbf{NP}$? Is it plausible that poly-size non-uniformity cannot simulate poly-time verification of proofs?

The computational power of non-uniformity is surprising. For example, we saw in Lecture 2 that there exists a family of circuits of the condescending size 1 that computes a non-decidable language. On the other hand, there is some kind of undecidability implicit in non-determinism, as indeed, a witness y for the membership of an element x can be *any* function of x, and this function $y = y(x)$ can be undecidable. Moreover, intuitively it seems impossible to "compress" the witnesses for all (exponentially many) inputs that share the same length, in such a small circuit, thus, the witnesses should have some structure to fit to the circuit.

These arguments are very weak, and are far from being formal. Do we have a more solid reason to believe that $\mathbf{NP} \not\subset \mathbf{P/poly}$? Yes! In 1980, Karp and Lipton proved that if this is not the case then the Polynomial Hierarchy collapses.

Theorem 3.1 (Karp and Lipton [1980]). $\mathbf{NP} \subset \mathbf{P/poly} \implies \mathbf{PH} = \mathbf{\Sigma_2^p}$.

Proof. By Lemma 2.5, to show that $\mathbf{PH} = \mathbf{\Sigma_2^p}$, it is enough to show that $\mathbf{\Pi_2^p} \subseteq \mathbf{\Sigma_2^p}$. Consider the $\mathbf{\Pi_2^p}$-complete language $\mathbf{\Pi_2^p}$SAT consisting of all unquantified Boolean formulas $\varphi \in$ SAT for which the following holds.

$$\forall u_1, \dots, u_n \; \exists v_1, \dots, v_m \; \varphi(u_1, \dots, u_n, v_1, \dots, v_m) = 1. \tag{4}$$

Note that φ has size poly(n, m). Consider the following quantified Boolean formula

$$\exists C \in \{0,1\}^{(n+m)^k} \; \forall u \in \{0,1\}^n \; \varphi(u, C(\varphi, u)) = 1, \tag{5}$$

where C is interpreted as a "small" circuit (polynomial in the size of φ's description and $n = |u|$) that outputs m bits, and $C(\varphi, u)$ is its evaluation on the inputs φ and u. Note that given C, the output of C on inputs φ, u can be calculated deterministically in polynomial time. This means that the language of formulas φ for which Equation 5 holds is in $\mathbf{\Sigma_2^P}$. Therefore, in order to prove that $\mathbf{\Pi_2^P}\mathrm{SAT} \in \mathbf{\Sigma_2^P}$, it is enough to prove that for every unquantified Boolean formula φ, Equation 4 holds iff Equation 5 holds. If $\mathbf{NP} \subset \mathbf{P/poly}$ then there exists a polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that for every Boolean formula φ and every partial assignment $u \in \{0,1\}^n$ to the first n variables, it holds that $C(\varphi, u) = 1$ for an appropriate circuit C in the family iff there exists an assignment $v \in \{0,1\}^m$ to the last m variables such that $\varphi(u, v) = 1$.
An algorithm that solves SAT (the decision problem) can be converted into an algorithm that outputs a satisfying assignment whenever one exists. The algorithm that finds the satisfying assignment, given that such an assignment exists, using the decision SAT solver, is in $\mathbf{P}$ and can be implemented by a polynomial-size circuit, as $\mathbf{P} \subset \mathbf{P/poly}$ (see Theorem 2.12). Therefore, we obtain from $\{C_n\}_{n\in\mathbb{N}}$ another polynomial-size circuit family $\{C'_n\}_{n\in\mathbb{N}}$ such that for every unquantified Boolean formula φ and every $u \in \{0,1\}^n$, if there exists $v \in \{0,1\}^m$ for which $\varphi(u, v) = 1$, then $C'(\varphi, u)$ outputs such a string. This implies that whenever Equation 4 is true, Equation 5 is also true. If Equation 4 is false, then for some $u \in \{0,1\}^n$ there is no string $v \in \{0,1\}^m$ such that $\varphi(u, v) = 1$. Therefore, there is no circuit that can output such a (non-existing) string, and Equation 5 is also false. $\square$

Currently, not only we do not know how to prove that $\mathbf{NP} \not\subset \mathbf{P/poly}$, but we do not even know how to show that $\mathbf{EXP} \not\subset \mathbf{P/poly}$. Therefore, it is interesting to find out what can be said if this is not the case. The following theorem shows that $\mathbf{P/poly}$ is unlikely to contain $\mathbf{EXP}$. This theorem appears in Karp and Lipton [1980], but it is attributed to Albert Meyer.

Theorem 3.2 (Meyer's Theorem)**.** $\mathbf{EXP} \subset \mathbf{P/poly} \implies \mathbf{EXP} = \mathbf{\Sigma_2^P}$.

In fact, we currently don't even know that $\mathbf{NEXP} \not\subset \mathbf{P/poly}$, and the consequences of $\mathbf{NEXP} \subset \mathbf{P/poly}$ are studied in Theorem 7.2 (which, interestingly enough, uses Theorem 3.2).

Proof of Theorem 3.2. Let $L \in \mathbf{EXP}$. Then, there exists a 2^{n^k}-time, oblivious Turing machine M that computes L, for some constant $k \in \mathbb{N}$. Fix an input string $x \in \{0,1\}^n$ and denote by $z_1, \ldots, z_{2^{n^k}}$ the snapshots of M running on input x. Assume that M has t tapes. For an index $i \in [2^{n^k}]$ of a snapshot and for $j \in [t]$, consider the location on the j^{th} tape of the relevant head during the i^{th} snapshot. Let i_j be the index of the snapshot where this location was last updated. Note that we assume M is oblivious, therefore, the relevant locations of the heads can be computed given i and do not depend on x. If $x \in L$ then for every $i \in [2^{n^k}]$, the indices $i, i_1, \ldots, i_t$ and the snapshots $z_i, z_{i_1}, \ldots, z_{i_t}$ should satisfy the following. If $i = 2^{n^k}$ then z_i should encode M outputting 1. Moreover, the value on the j^{th} tape in the location of the relevant head during the i^{th} snapshot, as written in z_{i_j}, should be consistent with what is written in z_i. It is easy to see that given x and i, one can, in exponential time, compute z_i. Therefore, as we assume $\mathbf{EXP} \subset \mathbf{P/poly}$, there exists a polynomial-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ that given x and i outputs z_i. Suppose that the size of C_n is at most n^c for some constant $c \in \mathbb{N}$. Then, for every $x \in \{0,1\}^n$, it holds that

$$x \in L \iff \exists C \in \{0,1\}^{n^c}\ \forall i \in \{0,1\}^{n^k}\ T(x, C(x,i), C(x,i_1), \ldots, C(x,i_t)) = 1,$$

where T is some polynomial-time Turing machine that checks that the transcript satisfies all local criteria described above. Note that the binary representation of the indices is polynomial in n. This implies that $L \in \mathbf{\Sigma_2^P}$, which completes the proof. □

Theorem 3.2 implies the following interesting corollary. It shows that even circuit *upper bounds* can potentially be used to separate $\mathbf{P}$ from $\mathbf{NP}$, as opposed to the *lower bound* $\mathbf{NP} \not\subset \mathbf{P/poly}$.

Corollary 3.3. $\mathbf{EXP} \subset \mathbf{P/poly} \implies \mathbf{P} \neq \mathbf{NP}$.

Proof. Assume towards a contradiction that $\mathbf{P} = \mathbf{NP}$ and $\mathbf{EXP} \subset \mathbf{P/poly}$. By Corollary 2.7, the assumption that $\mathbf{P} = \mathbf{NP}$ implies that the polynomial hierarchy collapses, i.e. $\mathbf{P} = \mathbf{PH}$. Since $\mathbf{EXP} \subset \mathbf{P/poly}$, by Theorem 3.2, $\mathbf{EXP} = \mathbf{\Sigma_2^P}$, and thus, $\mathbf{EXP} = \mathbf{PH}$. Therefore, $\mathbf{EXP} = \mathbf{P}$, which contradicts the deterministic polynomial-time hierarchy theorem (see Theorem 1.6). □

Proving $\mathbf{NP} \not\subset \mathbf{P/poly}$ means that there is $L \in \mathbf{NP}$ that cannot be computed by a family of circuits of size n^k for every k. A step towards proving this would be to prove that $\mathbf{NP} \not\subset \mathbf{SIZE}\left(n^k\right)$ for some constant $k > 1$. In such case we say that $\mathbf{NP}$ does not have *fixed* polynomial-size circuits. We currently do not know how to prove even such claim, however, an analog theorem for the class $\mathbf{\Sigma_2^P}$ was proved by Kannan.

Theorem 3.4 (Kannan [1982]). *For every $k \geq 1$, it holds that $\mathbf{\Sigma_2^P} \not\subseteq \mathbf{SIZE}\left(n^k\right)$.*

Proof. First we prove the following lemma.

Lemma 3.5. *For every $k \geq 1$, it holds that $\mathbf{\Sigma_6^P} \not\subseteq \mathbf{SIZE}\left(n^k\right)$.*

Proof. Fix $k \geq 1$. Recall that for every large enough $n \geq n_0(k) \in \mathbb{N}$, it holds that $\mathbf{SIZE}\left(n^k\right) \subsetneq \mathbf{SIZE}\left(4n^k\right)$ (see Theorem 2.11). For every $n \geq n_0(k)$, let C_n be the

first circuit in lexicographic order (according to some reasonable defined encoding) contained in $\mathbf{SIZE}\left(4n^k\right) \setminus \mathbf{SIZE}\left(n^k\right)$.
We show that the language $L_k = \{x : C_{|x|}(x) = 1\}$ is in $\mathbf{\Sigma_6^p}$. Note that in order to define L_k based on a circuit contained in $\mathbf{SIZE}\left(4n^k\right) \setminus \mathbf{SIZE}\left(n^k\right)$, we must pick a specific circuit in this set. That is why we picked the first one, according to some reasonable defined order. The following proves that $L_k \in \mathbf{\Sigma_6^p}$, merely by showing that six alternating quantifiers are expressive enough to capture the definition of L_k.

$$\begin{aligned}
&\exists C \text{ of size at most } 4n^k \\
&\forall C' \text{ of size at most } n^k \\
&\exists y \in \{0,1\}^n \text{ s.t. } C(y) \neq C'(y) \\
&\forall C'' \text{ that precedes } C \text{ in lexicographic order} \\
&\exists C''' \text{ of size at most } n^k \\
&\forall z \in \{0,1\}^n \ C''(z) = C'''(z) \\
&C_{|x|}(x) = 1
\end{aligned}$$

Given x of length n, the first three quantifiers make sure that the circuit we simulate has size at most $4n^k$, but is not equivalent to any circuit of size at most n^k. The following three quantifiers are responsible for us choosing the first lexicographic such circuit. The last line simply simulates the unique circuit that answers all criteria. This can be done in $\text{poly}(n)$ time as the circuit has size $4n^k$, and evaluating a circuit is done in time linear in its size. All other (somewhat implicit) operations such as evaluating C and C' on y, check the size of the circuits C, C', C'', etc, can also be executed in time $\text{poly}(n)$. □

To complete the proof of Theorem 3.4, note that if $\mathbf{NP} \not\subset \mathbf{P/poly}$ we are done. Otherwise, by Theorem 3.1, $\mathbf{\Sigma_2^p} = \mathbf{PH}$. In particular, we get that $\mathbf{\Sigma_6^p} = \mathbf{\Sigma_2^p}$ and Lemma 3.5 concludes the proof. □

We saw that for every constant k, there exists a language high in $\mathbf{\Sigma_2^p}$ that does not have circuits of size n^k. In the next theorem we are trying to find a language in the (potentially) smaller class $\mathbf{P}$ that does not have small circuits. We show a theorem by Schnorr [1974] that gives a lower bound of $3n - 3$ on the circuit size of PARITY on n variables. This bound is tight. The best known lower bound for a function in $\mathbf{P}$ is $5n - o(n)$, due to Iwama et al. [2002], using essentially the same method, but in a significantly more involved manner.
The following theorem uses a technique called *gate elimination*. Given a circuit for PARITY on n variables, we induce from it another circuit with exactly 3 gates less that computes PARITY on $n - 1$ variables. By continuing this way until we are left

with only 2 input variables and at least 3 gates, we in fact show that the original circuit must had at least $3n - 3$ gates. Note that when an input to a gate is constant (either 0 or 1), this gate can be removed from the circuit without changing the output of the circuit.

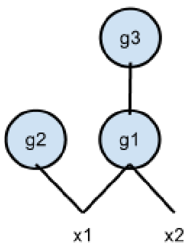
Theorem 3.6. *Let C be a Boolean circuit that computes* PARITY *over n variables. Then,* $\text{size}(C) \geq 3n - 3$.

Proof. We prove the theorem by induction on the number of variables. For $n = 2$, this is clear, as the circuit is Boolean and the PARITY function on 2 inputs is different than AND and OR. Assume that the claim holds for $n - 1$, where $n > 2$, and let C be a circuit that computes PARITY on n variables, $x_1, \ldots, x_n$. Let g_1 be a gate whose both inputs are variables and w.l.o.g its inputs are x_1, x_2.
Suppose that g_1 is an $\wedge$ gate (the case where g is an $\vee$ gate is similar). We construct a circuit that computes PARITY on $n - 1$ variables using $|C| - 3$ gates. Note that

$$\text{PARITY}(x_2, \ldots, x_n) = \text{PARITY}(0, x_2, \ldots, x_n).$$

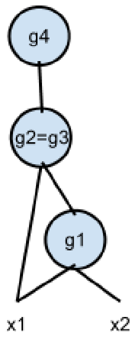
If both x_1 and x_2 are not connected to another gate, then when $x_1 = 0$, the output of C doesn't depend on x_2. Therefore, both of x_1, x_2 must be connected to another gate. Denote by $g_2 \neq g_1$ the gate that x_1 is connected to. The output of g_1 cannot be the output of C. Therefore the output of g_1 is an input to a gate denoted by $g_3 \neq g1$. We now split the analysis to two cases.

Suppose g_2 and g_3 are different gates.



When $x_1 = 0$, both g_1 and g_2 have an input 0, and we can eliminate them from the circuit. Since, the output of g_1 is 0 (recall that g_1 is an AND gate), g_3 has an input 0, so we can eliminate g_3 as well.

Suppose g_2 and g_3 are the same gate.



When $x_1 = 0$, the output of $g_2 = g_3$ doesn't depend on x_2. Therefore, the output of $g_2 = g_3$ cannot be the output of C and it must be the input to another gate g_4. When $x_1 = 0$, both g_1 and $g_2 = g_3$ have an input 0, and we can eliminate them from the circuit. Since, the output of g_1 is 0, both inputs to $g_2 = g_3$ are 0's, then also the output of $g_2 = g_3$ is 0. In this case g_4 has an input 0, and we can eliminate g_4 as well.

By the gate elimination technique (which was explain right above the proof), the proof follows. $\square$

3.2 Restricted Circuits

As mentioned, our holly grail is to show that $\mathbf{NP} \not\subset \mathbf{P/poly}$. We also discussed the implications of $\mathbf{EXP} \subset \mathbf{P/poly}$ (see Theorem 3.2). A potentially easier goal is to show that $\mathbf{NEXP} \not\subset \mathbf{P/poly}$. Unfortunately, the weakest uniform class for which we do have super-polynomial circuit lower bounds is $\mathbf{MA_{EXP}}$, which we will prove later in the course (see Theorem 6.13). Another natural subgoal would be to change the right-hand side, that is, understand natural subclasses of $\mathbf{P/poly}$. We turn to define such subclasses.

Definition 3.7. A language L is in $\mathbf{NC^0}$ if there exists a family of Boolean circuits $\{C_n\}_{n=1}^{\infty}$, a constant d, and a polynomial p such that:

- $\text{size}(C_n) \le p(n)$.
- $\forall n$, $\text{depth}(C_n) \le d$.
- The fan-in of the AND,OR gates is 2.
- $\forall x \; C_{|x|}(x) = 1 \iff x \in L$.

Since each gate has fan-in 2, an $\mathbf{NC^0}$ circuit cannot even read the whole input. In particular,

$$\text{AND}, \text{OR}, \text{PARITY}, \text{MAJORITY} \notin \mathbf{NC^0}.$$

We can only wish that all lower bounds would have been so easy!

Definition 3.8. A language L is in $\mathbf{AC^0}$ if there exists a family of circuits $\{C_n\}_{n=1}^{\infty}$, a constant d, and a polynomial p such that:

- $\text{size}(C_n) \leq p(n)$.
- $\forall n$, $\text{depth}(C_n) \leq d$.
- The fan-in of the AND,OR gates is unbounded.
- $\forall x \; C_{|x|}(x) = 1 \iff x \in L$.

Although it is clear that the AND and OR functions can be computed in $\mathbf{AC^0}$ (using just one gate), it is not at all clear whether PARITY or MAJORITY are in $\mathbf{AC^0}$. We will have to invest some effort to prove that PARITY $\notin \mathbf{AC^0}$ (see Theorem 4.3). MAJORITY has the same fate.

We define $\mathbf{NC^i}$ ($\mathbf{AC^i}$) similarly to $\mathbf{NC^0}$ ($\mathbf{AC^0}$), except that the depth is $O(\log^i n)$. Observe that every AND/OR gate with unbounded fan-in can be simulated by a circuit of polynomial size and depth $O(\log n)$ of fan-in 2 AND/OR gates, so $\mathbf{AC^i} \subseteq \mathbf{NC^{i+1}}$. Thus,

$$\mathbf{NC^0} \subseteq \mathbf{AC^0} \subseteq \mathbf{NC^1} \subseteq \mathbf{AC^1} \subseteq \mathbf{NC^2} \subseteq \cdots \subseteq \mathbf{P/poly}.$$

Are those containments strict? We do believe it, but our knowledge is very poor. We do know $\mathbf{NC^0} \subsetneq \mathbf{AC^0}$ (since, e.g., OR $\in \mathbf{AC^0} \setminus \mathbf{NC^0}$). It is easy to see that PARITY $\in \mathbf{NC^1}$. However, we will prove that PARITY $\notin \mathbf{AC^0}$ (see Theorem 4.3). Therefore, we get $\mathbf{AC^0} \subsetneq \mathbf{NC^1}$. This is all we know in terms of the strictness of the containments. In this spirit, it is worth mentioning that we don't even know how to prove that $\mathbf{NC^1} \neq \mathbf{PH}$.

The class $\mathbf{NC^1}$ is surprisingly strong. It is easy to see that addition of integers can be computed in $\mathbf{NC^1}$. It is somewhat more challenging to show that integer multiplication can be done in $\mathbf{NC^1}$. The fact that division between integers can be computed in $\mathbf{NC^1}$ is considered a classical breakthrough, discovered by Beame et al. [1986]. A classical result by Barrington [1989] shows that $\mathbf{NC^1}$ is equivalent in power to another non-uniform class that captures bounded space computation.

We define

$$\mathbf{AC} = \bigcup_{i=0}^{\infty} \mathbf{AC^i} \quad , \quad \mathbf{NC} = \bigcup_{i=0}^{\infty} \mathbf{NC^i}.$$

The last class we define is $\mathbf{ACC^0}$ which is the same as the class $\mathbf{AC^0}$ but allowing also to use counters gates. This class is natural, since PARITY $\notin \mathbf{AC^0}$, and we would like to extend the model in order that simple functions (as PARITY) could be implemented with constant depth circuits. The class $\mathbf{ACC^0}(i)$ is the same class as $\mathbf{AC^0}$ except that we allow MOD_i gates ($\mathrm{MOD}_i(x) = 0$ if $x_1 + \cdots + x_{|x|}$ is divided by i, and is equal to 0 otherwise). Formally, the class $\mathbf{ACC^0}$ is defined as:

$$\mathbf{ACC^0} = \bigcup_{m_1,\ldots,m_k \in \mathbb{N}} \mathbf{ACC^0}(m_1, \ldots, m_k).$$

In the literature our definition for $\mathbf{ACC^0}$ is sometime written as $\mathbf{ACC}$ because $\mathbf{ACC^i}$ for $i > 0$ hasn't been studied much.
In order to get some feeling of the above definition we show $\mathbf{ACC^0}(2,3) = \mathbf{ACC^0}(6)$. Observe that $\mathrm{MOD}_6(x) = 0$ iff $\mathrm{MOD}_2(x) = 0$ and $\mathrm{MOD}_3(x) = 0$, so each MOD_6 gate can be implemented by depth-2 circuit with MOD_2 and MOD_3 gates. The other direction is also easy, since $\mathrm{MOD}_3(x) = 0$ iff $\mathrm{MOD}_6(2x) = 0$, a MOD_3 gate can be easily implemented by a MOD_6 gate, and similarly MOD_2 gate can be implemented by a MOD_6 gate.
Another easy observation is that $\mathbf{ACC^0} \subseteq \mathbf{NC^1}$. We prove it by translating each MOD_i gate into a circuit with $\log(n)$-depth. It is easy to check that the resulting circuit is indeed in $\mathbf{NC^1}$. Formally, for every constants $i, j \in \mathbb{N}$, we define the gate $\mathrm{MOD}_{i,j}(x) = 0$ iff $x_1 + \cdots + x_{|x|}$ has residue j in division by i. We prove by induction on n that $\mathrm{MOD}_{i,j}$ gate can be computed by a depth $O(\log n)$-circuit. For simplicity we assume $n = 2^k$.
For $n = 2$, the claim is true. Assume that for every constants $i, j \in \mathbb{N}$ we have a circuit of depth $O(\log n)$ that computes $\mathrm{MOD}_{i,j}(x)$ for x of length n. Consider x of length $2n$. It holds that

$$\mathrm{MOD}_{i,j}(x) = 1 \iff \bigvee_{k=0,\ldots,i-1} (\mathrm{MOD}_{i,k}(x_1, \ldots, x_n) \wedge \mathrm{MOD}_{i,j-k}(x_{n+1}, \ldots, x_{2n})),$$

where the subtraction $j - k$ is done modulo i. Using this observation it is easy to see how to build a circuit of depth $O(\log n)$ that computes $\mathrm{MOD}_{i,j}(x)$ for x of length $2n$.
Now let us briefly summarize two main results associated with those classes. The first result is by Razborov [1987] which was later simplified by Smolensky [1987]. The result shows that PARITY $\notin \mathbf{ACC^0}(p)$ for all odd prime p. In particular, this result reproves the result of Furst et al. [1984] showing that PARITY $\notin \mathbf{AC^0}$. We prove this in Theorem 4.3.
Theorem 4.3 states that, as expected, $\mathbf{ACC^0}(p)$, for a prime $p \neq 2$, is not so powerful, as it cannot even compute a simple function like PARITY. What happens when we

allow for two types of counter gates, or equivalently (as shown above) one counter that is not a prime power, say 6. Is there an $\mathbf{ACC^0}(6)$ circuit for SAT?
Barrington [1989] raised the following open problem: are all languages in **NEXP** solvable by polynomial sized depth-3 circuits with unbounded fan-in AND, OR, NOT and MOD_6 gates? The expected negative answer had to wait for more than 20 years.

Theorem 3.9 (Williams [2011b]). $\mathbf{NEXP} \not\subset \mathbf{ACC^0}$

Presenting Williams' proof is the main goal of this course, and we build our way towards it.

3.3 Uniformly Generated Circuits

In some cases one is interested in circuits that can be constructed *uniformly*, by a Turing machine. That is, given n as input, the Turing machine should print out the n^{th} circuit C_n in some family of circuits $\{C_n\}$. In such case we say that the family of circuits $\{C_n\}$ is uniformly generated. More precisely, if the Turing machine runs in poly(n)-time, the generated circuit family is called **P**-uniform. This notion was introduced by Beame et al. [1986].
Obviously, there is no point talking about families of circuits in **P/poly** that are **P**-uniform. Indeed, the class of languages decided by these families is exactly **P**. Nevertheless, there is much interest, for example, in families of circuits of low-depth (say, $\mathbf{NC^1}$) that are **P**-uniform (we believe that $\mathbf{P} \neq \mathbf{NC^1}$, and even that $\mathbf{P} \neq \mathbf{NC}$, though currently we are unable to separate even **PH** from $\mathbf{NC^1}$).
There are other, stronger notions of uniformity, that is more suitable for small circuits, such as $\mathbf{AC^0}$. In this course however, we will not cover this theory.

Lecture 4
Razborov-Smolensky Theorem; Arithmetic Circuits

November 22nd, 2012

Lecturer: Gil Cohen   Scribe: Rani Izsak, Ilan Komargodski

This lecture consists of two main parts. The first is presenting the famous Razborov-Smolensky Theorem, and the second is an introduction to arithmetic circuits.

4.1 Razborov-Smolensky Theorem

In this section we present the Razborov-Smolensky Theorem, which gives a lower bound related to $\mathbf{ACC^0}$ circuits. Our presentation is highly based on the presentation of Arora and Barak [2009]. The reason we present Razborov-Smolensky Theorem in these notes is the motivation it gives for the problem that was finally resolved by Williams, as discussed in Lecture 3. We, however, do not use Razborov-Smolensky for the proof of Williams' result.
We begin by recalling some definitions from Lecture 3. We recall some definitions related to $\mathbf{ACC^0}$ circuits (see also Section 3.2).

Definition 4.1. For any $m \in \mathbb{N}$, the MOD_m *gate* outputs 0 if the sum of its inputs is 0 modulo m, and 1 otherwise.

Definition 4.2. Let $m_1, \ldots, m_k \geq 2$ be integers. A language L is in $\mathbf{ACC^0}[m_1, \ldots, m_k]$, if there exists a circuit family computing it, such that:

- The circuits are of polynomial size.
- The circuits are of constant depth.
- The circuits are consisted (only) of the following gates: $\neg, \wedge, \vee$ and MOD_{m_1}, $\ldots$, MOD_{m_k}.

A language L is in $\mathbf{ACC^0}$ if it is in $\mathbf{ACC^0}[m_1, \ldots, m_k]$ for some $k \in \mathbb{N} \cup \{0\}$ and integers $m_1, \ldots, m_k \geq 2$. For convenience, we sometimes refer also to circuits as belonging to $\mathbf{ACC^0}$ (meaning, the circuits are restricted as above).

Note that $\mathbf{ACC^0}$ is just "$\mathbf{AC^0}$ with counters" and that the definitions are actually identical, except that $\mathbf{ACC^0}$ is allowed to have also "counters" (i.e. MOD gates). In particular, $\mathbf{AC^0}$ is completely identical to $\mathbf{ACC^0}[m_1, \ldots, m_k]$ for $k = 0$. Note also that for any $k \in \mathbb{N}$, $\mathbf{ACC^k}$ is defined analogously to $\mathbf{AC^k}$.

As stated before, in this section we present the Razborov-Smolensky Theorem. This is an impressive lower bound for $\mathbf{AC^0}$ circuits extended by any one type of a prime MOD gate (or alternatively, for $\mathbf{ACC^0}$ circuits restricted to have only one type of a prime MOD gate).

Theorem 4.3 (Razborov [1986]; Smolensky [1987]). *Let p, q be distinct (i.e., $p \neq q$) primes, such that $q \neq 2$. Then, it holds that*

$$\mathrm{MOD}_p \notin \mathbf{ACC^0}[q].$$

We give a proof for the special case of $p = 2$ and $q = 3$. That is, we show the MOD_2 function (i.e. the PARITY function) cannot be computed in $\mathbf{ACC^0}[3]$.[*] The proof can be generalized to give Theorem 4.3 (see also Section 2.1 in Viola [2009]). The proof is by a method known as *the method of approximating polynomials* that was originated by Razborov [1986] and then strengthened by Smolensky [1987].
It is a major open problem whether this lower bound may be extended (or generalized) to circuits that are allowed to have more than one type of MOD gate.

Proof of Theorem 4.3 for $p = 2$ and $q = 3$.
The proof is composed of 2 steps:

Step 1 We show that for every $\ell \in \mathbb{N}$ and any $\mathbf{ACC^0}[3]$ circuit C with n inputs, depth d and size s, there exists a polynomial $p \in \mathbb{F}_3[x_1, \dots, x_n]$ of degree at most $(2\ell)^d$ which agrees with the output of C on at least $1 - \frac{s}{2^\ell}$ fraction of the inputs. If we set $2\ell = c \cdot n^{1/2d}$, where c is a constant smaller than 1, we obtain a polynomial of degree at most $c^d\sqrt{n} \leq c\sqrt{n}$ that agrees with C on at least $1 - s/(2^{(1/2)\cdot c \cdot n^{1/2d}})$ fraction of the inputs.

Meaning, for any small enough $c < 1$ and circuit $C \in \mathbf{ACC^0}[3]$ with n inputs, depth d and size s, there exists a polynomial $p \in \mathbb{F}_3[x_1, \dots, x_n]$ such that:

- $\deg(p) \leq c \cdot \sqrt{n}$
- $\Pr_{x \in \{0,1\}^n}[\mathcal{C}(x) \neq p(x)] \leq s \cdot 2^{-(1/2)\cdot c \cdot n^{1/2d}}$

Step 2 We show that no polynomial in $\mathbb{F}_3[x_1, \dots, x_n]$ of degree at most $c \cdot \sqrt{n}$ agrees with MOD_2 (i.e. PARITY) on more than $\frac{49}{50}$ fraction of the inputs.

Meaning, for every polynomial $p : \mathbb{F}_3^n \to \mathbb{F}_3$ of degree at most $c \cdot \sqrt{n}$, it holds that

$$\Pr_{x \sim \{0,1\}^n}[p(x) \neq \mathrm{MOD}_2(x)] > \frac{1}{50}.$$

[*] Indeed, we defined $\mathbf{ACC^0}$ in terms of languages and MOD_2 is a function, not a language... However, it is straightforward to define a language for any Boolean function.

Together, the two steps imply that for any depth d circuit computing MOD_2, its size s must be exponential in $n^{\frac{1}{2d}}$, thus proving the theorem. We turn to the proof of both steps.

Step 1 Let $\ell \in \mathbb{N}$ and let C be an $\mathbf{ACC^0}[3]$ circuit with n inputs, depth d and size s. We show how to construct a polynomial in $F_3[x_1, \dots, x_n]$ that agrees with C on at least $1 - \frac{s}{2^\ell}$ fraction of the inputs in $\{0,1\}^n$.. We denote the elements of the field $\mathbb{F}_3$ by $\{-1, 1, 0\}$. Since we only care about agreement with C, whose output is Boolean, we only care about the output of the polynomial over the Boolean cube.

The proof is by induction on the depth d of the circuit. For $d = 0$, we simply have for an input x_i, the polynomial x_i. Correctness is straightforward. For $d = d' > 0$, we apply the theorem by induction for any $0 < d < d'$ and show how to extend it for a gate g at height d'. We separate the proof by the possible type of g:

$\neg$ gate: Let $g_{d'-1}$ be the gate that its output is the input of g. Then, by the induction hypothesis, we have an approximator $\tilde{g}_{d'-1} \in \mathbb{F}_3[x_1, \dots, x_n]$ for the original circuit whose output gate is $g_{d'-1}$. We may use it and have $\tilde{g} = 1 - \tilde{g}_{d'-1}$ as an approximator for g. Note that this introduces neither new error nor higher degree. By *no error* we mean that if on input x the original polynomial $\tilde{g}_{d'-1}(x)$ agreed with the output of the circuit without the $\neg$ gate, then $\tilde{g}(x)$ will necessarily agree with the output of the circuit with the $\neg$ gate.

MOD_3 gate: Let $g^1_{d'-1}, \dots, g^k_{d'-1}$ be the gates that their output is an input of g. Let $\tilde{g}^1_{d'-1}, \dots, \tilde{g}^k_{d'-1}$ be their respective appoximators. Since we have assumed we are dealing only with Boolean inputs, we may have the approximator $\left(\sum_{i=1}^k \tilde{g}^i_{d'-1}\right)^2$. This introduces no new error and it obeys the requirement of Theorem 4.3, with respect to the degree, since by the induction hypothesis, each approximator $\tilde{g}^i_{d'-1}$ has degree at most $(2\ell)^{(d'-1)}$ and then we have degree at most $2 \cdot (2\ell)^{d'-1} \leq (2\ell)^{d'}$, as desired.

$\vee$ gate: Let $g^1_{d'-1}, \dots, g^k_{d'-1}$ be the gates that their output is an input of g. Let $\tilde{g}^1_{d'-1}, \dots, \tilde{g}^k_{d'-1}$ be their respective approximators. Firstly, note that the naive approach, of using the polynomial $1 - \prod_{i=1}^k (1 - \tilde{g}^i_{d'-1})$ as an approximator, does not work, since the degree of the approximator may be (much) too large. We show another approach that does work. We randomly pick ℓ subsets of indices of the inputs of g: $S_1, \dots, S_\ell$. Each such subset contains each index with probability exactly $\frac{1}{2}$, independently.

We then compute for each subset S_i the polynomial $p_{S_i} = \left(\sum_{j \in S_i} \tilde{g}^j_{d'-1}\right)^2$. Finally, we compute the $\vee$ of these ℓ terms, using the naive approach (i.e., we composite these polynomials with the polynomial that naively computes $\vee$). It is straightforward to verify that the degree requirement is obeyed (since $\deg(p) \le \ell \cdot \max_{i \in [\ell]} \deg(p_{S_i}) \le \ell \cdot 2(2\ell)^{d-1} = (2\ell)^d$), but this solution, not surprisingly, introduces some new error. Additionally, we would like to have an explicit polynomial (a polynomial cannot flip coins...). We bound the error and then get rid of the randomness (that is, derandomize the construction).

Bounding the error: We first show that the probability of error for a single choice of indices S_i is bounded by $\frac{1}{2}$. Notice that the output of an $\vee$ gate is 1 if and only if at least one of the inputs is 1. Hence if the output of the $\vee$ gate is 0, it is straightforward to see that the approximator gives 0, as well, regardless of the random choices. For output 1, all we need to show is that with probability at least $\frac{1}{2}$ it holds that $\sum_{j \in S_i} \tilde{g}^j_{d'-1}$ is either -1 or 1. Let k_{last} be the index of the last 1 in the inputs of the $\vee$ gate. Consider the partial list of indices $1, \dots, k_{\text{last}} - 1$. We separate to cases (again):

$\boldsymbol{\bigvee_{i=1}^{k_{\text{last}}-1} g^i_{d'-1} = 0}$**:** If k_{last} is chosen to S_i, we have overall sum of 1, and thus we are fine with probability $\frac{1}{2}$, in this case.

$\boldsymbol{\bigvee_{i=1}^{k_{\text{last}}-1} g^i_{d'-1} = 1}$**:** We are fine in any case; if k_{last} is not chosen to S_i, the 1 remains; otherwise, we have -1, which is fine, as well.

$\boldsymbol{\bigvee_{i=1}^{k_{\text{last}}-1} g^i_{d'-1} = -1}$**:** We are fine if k_{last} is not chosen to S_i (analogously to the case of partial sum 0), and thus, again, we are fine with probability $\frac{1}{2}$.

Since the subsets $S_1, \dots, S_\ell$ are chosen independently, the error of the computation of the $\vee$ gate is at most $1/2^\ell$, as desired.

Getting rid of the randomness: We have shown thus far that *for any input* (for which the inputs to the gate are all correct) the probability of error, over the random choices of the subsets $S_1, \dots, S_\ell$ is bounded by $1/2^\ell$. If we show the statement with reversed quantifiers, i.e. that there *exist* subsets $S_1, \dots, S_\ell$ such that the probability of error *over the inputs* is bounded by $1/2^\ell$, we are done, since we may simply plug in these subsets into the construction. The latter is true, by just averaging over all inputs and choises of $S_1, \dots, S_\ell$ (this is a simple application of the *the probabilistic method*). Note, however, that using the last argument makes the proof non constructive. That is, it does not give us an efficient way to find these subsets $S_1, \dots, S_\ell$ in order to construct the polynomial that is proved to exist.

$\wedge$ **gate:** The approximator polynomial for $\wedge$ gates may be concluded by using the argument for $\vee$ gates together with De Morgan's laws.

Since there are s gates in the circuit C, and since our maximal possible probability of error, for any gate, is $1/2^\ell$, we have by the union bound that the overall probability of error is at most $s/2^\ell$.

This concludes the proof of Step 1. $\square$

Step 2 We now show the correlation proved to exist in Step 1 does not hold for MOD_2 (PARITY). Formally speaking, let $f \in \mathbb{F}_3[x_1, \dots, x_n]$ be a polynomial of degree bounded by $c \cdot \sqrt{n}$ and let $G' = \{x \in \{0,1\}^n : f(x) = \text{MOD}_2(x)\}$ be a set. We show that for an appropriate choice of c, $|G'| < \left(\frac{49}{50}\right) 2^n$. Firstly, we change our groundset to be $\{-1, 1\}$, i.e., we define for any input variable x_i, another, transformed, input variable $y_i = x_i + 1 (\text{mod } 3)$. That is, 0 becomes 1 and 1 becomes -1. Let $G \subseteq \{-1,1\}^n$ be the output of the transformation on the input variables $G' \subseteq \{0,1\}^n$, and let g be the transformed polynomial (i.e. the polynomial defined as f defined, but on the y_is). Then, $|G| = |G'|$ and the degree of g has not been changed (as we only conducted a linear transformation), and in particular has not exceeded $c \cdot \sqrt{n}$. This means, we may now show the desired claim using G and g. Let's see how does MOD_2 looks like with respect to the transformation:

$$\text{MOD}_2(x_1, \dots, x_n) = \begin{cases} 1 & \iff \prod_{i=1}^n y_i = -1 \\ 0 & \iff \prod_{i=1}^n y_i = 1 \end{cases}$$

Intuitively, all we should now show is that any degree $c \cdot \sqrt{n}$ polynomial cannot approximate well the polynomial above, which is of degree n (this seems very reasonable...). Formally speaking, let F_G be the set of all possible functions $f_G : G \to \{0, 1, -1\}$. We show that $|F_G| \leq 3^{\left(\frac{49}{50}\right)2^n}$ which concludes Step 2, since (of course) $|F_G| = 3^{|G|}$. For this, we do another (last![*]) transformation. We show that for any function $f_G \in F_G$ there exists another polynomial $g_G : \mathbb{F}_3^n \to \mathbb{F}_3$ with monomials of degree bounded by $\frac{n}{2} + c \cdot \sqrt{n}$, totally agreeing with it on G. Then, finally, we just bound the possible number of the latter polynomials, concluding the desired. Let $f_G \in F_G$.

[*]but composed of (simple) sub-transformations...

Transforming to g_G: Of course, there exists some (non further restricted) polynomial $g'_G : \mathbb{F}_3^n \to \mathbb{F}_3$ (totally) agreeing with f_G on G. Since $G \subseteq \{-1,1\}^n$, for any input y_i, we have $y_i^2 = 1$, and therefore, any exponent may be reduced by any even number. In particular, this means g'_G may be transformed to another polynomial which is multi-linear (i.e. without any exponent greater than 1). To finish the transformation, we just need to show the existence of a polynomial that does not have large degree monomials, that still agree with f_G on G. Let $\prod_{i \in I} y_i$ be one of the monomials with $|I| > \frac{n}{2}$. Denote $\bar{I} = [n] \setminus I$. We transform this monomial to:

$$\prod_{i=1}^{n} y_i \prod_{i \in \bar{I}} y_i = g(y_1, \ldots, y_n) \cdot \prod_{i \in \bar{I}} y_i \ .^{*}$$

which has degree at most $\frac{n}{2} + c \cdot \sqrt{n}$ (note that this is no longer a monomial, but rather a polynomial). This is correct since by our assumption for Step 2, g has degree at most $c \cdot \sqrt{n}$, and since $\prod_{i \in \bar{I}} y_i$ has degree strictly less than $\frac{n}{2}$ (since $|I| > \frac{n}{2}$, by our assumption for this specific monomial). This gives the desired properties of the transformation.

It is left to bound the possible number of polynomials with degrees of all monomials bounded by $\frac{n}{2} + c \cdot \sqrt{n}$. It is straightforward that this number is bounded by:

$$3^{(\text{number of monomials with degree at most } \frac{n}{2} + c \cdot \sqrt{n})}$$

(since any monomial may appear with coefficient 0 (i.e., to not appear), 1 or -1). The number of monomials with degree at most $\frac{n}{2} + c \cdot \sqrt{n}$ is bounded by:

$$\sum_{d=0}^{\frac{n}{2} + c \cdot \sqrt{n}} \binom{n}{d}$$

which is upper bounded by $\left(\frac{49}{50}\right) 2^n$, for some constant c by bounds on the tails of the binomial distribution. This concludes the proof of Step 2. □

The proof of Theorem 4.3 is concluded from Steps 1 and 2. □

4.2 Introduction to Arithmetic Circuits

In this section we introduce arithmetic circuits. While a Boolean circuit computes a Boolean function, an arithmetic circuit computes a polynomial (think of the deter-

[*] Note that this equality only holds over the Boolean cube.

minant of a matrix for example). Studying arithmetic circuits is natural on its own, but our interest in them is their application for our main goal - proving William's result. For more information, we refer the reader to a recent survey by Shpilka and Yehudayoff [2010].
We first define the basic notions, following Arora and Barak [2009]. In general, an arithmetic circuit computes a polynomial over a field $\mathbb{F}$. The following definition is almost exactly as the definition of Boolean circuits in Section 2.2:

Definition 4.4. An n-input *arithmetic circuit* is a directed acyclic graph with n *sources* and one *sink*. All non-source vertices are called *gates* and are labeled with one of $\{+, \times\}$. In this lecture we consider gates with fan-in 2.

Definition 4.5. A polynomial P over a field $\mathbb{F}$ is the *(identically) zero polynomial* if all its coefficients are 0.

4.2.1 The Determinant and the Permanent

In this section we introduce the Determinant and Permanent polynomials.

Definition 4.6. The *determinant* of an $n \times n$ matrix $X = (X_{ij})$ is defined as

$$\mathsf{Det}(X) \overset{\text{def}}{=} \sum_{\sigma \in S_n} (-1)^{\mathsf{sign}(\sigma)} \prod_{i=1}^{n} x_{i\sigma(i)},$$

where S_n is the group of all $n!$ permutations on $\{1, 2, \ldots, n\}$.

The determinant of an $n \times n$ matrix $X = (X_{ij})$ can be computed using the familiar Gaussian elimination algorithm. This algorithm uses at most $O(n^3)$ addition and multiplication operations and thus one obtains an *arithmetic circuit* of size $O(n^3)$.
The (famous) determinant polynomial is a nice example for the fact that a polynomial may generally have exponentially many monomials (in this case $n!$), but nevertheless, be computable by a family of polynomial-size circuits.
The determinant polynomial is a complete problem for the class **VP** (also known as $\mathbf{AlgP}_{/\mathbf{poly}}$), defined as follows:

Definition 4.7 (**VP**, Informal)**.** **VP** is the class of polynomials f of polynomial degree that have polynomial size arithmetic circuits computing them.

For a formal definition, see for example Arora and Barak [2009].

Definition 4.8. The *permanent* of an $n \times n$ matrix $X = (X_{ij})$ is defined as

$$\mathsf{Perm}(X) \overset{\text{def}}{=} \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i\sigma(i)},$$

where S_n is the set of all $n!$ permutations on $\{1, 2, \ldots, n\}$.

The permanent polynomial that, at first sight, seems to be very similar to the determinant, is conjectured to be much harder to compute. In particular, it is conjectured that there is no family of circuits of polynomial size that computes it. The permanent polynomial is a complete problem for the class **VNP** (also known as $\mathbf{AlgNP}_{/\mathbf{poly}}$) Valiant [1979b]:

Definition 4.9 (**VNP**, Informal)**.** **VNP** is a complexity class of polynomials such that the coefficient of any given monomial can be computed efficiently (i.e. by a polynomial size arithmetic circuit).

For a formal definition, see for example Arora and Barak [2009].

4.2.2 Bipartite Matching and the Determinant

This section is partially based on lecture notes of Rubinfeld [2006]. Let's consider the following motivating decision problem.

Problem 4.10. Given a bipartite graph $G = (V \cup U, E)$, does a perfect matching exist?

We show an algorithm for Problem 4.10 that does not rely on network flows, but is based on algebraic techniques. Let $G = (V \cup U, E)$ be a bipartite graph. We construct the matrix $A_G = [a_{ij}]$, known as *Edmond's matrix* (see e.g., Motwani and Raghavan [1995]), where a_{ij} gets some free variable X_{ij} if $(i, j) \in E$, and $a_{ij} = 0$ otherwise. We prove the following theorem.

Theorem 4.11. *Given a bipartite graph $G = (V \cup U, E)$, G has a perfect matching if and only if $\mathsf{Det}(A_G) \neq 0$.*

Proof. Recall that

$$\mathsf{Det}(A_G) = \sum_{\sigma \in S_n} (-1)^{\mathsf{sign}(\sigma)} \prod_{i=1}^{n} x_{i\sigma(i)}.$$

Observe that each permutation σ corresponds to a possible perfect matching, and vice-versa, in the natural way, namely i is matched to $\sigma(i)$. The product $\prod_{i=1}^{n} x_{i\sigma(i)}$ will be non-zero if and only if $\forall_{i \in [n]} : (v_i, v_{\sigma(i)}) \in E$. In this case, σ corresponds to a perfect matching in G. The polynomial $\mathsf{Det}(A_G)$ is non-zero if and only if any term in the determinant is non-zero (notice that there are no cancellations since every term corresponds to a different monomial that differs in at least one variable). $\square$

In other words, in order to check whether there is a perfect matching in G, all we need to do is to check if $\mathsf{Det}(A_G)$ is the (identically) zero polynomial. This problem can be reduced to a famous problem known as Polynomial Identity Testing (henceforth PIT):

Problem 4.12 (PIT). Given two polynomials P and Q, is it true that $P \equiv Q$?

A straightforward way to solve this problem is just by expanding P and Q and comparing their coefficients (one by one). However, these expansions may result in exponentially many terms (in the number of variables) which, in turn, results in an inefficient algorithm.
We show an alternative algorithm solving Problem 4.12 in polynomial-time, but using randomness. The algorithm we present actually checks whether a polynomial is the zero polynomial (this is known as the ZEROP problem). Since for any polynomials P and Q, $P \equiv Q$ if and only if $P - Q \equiv 0$, this algorithm solves (also) Problem 4.12. This algorithm may work for any representation of the polynomial as long as there is an efficient algorithm for evaluating the polynomial at any input point x. That is, since there exists a family of polynomial-size circuits computing the determinant of a given matrix (or, even better, an efficient algorithm for doing so), we may use the algorithm we show below, in order to solve Problem 4.10, as well.

Algorithm 4.13. For an input polynomial (over $\mathbb{Z}$)[*] of degree d, sample integer random variables in the range $[10d]$, and then evaluate the polynomial on these points. Return TRUE if and only if all the evaluations result with 0 value.[†]

The analysis of Algorithm 4.13 follows, straightforwardly, from a lemma known as the Scwartz-Zippel Lemma:

Lemma 4.14 (Zippel [1979]; Schwartz [1980]). *Let $p \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree d. Then, it holds that*

$$\forall S \subseteq \mathbb{Z} : \Pr_{r_1,\ldots,r_n \in_R S}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

[*]Arithmetic circuits are interesting over any field, and also over the ring of integers $\mathbb{Z}$. In addition, we note that a similar technique works for finite fields.

[†]A circuit of size s might compute a polynomial of degree 2^s and thus the numbers that can be computed along the way could get as high as $(2^s)^{2^s}$, and so just representing this numbers will require exponential number of bits in the input length. One solution to this problem is to do all the computation modulo a randomly chosen prime. Details are omitted.

Note that it is not known whether Algorithm 4.13 may be derandomized. Moreover, it is not known at all whether PIT may be solved deterministically in polynomial-time; in fact it is a major open problem:

Open Problem 4.15. Does there exist a *deterministic* polynomial-time algorithm solving PIT?

In Theorem 7.1 we show that if PIT can be derandomized, some kind of circuit lower bounds follow. Optimistic would consider such result as a motivation for derandomizing PIT. The more pessimistic of us might consider this result as a barrier for derandomizing PIT.

LECTURE 5
RANDOMIZATION IN COMPUTATION

NOVEMBER 29TH, 2012

LECTURER: Gil Cohen SCRIBE: Inbal Livni, Shani Nitzan

5.1 Complexity Classes for Randomized Computation

A priori, it is not clear that there is any relation between randomized computation and circuit lower bounds. But in fact, they are as related as they could be. The bottom line is that circuit lower bounds will imply derandomization of any efficient algorithm, and vice versa! Even without a formal treatment, one can appreciate such a deep discovery, which I personally consider to be one of the most joyful pearls in complexity theory (yes yes, PCP is cooler..)
To the light of that, it is clear that a formal treatment of randomness in computation is necessary in a self contained document concerning circuit lower bounds. This treatment is what will occupy us in this lecture. We define probabilistic Turing machines and complexity classes that capture randomized computation. We then discuss the relations of these complexity classes to familiar classes. We end this lecture with a derandomization of one specific and simple randomized algorithm.

Definition 5.1. A *Probabilistic Turing Machine* is a Turing machine that has an additional state, q_{sample}. If the machine is in state q_{sample} the next state will be either q_0 or q_1, with probability $\frac{1}{2}$ for each state.

An equivalent definition is a Turing machine that has an additional tape, with random bits on it. This tape is read only and the machine can only go forward on it.
We now define complexity classes that capture randomized computation. The first is analog to $\mathbf{DTIME}(\cdot)$ and the second to $\mathbf{P}$. The initials **BP** in both definitions stands for "Bounded Probabilistic", as the probability of error is bounded.

Definition 5.2. Let $T : \mathbb{N} \to \mathbb{N}$. $L \in \mathbf{BPTIME}(T(n))$ if there exists a probabilistic Turing machine M that halts within $O(T(n))$ steps on input of size n, such that

- $\forall x \in L,\ \Pr[M(x) = 1] \geq \frac{2}{3}$
- $\forall x \notin L,\ \Pr[M(x) = 1] \leq \frac{1}{3}$,

where the probability is over the random bits of the Turing machine, and not over the inputs.

Definition 5.3.

$$\mathbf{BPP} = \bigcup_{c \geq 1} \mathbf{BPTIME}\left(n^c\right)$$

Notice that $\mathbf{P} \subseteq \mathbf{BPP}$ as $\mathbf{DTIME}\left(f(n)\right) \subseteq \mathbf{BPTIME}\left(f(n)\right)$ for every f. The class $\mathbf{BPP}$ arguably models efficient computation better than $\mathbf{P}$, as (arguably) random bits can be efficiently obtained in the real world (unlike, say, non-determinism). This is supported by the tendency of researchers in algorithms to settle for randomized algorithms for all practical purposes.
It is not known whether $\mathbf{P}$ is *strictly* contained in $\mathbf{BPP}$. An immediate upper bound on efficient randomized computation is given by $\mathbf{BPP} \subseteq \mathbf{PSPACE}$. To see this, notice that when fixing the random string in a $\mathbf{BPP}$ algorithm, the randomized algorithm is in fact a polynomial-time deterministic algorithm. One can reuse the same space in order to simulate this deterministic algorithm for every possible random string, and count for how many random strings the computation ends in the accepting state. Since the length of the random strings is polynomial in the input length (as it is bounded by the running time), the counter will count only to something exponential in the input length, which requires only polynomial space.

5.2 Efficient Randomized Computation vs. Non-Determinism

Unfortunately, the tradeoff between efficient randomized computation and efficient verification (that is, $\mathbf{NP}$ computation) is much less understood. That is, it is not known what is the relation between $\mathbf{BPP}$ and $\mathbf{NP}$. Nevertheless, going up one level in the Polynomial Hierarchy, we have the following result.

Theorem 5.4 (Sipser [1983])**.**

$$\mathbf{BPP} \subseteq \mathbf{\Sigma_2^p} \cap \mathbf{\Pi_2^p}$$

In fact, Sipser "only" proved that $\mathbf{BPP} \subseteq \mathbf{PH}$. The (potentially) stronger result stated above is due to Gacs. It is worth noting that Theorem 5.4 has found many alternative proofs, e.g., Nisan and Wigderson [1994]; Goldreich and Zuckerman [1997]. We will follow a proof by Lautemann [1983].

Although we won't use Theorem 5.4 in these notes, we find its proof to be quite insightful for randomization as well as for the expressive power of alternating quantifiers.

Proof. [*] Since **BPP** is closed under complement, it is enough to show $\mathbf{BPP} \subseteq \mathbf{\Sigma_2^p}$. Suppose $L \in \mathbf{BPP}$. Then using repetition (see Lemma 5.7), there exists a polynomial-time randomized Turing machine M such that

$$x \in L \Rightarrow \Pr_r[M(x,r) \text{ accept}] > 1 - 2^{-2n}$$
$$x \notin L \Rightarrow \Pr_r[M(x,r) \text{ accept}] < 2^{-2n}$$

Let $m = \mathrm{poly}(n)$ be the number of random bits M uses for inputs of length n. Fix $x \in \{0,1\}^n$ and consider the set $S_x \subseteq \{0,1\}^m$ of all random strings for which $x \in \{0,1\}^n$ is accepted by M. That is,

$$r \in S_x \iff M(x,r) = 1.$$

We have a dichotomy: if $x \notin L$ then $|S_x| \leq 2^{m-2n}$, while if $x \in L$ then $|S_x| \geq 2^m \cdot (1 - 2^{-2n})$. For a vector $u \in \mathbb{F}_2^m$ and a set $S \subseteq \mathbb{F}_2^m$ define the *shift* of S by u as

$$u + S = \{u + s : s \in S\},$$

where addition is vector addition over $\mathbb{F}_2^m$ (i.e. bitwise XOR). We claim the following lemma.

Lemma 5.5. *For $k = \lceil m/n \rceil$, $x \in L$ iff there exist $u_1, \ldots, u_k \in \mathbb{F}_2^m$ such that $\bigcup_{i=1}^k (u_i + S_x) = \mathbb{F}_2^m$.*

Proof. For $x \notin L$, we know that S_x is small, and thus for any k vectors $u_1, \ldots, u_k \in \mathbb{F}_2^m$:

$$\left| \bigcup_{i=1}^{k} u_i + S_x \right| \leq k \cdot |S_x| \leq k \cdot 2^{m-2n} < 2^m$$

for n large enough.
For $x \in L$, we will use the probabilistic method to show the existence of vectors $u_1, \ldots, u_k \in \mathbb{F}_2^m$ such that $\bigcup_{i=1}^k u_i + S_x = \mathbb{F}_2^m$. Choose uniformly at random $u_1, \ldots, u_k \in \mathbb{F}_2^m$. What is the probability that a certain element $y \in \mathbb{F}_2^m$ is not covered by $\bigcup_{i=1}^k u_i + S_x$? The probability that y is not covered by one shift, $y \notin u_i + S_x$, is exactly the probability that $y + u_i \notin S_x$. Since $y + u_i$ is a uniformly random

[*]The proof of this theorem was scribed by Avishay Tal.

vector in $\mathbb{F}_2^m$, this probability is exactly $1 - |S_x|/2^m$. Since $u_1, u_2, \dots, u_k$ are chosen independently:

$$\Pr[y \text{ is not covered}] = (1 - |S_x|/2^m)^k \leq (2^{-2n})^{m/n} = 2^{-2m} \ .$$

By union bound

$$\Pr[\exists y \text{ which is not covered}] \leq 2^m \cdot 2^{-2m} \leq 2^{-m} \ .$$

Thus, most choices of $u_1, \dots, u_k$ give $\bigcup_{i=1}^k u_i + S_x = \mathbb{F}_2^m$. □

We return to the proof of the claim. Using Lemma 5.5 we know

$$x \in L \Leftrightarrow \exists u_1, \dots, u_k \in \{0,1\}^m \ \bigcup_{i=1}^k u_i + S_x = \mathbb{F}_2^m \ .$$

We can write this equivalently as

$$x \in L \Leftrightarrow \exists u_1, \dots, u_k \in \{0,1\}^m \quad \forall y \in \{0,1\}^m \ \bigvee_{i=1}^k M(x, y + u_i) \ .$$

As the inner expression $\bigvee_{i=1}^k M(x, y+u_i)$ can be computed by a polynomial-time Turing machine, and all strings in the quantifiers are of length $\mathrm{poly}(n)$ this shows that $L \in \mathbf{\Sigma_2^P}$. □

The lack of understanding regarding the tradeoff between randomization and non-determinism is also given by the following major open problem:

Open Problem 5.6. Prove that **BPP** is strictly contained in **NEXP**.

5.3 Efficient Randomized Computation vs. Non-Uniformity

We now turn to discuss the tradeoff between randomization and non-uniformity. For that we will need the following lemma, which shows that the constants $\frac{1}{3}$ and $\frac{2}{3}$ in the definition of **BPP** are arbitrary. In fact, even when replacing them with an exponentially vanishing quantities, the class **BPP** remains as is.

Lemma 5.7. *For any $L \in$ **BPP** and any constant $c > 0$, there exists a probabilistic Turing machine that on input x runs in polynomial-time in $|x|$, and fails with probability less than $2^{-c \cdot |x|}$.*

Proof. Let $L \in \mathbf{BPP}$. There exists a probabilistic Turing machine M, that runs in polynomial-time, such that

$$\forall x \in L, \Pr[M(x) = 1] \geq \frac{2}{3},$$

and

$$\forall x \notin L, \Pr[M(x) = 1] \leq \frac{1}{3}.$$

The proof idea is as follows. Given M, x one can think of $M(x)$ as a random variable that can be sampled efficiently (as the running time of M on x is polynomial in $|x|$). If M accepts x this random variable has high expectation (at least $\frac{2}{3}$), whereas if M rejects x, the random variable $M(x)$ has low expectation (at most $\frac{1}{3}$). We now show that a polynomial number of samples of $M(x)$ is enough to approximate the expectation of $M(x)$ to within, say 1/10, with probability $2^{-c\cdot|x|}$. This concludes the proof as $\frac{1}{3} + \frac{1}{10} < \frac{2}{3} - \frac{1}{10}$.
We now make it formal. For a number m to be determined later on, we construct a probabilistic Turing machine M' that on input x, will estimate $\mathbb{E}[M(x)] = \Pr[M(x) = 1]$. M' will simulate M on x for m times independently, and calculate the ratio of acceptance, denoted by

$$A = \frac{\text{number of runs in which } M \text{ accepts}}{m}.$$

M' will accept if and only if $A \geq \frac{2}{3} - \frac{1}{10}$. To calculate M' probability of mistake, denote by A_i the random variable $M(x)$ on the i'th run. Using this notation,

$$A = \frac{1}{m} \cdot \sum_{i=1}^{m} A_i.$$

By linearity of expectation, we have that $\mathbb{E}[A|x \in L] \geq \frac{2}{3}$, and $\mathbb{E}[A|x \notin L] \leq \frac{1}{3}$. The expectation of $\mathbb{E}[A|x \in L]$ and $\mathbb{E}[A|x \notin L]$ are far away, which makes it possible to distinguish between the two cases with high probability. M' is mistaken only when A differs from its expectation by at least $m/10$. Since the simulations are independent, we can bound the probability for error by Chernoff's inequality.

$$\begin{aligned}
\Pr\left[M'(x)=0|x\in L\right] &= \Pr\left[A<\frac{2}{3}-\frac{1}{10}\middle| x\in L\right]\\
&\le \Pr\left[A-\mathbb{E}[A]\le -\frac{1}{10}\right]\\
&= \Pr\left[\frac{1}{m}\cdot\sum_{i=1}^{m}A_i-\mathbb{E}\left[\frac{1}{m}\cdot\sum_{i=1}^{m}A_i\right]\le -\frac{1}{10}\right]\\
&= \Pr\left[\sum_{i=1}^{m}A_i-\mathbb{E}\left[\sum_{i=1}^{m}A_i\right]\le -\frac{1}{10}m\right]\\
&\le e^{-2\left(\frac{1}{10}m\right)^2\frac{1}{m}}\\
&= e^{-\frac{1}{50}m},
\end{aligned}$$

and similarly

$$\begin{aligned}
\Pr\left[M'(x)=1|x\notin L\right] &= \Pr\left[A\ge\frac{2}{3}-\frac{1}{10}\middle| x\notin L\right]\\
&= \Pr\left[A-\mathbb{E}[A]\ge \frac{7}{30}\right]\\
&= \Pr\left[\frac{1}{m}\cdot\sum_{i=1}^{m}A_i-\mathbb{E}\left[\frac{1}{m}\cdot\sum_{i=1}^{m}A_i\right]\ge \frac{7}{30}\right]\\
&= \Pr\left[\sum_{i=1}^{m}A_i-\mathbb{E}\left[\sum_{i=1}^{m}A_i\right]\ge \frac{7}{30}m\right]\\
&\le e^{-2\left(\frac{7}{30}m\right)^2\frac{1}{m}}\\
&\le e^{-\frac{1}{50}m}.
\end{aligned}$$

We conclude the proof by choosing the minimal m such that $e^{-\frac{1}{50}m}\le 2^{-c\cdot n}$. Note that $m=O(n)$, which concludes the proof. □

We can now show that non-uniformity is, in some sense, stronger than randomization.

Theorem 5.8 (Adleman [1978]). **BPP** $\subset$ **P/poly**.

Before proving the theorem, it is worth noting that it implies that Open Problem 5.6 is easier (or not less hard) than showing that **NEXP** $\not\subset$ **P/poly**.

Proof. Take $L\in$ **BPP**. By Lemma 5.7 there exists a polynomial-time probabilistic Turing machine M, that has a probability of error less than $2^{-(n+1)}$. Let t_n, which is polynomial in n, be the maximal number of random bits that M uses, running on inputs of length n. Denote by $M_r(x)$ the result of M on x, using the string r as the random string. As before, M_r is a deterministic Turing machine. Since the probability of error is less than $2^{-(n+1)}$, for any x of length n,

$$\left|\left\{r \in \{0,1\}^{t_n} | M_r(x) \text{ is wrong } \right\}\right| \leq 2^{-(n+1)} \cdot 2^{t_n}.$$

Taking the union bound of these sets for all $x \in \{0,1\}^n$

$$\left|\left\{r \in \{0,1\}^{t_n} | \exists x \in \{0,1\}^n, \text{such that } M_r(x) \text{ is wrong } \right\}\right| \leq 2^n \cdot 2^{-(n+1)} \cdot 2^{t_n} = 2^{t_n - 1}.$$

Meaning that there are at least 2^{t_n-1} random strings that when M uses them on any input of length n the result is correct. For any n, let r_n be such string. For every n, the deterministic Turing machine M_{r_n} is correct for all inputs of length n, and runs in polynomial-time. From the proof of $\mathbf{P} \subseteq \mathbf{P/poly}$ (see Theorem 2.12), for any polynomial-time Turing machine there exists a polynomial size circuit C_n such that $C_n(x) = M_{r_n}(x)$. This concludes the proof, as the family $\{C_n\}_n$ decides L. $\square$

5.4 P vs. BPP

We postponed the perhaps most natural question of all - does randomization contributes to efficient computation? In other words, is **P** strictly contained in **BPP**? There are examples of computational problems that we do know how to solve efficiently with randomness and do not know how to efficiently solve without. One prominent example would be the PIT problem, introduced in Lecture 4. At the early days, the answer to this question wasn't clear, and people might had the tendency to think that the containment is strict.

Nowadays, it is believed by many that **P** = **BPP**, and it will be hard to find a complexity theorist that doesn't believe that any randomized algorithm can be simulated by a deterministic subexponential time algorithm. Thus, randomization is, in some sense, an algorithmic design tool - it doesn't believed to add to computational power, but it is certainly a helpful way of thinking of solutions to computational problems. Moreover, even in cases where we do have deterministic algorithms, it is quite common that randomized algorithms are simpler and faster.

How would one prove that **P** = **BPP**? or, even simply improve upon the trivial derandomization obtained by enumerating all possible random strings, and taking the majority vote? The key idea is to "convert" an efficient randomized algorithm to a deterministic one by exploiting the fact that the algorithm is efficient, and thus, may not have enough time to "appreciate" the purely random string we feed to it. Thus, the

conversion is done by replacing the completely random string with a "pseudorandom string" - a string that looks random to an efficient observer. The hope is that the sample space of pseudorandom strings has small support, and thus, one can simulate the algorithm on all pseudorandom strings, and answer according to the majority.
This procedure, which is a natural method for *derandomization*, might seem a bit abstract, thus we find it helpful to start with an example. In the next section we derandomize one simple efficient algorithm. After all, if you want to derandomize all efficient algorithms, it is wise to start with a simple one. As it turns out, derandomizing this simple algorithm is extremely fruitful, and we will use it in the proof of Williams Theorem (see Lecture 10).

5.5 Small-Bias Sets

The algorithm below is an example of a probabilistic algorithm for which we do know how to efficiently construct a sample space with support that has a polynomial-size (as appose to exponential size). The algorithm, given $x \in \{0,1\}^n$ as input, returns a parity of a random subset of x's bits.
Random Parity Algorithm: On input $x \in \{0,1\}^n$

1. Sample n random bits, $y \sim \{0,1\}^n$.
2. Return $< x, y >= \sum_{i=1}^{n} x_i y_i \pmod 2$.

Although this algorithm might look like an odd "toy example", its pseudorandom sample space is extremely important and central in theoretical computer science, with an overwhelming range of applications!
Denote the algorithm by A. The returned value by this algorithm, for every $x \neq 0$, satisfies $\Pr[A(x) = 1] = \frac{1}{2}$, because y is chosen uniformly at random. In other words, the bias of $A(x)$, which is defined as $|\mathbb{E}_y \left[(-1)^{A(x)}\right]|$ is 0 for every input $x \neq 0$.
One can prove (this is not hard) that the only way to get a zero bias for any non-zero input will require sampling a completely uniform string $y \sim \{0,1\}^n$. Therefore, our goal is to find an algorithm, that samples y from a polynomial-size set $S \subset \{0,1\}^n$, and gives a bias close to 0 for all $x \neq 0$.

Definition 5.9 (Naor and Naor [1990]). A set [*] $S \subset \{0,1\}^n$ is called *ε-biased* if $\forall x \in \{0,1\}^n \setminus \{0\}$ it holds that

$$|\mathbb{E}_{y \sim S}\left[(-1)^{<x,y>}\right]| \leq \varepsilon.$$

[*]In fact, S might be a multi-set as it is the support of some sample space. Still, the term "Small-Bias Set" is somewhat more common than "Small-Bias Sample Space", and se we stick to it.

How small can we expect an ε-biased set to be, as a function of n, ε? This question might be a bit odd at first look, after all, how can we hope to know something about the size of ε-biased sets without actually finding one? In other words, is it easier to show that an ε-biased set of such and such size exists without finding one? The answer to this question is yes, and this is the case in many situations where one wants to mimic random behavior. The idea is to show that a sufficiently, yet not too large random set $S \subseteq \{0,1\}^n$ is, with positive probability, an ε-biased set. This gives us no clue on how to find such a set efficiently yet assures us of its existence. We actually used this method, known us the *probabilistic method*, several times before (e.g., in the proof of Theorem 4.3 and Theorem 5.4).

Lemma 5.10. *For every $n \in \mathbb{N}$ and $\varepsilon > 0$ there exists an ε-biased set $S \subseteq \{0,1\}^n$ of size $S = O(n/\varepsilon^2)$.*

Proof. Let S be the multiset $\{y_1, \dots, y_k\}$, where each y_i is sampled uniformly from $\{0,1\}^n$ independent of all other y_j's (so it's possible to have repetitions). For input $x \in \{0,1\}^n$, define the random variables $I_{j,x} =< x, y_j >$ and $U_x = \sum_{j=1}^{k} I_{j,x}$ (where the sum is not modulo 2). Note that

$$|\mathbb{E}_{y\sim S}\left[(-1)^{A(x)}\right]| < \varepsilon \iff U_x \in \left[(1-\varepsilon)\cdot\frac{k}{2}, (1+\varepsilon)\cdot\frac{k}{2}\right].$$

By linearity of expectation,

$$\mathbb{E}[U_x] = \sum_{j=1}^{k} \mathbb{E}[I_{j,x}] = \sum_{j=1}^{k} \frac{1}{2} = \frac{k}{2}.$$

Therefore, by Chernoff bound,

$$\Pr\left[|U_x - \mathbb{E}[U_x]| \geq \frac{k}{2}\cdot\varepsilon\right] \leq 2\cdot e^{-2\cdot\frac{\left(\frac{k}{2}\cdot\varepsilon\right)^2}{k}} = 2\cdot e^{-\frac{k\varepsilon^2}{2}}.$$

By taking the union bound over all $x \in \{0,1\}^n \setminus \{0\}$,

$$\Pr\left[\exists x, |U_x - \mathbb{E}[U_x]| \geq \frac{k}{2}\cdot\varepsilon\right] \leq 2^n \cdot 2 \cdot e^{-\frac{k\varepsilon^2}{2}}.$$

For $k \geq \frac{2n}{\varepsilon^2}$, the expression $2^n \cdot 2 \cdot e^{-\frac{k\varepsilon^2}{2}} < 1$. This means that the probability of this "bad event" (of a random set not to be an ε-biased set) is strictly smaller than 1 (and in fact, exponentially small), and therefore there exists an S of size $O(n/\varepsilon^2)$ which is ε-biased. □

In their seminal paper, Naor and Naor [1990] not only defined small-bias sets, but also gave an explicit construction of such sets with size $O(n/\varepsilon^c)$, for some constant $c \geq 3$. By "explicit construction" of an ε-biased set we mean that there exists an efficient algorithm that given n, ε as inputs, outputs an ε-bias set in $\{0,1\}^n$. We present an alternative construction (incomparable in terms of size) called *the Powering Construction*.

Theorem 5.11 (Alon et al. [1992]). *There exists an explicit construction of an ε-biased set S such that $|S| = O\left(\left(\frac{n}{\varepsilon}\right)^2\right)$.*

Proof. This construction uses the fact that $\mathbb{F}_{2^m}$ is isomorphic to $\mathbb{F}_2^m$, where $\mathbb{F}_{2^m}$ is the field with 2^m elements, and $\mathbb{F}_2^m$ is the m dimensional vector space over $\mathbb{F}_2$, the field of 2 elements. We define

$$S = \left\{ S_{a,b} | a, b \in \mathbb{F}_{2^m}, \left(S_{a,b}\right)_i =< a^i, b > \right\}.$$

Meaning, every element in S is indexed by two field elements, $a, b \in \mathbb{F}_{2^m}$, and $|S| = 2^m \cdot 2^m = 2^{2m}$. We need to find an $m = m(n, \varepsilon)$ such that S will be an ε-biased set. For every $x \neq 0$, we want to find an upper bound on $\left|\mathbb{E}_{a,b \in \mathbb{F}_{2^m}} (-1)^{<x, S_{a,b}>}\right|$. To do this, we bound the difference between the number of $S_{a,b}$ such that $< x, S_{a,b} >= 1$, and those that give $< x, S_{a,b} >= 0$. Notice, that when calculating a^i, a is treated as an element in $\mathbb{F}_{2^m}$ (hence the name "The Powering Construction"), and when calculating the inner product, a^i and b are both treated as vectors in $\mathbb{F}_2^m$.
By the linearity of inner product,

$$< x, S_{a,b} >= \sum_{i=1}^{n} x_i \left(S_{a,b}\right)_i = \sum_{i=1}^{n} x_i < a^i, b >= \left\langle \sum_{i=1}^{n} x_i a^i, b \right\rangle.$$

For every $x \in \{0,1\}^n$, define the polynomial $p_x(y) = \sum_{i=1}^{n} x_i \cdot y^i$ (x_i are the coefficients). For a fixed a which is a root of p_x, $p_x(a) = 0$, and for every $b \in \mathbb{F}_{2^m}$,

$$< x, S_{a,b} >= \left\langle \sum_{i=1}^{n} x_i a^i, b \right\rangle = \langle p_x(a), b \rangle = \langle 0, b \rangle = 0,$$

and thus, for such a,

$$\left| \left\{ b \in \mathbb{F}_{2^m} \middle| \left\langle \sum_{i=1}^{n} x_i a^i, b \right\rangle = 0 \right\} \right| = 2^m.$$

For a fixed a which is not a root of p_x, $\sum_{i=1}^{n} x_i a^i = p_x(a)$ is a non-zero field element and therefore a non-zero vector, and so

$$\left|\left\{b \in \mathbb{F}_{2^m} \,\middle|\, \left\langle \sum_{i=1}^{n} x_i a^i, b \right\rangle = 0 \right\}\right| = \left|\left\{b \in \mathbb{F}_{2^m} \,\middle|\, \left\langle \sum_{i=1}^{n} x_i a^i, b \right\rangle = 1 \right\}\right| = 2^{m-1}.$$

For such a, the number of $S_{a,b}$'s such that $< x, S_{a,b} >= 1$ equals to the number of $S_{a,b}$'s such that $< x, S_{a,b} >= 0$. This means that the only difference will be from an a which is a root of p_x. p_x is a polynomial of degree at most n, so it has at most n roots. Hence,

$$\left|\mathbb{E}_{a,b \in \mathbb{F}_{2^m}}\left[(-1)^{\langle x, S_{a,b}\rangle}\right]\right| \leq \frac{2^m \cdot n}{2^{2m}} = \frac{n}{2^m}.$$

To find the right $m(n, \varepsilon)$, m needs to satisfy $\frac{n}{2^m} \leq \varepsilon$, meaning $m \geq \log \frac{n}{\varepsilon}$. For this m, $|S| = 2^{2m} = O\left(\left(\frac{n}{\varepsilon}\right)^2\right)$. It can be shown that the set S can be constructed efficiently, that is, in time poly(n/ε). $\square$

The above theorem states that there are near-optimal small-bias sets that can be computed in **P**. It is interesting to add that in fact, near-optimal small-bias sets can even be computed in uniformly generated (see Section 3.3) $\mathbf{ACC^0}$[2], that is, in $\mathbf{AC^0}$ circuits with PARITY gates. Such circuits, in fact, can preform impressively powerful tasks (see Healy [2006]).

5.6 Pseudorandom Generators

Now we are heading towards a PRG for any polynomial-time algorithm, as apposed to the "toy example" in the previous section.

Definition 5.12. For $S : \mathbb{N} \to \mathbb{N}$, a function $G : \{0,1\}^* \to \{0,1\}^*$ is called S-pseudo random generator, if

1. $|G(z)| = S(|z|)$, for any $z \in \{0,1\}^*$.
2. G on input of length ℓ runs for $2^{O(\ell)}$ steps.
3. For U_ℓ (a uniform random string of length ℓ) and any circuit C of size $O\left(S(\ell)^3\right)$, $\left|\Pr\left[C\left(U_{S(\ell)}\right) = 1\right] - \Pr\left[C\left(G\left(U_\ell\right)\right) = 1\right]\right| \leq \frac{1}{10}$.

Theorem 5.13. *If there exists an S-pseudo random generator then* $\mathbf{BPTIME}\left(S(\ell)\right) \subseteq \mathbf{DTIME}\left(2^{O(\ell)}\right)$.

Example: If there exists an $\exp(\ell)$-pseudo random generator then $\mathbf{BPP} = \mathbf{P}$. However, a more modest pseudo random generator will still give some non-trivial result. For example, if there exists an $\exp(\ell^{\varepsilon})$-pseudo random generator for some constant ε this gives a deterministic simulation of $\mathbf{BPP}$ in $\exp(\log^c n)$ for some constant $c = c(\varepsilon)$.

Proof. Let L be a language that is determined by the random Turing machine M with running time $S(\ell)$, on input of length ℓ. Let $r \in \{0,1\}^{S(\ell)}$ be the random bits that M uses. If G can be used to derandomize the pseudo-random generator then the proof is done. If not, this can be used to create circuits that will contradict G as a pseudo-random generator. If G satisfies the following:

$$\left| \Pr_{r \in \{0,1\}^{S(\ell)}}[M_r(x) = 1] - \Pr_{z \in \{0,1\}^{\ell}}[M_{G(z)}(x) = 1] \right| \leq \frac{1}{10}$$

We can use G to derandomize M: Let M' be a deterministic Turing machine. For every $z \in \{0,1\}^{\ell}$, M' simulates $M_{G(z)}(x)$ and decides by the majority. M' will be correct on all input x, for each $x \in L$:

$$\Pr_{r \in \{0,1\}^{S(\ell)}}[M_r(x) = 1] \geq \frac{2}{3}$$

$$\Pr_{z \in \{0,1\}^{\ell}}\left[M_{G(z)}(x) = 1\right] \geq \frac{2}{3} - \frac{1}{10} = \frac{17}{30}$$

And for each $x \notin L$:

$$\Pr_{r \in \{0,1\}^{S(\ell)}}[M_r(x) = 1] \leq \frac{1}{3}$$

$$\Pr_{z \in \{0,1\}^{\ell}}\left[M_{G(z)}(x) = 1\right] \leq \frac{1}{3} + \frac{1}{10} = \frac{13}{30}.$$

The runtime will be $S(\ell) \cdot 2^{\ell}$, which is $2^{O(\ell)}$ (because $S(\ell) = 2^{O(\ell)}$).
If G does not satisfies the previous condition, meaning

$$\left| \Pr_{r \in \{0,1\}^{S(\ell)}}[M_r(x) = 1] - \Pr_{z \in \{0,1\}^{\ell}}[M_{G(z)}(x) = 1] \right| > \frac{1}{10}$$

for an infinite number of x's, then it can be used to contradict the definition of G as a pseudo random generator (if this is true only for a finite number of x's, then we can create M'' that has these x's hard-coded, and then we can use G to derandomize M'', as explained before).
Let $\{x_i\}_{i \in I}$ be an infinite series of x's, one for each length, that satisfy the above condition. The series of circuits $\{C_i\}$, such that C_i on input r, has x_i hard-coded,

and simulates $M_r(x_i)$ (if there is no x_i of its length, C_i returns 0). $\{C_i\}$ distinguishes between r and $G(z)$ with probability larger than $\frac{1}{10}$. $\mathbf{SIZE}\,(C_i) = O\left((S(\ell))^2\right)$, in the proof of $\mathbf{P} \subset \mathbf{P/poly}$ (Theorem 2.12), we have seen that there exists a circuit of size $O(t^2)$ for any deterministic Turing machine with runtime t. $\{C_i\}$ contradicts that G is a pseudo-random generator.

$\square$

In the proof above, we have seen that if we a have pseudo random generator that fools polynomial-size *circuits*, we can use it to derandomize polynomial-time *algorithms*. We needed the generator to fool small circuits (rather than efficient algorithms), because in the proof above, to get a contradiction, for every input length n, the input x_i has to be hard coded into the circuit. This is perhaps a first clue on the connection between circuits and derandomization of efficient uniform computation. In the next lecture, we will see this connection in its full glory.

Lecture 6

Derandomization and Circuit Lower Bounds; Interactive Proof Systems

December 6th, 2012

Lecturer: Gil Cohen Scribe: Sagie Benaim, Yuval Madar

In the last lecture, Section 5.6, we discussed pseudorandom generators against circuits and the implication of such pseudorandom generators for the derandomization of efficient uniform computation.

Unfortunately, we currently do not know how to construct a pseudorandom generator, even one with a modest stretch. However, we do know something remarkable - we know how to translate circuit lower bounds to the construction of pseudorandom generators. In other words, if one can generate the truth table of a "hard" function (namely, a function that cannot be computed by small circuits) then one can generate a string that looks random to small circuits (and then use it to derandomize efficient uniform computation).

Nisan and Wigderson [1994], following Yao [1982]; Blum and Micali [1984], showed that given a strong enough circuit lower bound, it is possible to construct a pseudorandom generator and therefore get a (perhaps partial, yet non-trivial) derandomization of **BPP**. The result of Nisan and Wigderson [1994] were improved and simplified over the years. We state a more recent result by Umans [2003].

Theorem 6.1 (Umans [2003])**.** *There exists a universal constant $0 < c < 1$ such that the following holds. Let $S : \mathbb{N} \to \mathbb{N}$. Given the truth table of a function $f : \{0,1\}^s \to \{0,1\}$ that cannot be computed by a circuit with size at most $S(s)$, there exists an $S(s)^c$-pseudorandom generator.*

Informally, Theorem 6.1 states that given a hard function, there exists a PRG with stretch that is polynomial in the hardness of the function. Unfortunately, due to time (and space) constraints, we do not provide a proof for Theorem 6.1 in these notes. We will apply the theorem in the future (see Theorem 7.2), but for now, only consider its immediate implication - for the derandomization of efficient computation, it is enough to prove strong circuit lower bounds. Perhaps in the early 90s, researchers were more optimistic about proving such bounds. With time, however, it was realized that circuit lower bounds are hard to prove (we discuss one reason for that in Lecture 11), and therefore people considered whether derandomization can be based on uniform hardness assumptions. The following theorem (which we will not prove and not use during the course, however it is certainly worth stating) shows that under the plausible complexity assumption that **BPP** $\neq$ **EXP** [*], this is indeed possible, to some extent.

Theorem 6.2 (Impagliazzo and Wigderson [1998]). *Assume* **BPP** $\neq$ **EXP**. *Then for every* $L \in$ **BPP** *there exists a subexponential* ($2^{n^{o(1)}}$) *time* deterministic *algorithm* A *such that for infinitely many* n*'s*

$$\Pr_{x \sim \{0,1\}^n}[L(x) = A(x)] \geq 1 - 1/n.$$

In fact, the above theorem holds even when x is sampled from any distribution that can be sampled efficiently, that is, a distribution for which there exists a Turing machine, that given n, samples a string of length n from the distribution, in time $\mathrm{poly}(n)$. This means that it would be computationally difficult to output an instance x (for infinitely many input lengths) such that the above derandomization fails.
Although Theorem 6.2 shows that some kind of derandomization is possible under uniform hardness assumptions, it does not give a full fledge derandomization. Indeed, although it beats the naive exponential time derandomization, it still runs in subexponential (as apposed to polynomial) time, and it is promised to work for most, but not for all inputs.
A beautiful and surprising result states that, in fact, one cannot derandomize efficient uniform computation while avoiding proving some sort of circuit lower bounds! This is an deep statement - circuits are crucial to derandomization not just because of our proof techniques (namely, derandomization via pseudorandom generators, that in turn are currently based on non-uniform hardness assumptions) - circuits are crucial to derandomization inherently.
The formal statement relies on the definitions of the complexity class **VP** (Definition 4.7), the problem ZEROP (Problem 4.12) and the permanent of a matrix (Definition 4.8).

Theorem 6.3 (Kabanets and Impagliazzo [2004]). *If ZEROP* $\in$ **P**, *then either* **NEXP** $\not\subseteq$ **P/poly** *or PERM* $\notin$ **VP**.

Note the first lower bound (that **NEXP** $\not\subseteq$ **P/poly**) is believable yet a somewhat weak bound. Indeed in the next section we prove (unconditionally) that the randomized version of **NEXP** is not contained in **P/poly**. It is commonly believed that the second lower bound (PERM $\notin$ **VP**) is also true.
The proof of Theorem 6.3 (which we do give) is quite involved and requires some more background. One particular result needed is **IP** = **PSPACE** and so in the next

[*]While it is widely believed that **BPP** $\neq$ **EXP**, we don't even know how to separate **BPP** from **NEXP** (see Open Problem 5.6).

section we introduce the notion of interactive proof systems. As a side tour, in the next section we also prove the best known lower bound for **P/poly** (Theorem 6.13), which is also based on **IP** = **PSPACE**. Moreover, we prove necessary results for the proof of Theorem 6.3, such as a strengthening of Meyer's Theorem which we saw in Lecture 3 (see Theorem 3.2). In the next lecture we will prove Theorem 6.3 given a theorem due to Impagliazzo et al. [2001] (see Theorem 7.2), which its proof we defer to Lecture 8. Theorem 7.2 is crucial to the proof of Williams' Theorem as well. This is indeed quite a long journey!

6.1 Interactive Proof Systems

We first give the notion of an Interactive Proof System:

Definition 6.4 (Interactive Proof System)**.** An interactive proof system is a multi-round protocol between two parties, a prover and a verifier, such that on each round, messages are exchanged between the verifier and the prover to establish if a string belongs to the language or not. We assume that the prover is all powerful, but cannot be trusted, while the verifier has bounded resources. An interactive proof system must satisfy the following properties:

Completeness There exists a proof strategy for the prover, such that if a string is in the language, then the verifier is convinced of this.

Soundness If a string is not in the language, then no proof strategy (no prover) can convince the verifier that the string is in the language.

Recall the definition of **NP** (given also in Lecture 2, see Definition 2.1): A language $L \in \mathbf{NP}$ if there exists a Turing machine M for which the following holds:

$$x \in L \iff \exists y \in \{0,1\}^{|x|^c} M(x,y) = 1,$$

for some constant c. **NP** is therefore a simple Interactive Proof System where the verifier is a **P** machine: The prover produces a polynomial size certificate and the verifier verifies it in polynomial time. Note that in the definition of **NP** there is no assumption on the hardness of computing $y = y(x)$. The fact that the prover is computationally unbounded is formalized by the existential quantifier. In the following definition, we look at an Interactive Proof System, where the verifier can use random bits to decide if to accept a certificate sent by the prover.

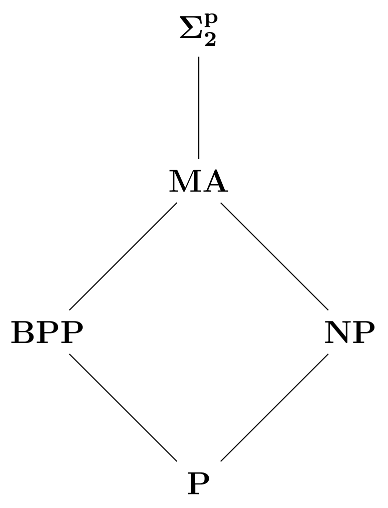


Figure 2: The relation of **MA** to other complexity classes.

Definition 6.5 (**MA**)**.** We define **MA** as the class of languages L for which there exists a probabilistic Turing machine M such that:

$$x \in L \Rightarrow \exists y \in \{0,1\}^{|x|^c} \Pr[M(x,y) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \forall y \in \{0,1\}^{|x|^c} \Pr[M(x,y) = 1] \leq 1/3.$$

Similarly to **NP**, the class **MA** can be viewed as an Interactive Proof System, where the verifier (Arthur) is a probabilistic polynomial time machine (instead of deterministic), and the prover (Merlin) has unbounded resources.

The relation between **MA** and other complexity classes is illustrated in Figure 2. One can informally think of **MA** as a randomized version **NP**, which means that both **BPP** and **NP** are contained in **MA**. The top relation in the diagram ($\mathbf{MA} \subset \mathbf{\Sigma_2^p}$) was proven in Russell and Sundaram [1998].

Definition 6.6 (**IP**, Goldwasser et al. [1985])**.** **IP** is the class of languages described by an Interactive Proof System, where the two parties communicate using messages of polynomial length sent over polynomially many rounds. That is, there exists a prover P and a verifier V s.t, for all provers Q

$$x \in L \Rightarrow \Pr[V \leftrightarrow P \text{ accepts } x] \geq \frac{2}{3}.$$

$$x \notin L \Rightarrow \Pr[V \leftrightarrow Q \text{ accepts } x] \leq \frac{1}{3}.$$

How large is **IP**? Namely how much power does interactive proofs gives? Clearly

NP $\subseteq$ **IP**. At the “early days” researchers had a good reason to believe that **coNP** $\not\subseteq$ **IP**.

Theorem 6.7 (Fortnow and Sipser [1988]). *There exists an oracle O such that* $\mathbf{coNP}^O \not\subseteq \mathbf{IP}^O$.

Recall Section 1.4 - given Theorem 6.7, in order to prove that **coNP** $\subseteq$ **IP** one has to supply a non-relativizing proof. In particular, diagonalization arguments alone are not enough. Some new ideas are required. New ideas were found!

Theorem 6.8 (Shamir [1992] (based on Lund et al. [1990])).

$$\mathbf{IP} = \mathbf{PSPACE}$$

We do not prove Theorem 6.8 in these notes. One can found a proof in Goldreich [2008], Thoerem 9.4. An extension of the class **IP** was provided in Ben-Or et al. [1988], called **MIP**:

Definition 6.9 (**MIP**). **MIP** (Multi prover Interactive Proof) is an interactive proof system in which there are several provers who cannot communicate from the moment the verification process begins.

It was shown in Ben-Or et al. [1988] that having more than two, but a constant number of provers, does not increase the class's computational power. It was shown in Babai et al. [1991] that **MIP** = **NEXP**.

6.2 Three Corollaries of IP = PSPACE

In Theorem 3.1 it was shown that $\mathbf{NP} \subset \mathbf{P/poly} \Rightarrow \mathbf{PH} = \mathbf{\Sigma_2^p}$. This can be read as a conditional lower bound as it is equivalent to saying that if the Polynomial Hierarchy does not collapse to its second level, then **NP** does not have polynomial size circuits). We now prove an additional conditional lower bound:

Corollary 6.10.

$$\mathbf{PSPACE} \subset \mathbf{P/poly} \Rightarrow \mathbf{PSPACE} = \mathbf{MA}$$

Proof. The interaction between Merlin and Arthur is an instance of TQBF (True Quantified Boolean Formula), and so the the prover Merlin is a **PSPACE** machine. Since **PSPACE** = **IP** and by the assumption, Merlin can be replaced by a polynomial size circuit family $\{C_n\}$.

The interaction between Merlin and Arthur can now be carried in one round: Given input x of length n, Merlin sends to Arthur C_n which is of polynomial size in $|x|$. Arthur then simulates the interactive proof getting answers from C_n instead of Merlin. Note that if the input is not in the language, then every circuit sent to Arthur by Merlin fails to act as a prover (it does not have a reasonable chance to convince the verifier). □

We can also improve upon a result by Meyer (Theorem 3.2), which states that

$$\mathbf{EXP} \subseteq \mathbf{P/poly} \Rightarrow \mathbf{EXP} = \mathbf{\Sigma_2^p}.$$

Corollary 6.11 (Babai et al. [1993])**.**

$$\mathbf{EXP} \subseteq \mathbf{P/poly} \Rightarrow \mathbf{EXP} = \mathbf{MA}$$

Proof. Assume $\mathbf{EXP} \subseteq \mathbf{P/poly}$. Since $\mathbf{PSPACE} \subseteq \mathbf{EXP}$ it follows by Corollary 6.10 that $\mathbf{PSPACE} = \mathbf{MA}$. On the other hand, since $\mathbf{EXP} \subseteq \mathbf{P/poly}$, Theorem 3.2 yields $\mathbf{EXP} = \mathbf{\Sigma_2^p}$. The proof then follows since

$$\mathbf{EXP} = \mathbf{\Sigma_2^p} \subseteq \mathbf{PSPACE} = \mathbf{MA}.$$

□

Definition 6.12 ($\mathbf{MA_{EXP}}$)**.** $\mathbf{MA_{EXP}}$ is the class of languages decided by a one round Interactive Proof System where the verifier has exponential time (and exponential number of random bits) and the prover sends an exponentially long proof (this is an exponential analogue to the class $\mathbf{MA}$ defined earlier).

We now present the current best unconditional circuit lower bound for $\mathbf{P/poly}$.

Theorem 6.13 (Buhrman et al. [1998])**.** $\mathbf{MA_{EXP}} \not\subseteq \mathbf{P/poly}$.

Proof. We prove this by contradiction:

$$\begin{aligned}
&\mathbf{MA_{EXP}} \subseteq \mathbf{P/poly} \\
&\Rightarrow \mathbf{PSPACE} \subseteq \mathbf{P/poly} && \text{(Since } \mathbf{PSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{MA_{EXP}}\text{)} \\
&\Rightarrow \mathbf{PSPACE} = \mathbf{MA} && \text{(By Corollary 6.10)} \\
&\Rightarrow \mathbf{EXPSPACE} = \mathbf{MA_{EXP}} && \text{(Explained below)} \\
&\Rightarrow \mathbf{EXPSPACE} \subseteq \mathbf{P/poly}.
\end{aligned}$$

But, **EXPSPACE** $\not\subseteq$ **P/poly**, since in exponential space, one can go over all functions, and for each such function simulate all polynomial size circuits, until a function is found which doesn't agree with any of the circuits (and then output as this function does).
Now we show that if **PSPACE** $=$ **MA**, then **EXPSPACE** $=$ $\mathbf{MA_{EXP}}$ using a standard padding argument described below:
Let $L \in$ **EXPSPACE**. Therefore $L \in \mathbf{DSPACE}\left(n^{2^c}\right)$ for some constant c. Define $L' = \{x \cdot 1^{2^{|x|^c}} \mid x \in L\}$. Then $L' \in$ **PSPACE** as given x we check if it is in L' by simply checking if the first $c \cdot \log_2(|x|)$ bits are in L using polynomial space and using a counter of size $|x|^c$ to check that it is followed by the correct number of 1's. Therefore, by our assumption, $L' \in$ **MA**.
We have that $L \in \mathbf{MA_{EXP}}$ since, given x, we can write $2^{|x|^c}$ 1's at the end of x and check that it is in L'. $\square$

Lecture 7

Kabanets-Impagliazzo Theorem: Derandomization implies Circuit Lower Bounds

December 13rd, 2012

Lecturer: Gil Cohen Scribe: Avishay Tal

In previous lectures we mentioned that hardness implies derandomization, by the works of Nisan, Wigderson and Impagliazzo. In paritcular, if $\mathbf{E} = \mathbf{DTIME}\left(2^{O(n)}\right)$ contains a language which requires exponential size circuits then $\mathbf{P} = \mathbf{BPP}$. This was great news for algorithmic people two decades ago, who thought that circuit lower bounds were right around the corner, thus implying derandomization of all randomized efficient algorithms. However, the work of Razborov and Rudich [1997] on natural proofs showed that proving circuit lower bounds such as $\mathbf{E} \not\subset \mathbf{P/poly}$ cannot be proven by a "natural" proof, assuming the existence of a cryptographic primitive called *one way function* (we will discuss this in Lecture 11).
The question remained whether derandomization can be achieved in other ways? The work of Kabanets and Impagliazzo [2004] (based on previous work of Impagliazzo et al. [2001]) showed that actually derandomization implies some circuit lower bounds: either for Boolean circuits or for arithmetic circuits. Thus, both problems are essentially related, and one cannot hope to solve one without the other. In this lecture we show this proof. More formally, we prove the following (Theorem 6.3, restated).

Theorem 7.1 (Kabanets and Impagliazzo [2004]). *If* $\mathsf{ZEROP} \in \mathbf{P}$ *then either* $\mathbf{NEXP} \not\subset \mathbf{P/poly}$ *or* $\mathsf{PERM} \notin \mathbf{VP}$.

Note that as $\mathsf{ZEROP} \in \mathbf{BPP}$ (and even $\mathsf{ZEROP} \in \mathbf{coRP}$), derandomizing $\mathbf{BPP}$ or even $\mathbf{coRP}$ would yield, according to Theorem 7.1, some circuit lower bounds which are out of our current reach.
The proof of Theorem 7.1 uses many classical as well as more modern results in complexity theory. We now state these results, and then turn to prove Theorem 7.1. We start with the following beautiful theorem.

Theorem 7.2 (Impagliazzo et al. [2001]).

$$\mathbf{NEXP} \subseteq \mathbf{P/poly} \implies \mathbf{NEXP} = \mathbf{EXP}$$

We will see the (quite involved) proof of this theorem in Lecture 8. The next theorem we use, attributed to Meyer, was previously stated and proved in Lecture 3 (see Theorem 3.2).

Theorem 7.3 (Karp and Lipton [1980])**.**

$$\mathbf{EXP} \subseteq \mathbf{P/poly} \implies \mathbf{EXP} = \mathbf{\Sigma_2^p}$$

We state without proof two beautiful theorems (see Arora and Barak [2009], Chapter 17 for proofs) concerning the complexity class $\#\mathbf{P}$. We must appologize, $\#\mathbf{P}$ deserves a broader introduction, but due to lack of space we only give its definition and two relevant classical results concerning it.

Definition 7.4. The complexity class $\#\mathbf{P}$ is the class of functions counting the number of accepting paths of an $\mathbf{NP}$ machine.

$$f \in \#\mathbf{P} \Leftrightarrow \exists \text{ polynomial time non-deterministic } M : f(x) = |\{y \mid M(x,y) = 1\}|$$

Theorem 7.5 (Toda [1989])**.** $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$

Theorem 7.6 (Valiant [1979a])**.** *PERM is $\#\mathbf{P}$-complete*

Note that PERM is also complete for $\mathbf{VNP}$, which is a completely different complexity class defined by (the same) Valiant (we mentioned this class in Lecture 4, see Section 4.2). The following lemma is the last piece of the puzzle needed to prove Theorem 7.1.

Lemma 7.7. *If PERM $\in \mathbf{VP}$ and ZEROP $\in \mathbf{P}$ then $\mathbf{P}^{\mathit{PERM}} \subseteq \mathbf{NP}$.*

Proof. The proof idea is to simulate the oracle to PERM by guessing (using the non-determinism of $\mathbf{NP}$) small circuits for the Permanent (which exist, under the assumption PERM $\in \mathbf{VP}$). The guessed circuits are verified using the polynomial-time algorithm for ZEROP, which is assumed to exists. We now make this formal.
Let $L \in \mathbf{P}^{\mathsf{PERM}}$. Let M be a polynomial-time Turing machine with oracle access to PERM that decides L. Let $p(n)$ be a polynomial bounding the running time of M on input of length n. Clearly, M can't ask questions of size larger than $p(|x|)$ on input x, so it is enough to simulate the oracle for questions of size $\leq p(|x|)$. Also by our assumption there are circuits of size $q(n)$ that computes the permanent of $n \times n$ matrices correctly for some polynomial $q(\cdot)$.
We describe a non-deterministic machine M' that decides L in polynomial-time. On input x, M' will guess circuits $\{C_i\}$ solving PERM for input size $i = 1, \ldots, p(|x|)$. Each C_i will be of size $\leq q(i)$ using our assumption that PERM $\in \mathbf{VP}$. M' will then validate that these circuits do compute the permanent of the corresponding sizes. If

the circuits were successfully verified, M' will simulate M on input x evaluating the circuits on each query to the oracle, and accept/reject according to M.
We will now see how M' can verify the correctness of the circuits it guesses. M' will check that C_1 is the circuit with one gate computing $\mathsf{PERM}_1(A) \triangleq A_{1,1}$. For $t \geq 2$, M' will verify C_t computes the permanent of $t \times t$ matrices assuming that C_{t-1} computes the permanent of $(t-1) \times (t-1)$ matrices. To show this we will use the self-reducible nature of the permanent and our assumption that $\mathsf{ZEROP} \in \mathbf{P}$. Recall that

$$\mathsf{PERM}_t(A) = \sum_{i=1}^{t} A_{1,i} \cdot \mathsf{PERM}_{t-1}(M_{1,i}(A)) \tag{6}$$

where $M_{i,j}(A)$ is the i, j minor of A, i.e. the matrix A without the ith row and jth column. It is enough to verify that

$$C_t(A) \equiv \sum_{i=1}^{t} A_{1,i} \cdot C_{t-1}(M_{1,i}(A)) \tag{7}$$

since by the induction hypothesis and Equation 6 the RHS equals $\mathsf{PERM}_t(A)$ for all As. Equation 7 is equivalent to

$$C_t(A) - \sum_{i=1}^{t} A_{1,i} \cdot C_{t-1}(M_{1,i}(A)) \equiv 0 \tag{8}$$

and there is a small circuit of size $|C_t| + t \cdot |C_{t-1}| + O(t)$ which computes the LHS of Equation 8. Hence, we can use the polynomial time algorithm for ZEROP to verify this equation. $\square$

We are finally ready to prove Theorem 7.1.

Proof of Theorem 7.1. We will show an equivalent form of the statement in the theorem - all of the following can't hold together:

- $\mathsf{ZEROP} \in \mathbf{P}$
- $\mathbf{NEXP} \subseteq \mathbf{P/poly}$
- $\mathsf{PERM} \in \mathbf{VP}$.

We will assume all three hold, and arrive to a contradiction.

$$\begin{aligned}
\mathbf{NEXP} &= \mathbf{EXP} && (\mathbf{NEXP} \subseteq \mathbf{P/poly}, \text{ Theorem 7.2})\\
&= \mathbf{\Sigma_2^p} && (\mathbf{EXP} \subseteq \mathbf{P/poly}, \text{ Theorem 3.2})\\
&\subseteq \mathbf{PH} && (\text{by definition})\\
&\subseteq \mathbf{P}^{\#\mathbf{P}} && (\text{Toda, Theorem 7.5})\\
&= \mathbf{P}^{\mathsf{PERM}} && (\text{Valiant, Theorem 7.6})\\
&\subseteq \mathbf{NP} && (\mathsf{ZEROP} \in \mathbf{P},\ \mathsf{PERM} \in \mathbf{VP}, \text{ Lemma 7.7})
\end{aligned}$$

However, by the non-deterministic time hierarchy (see Theorem 1.7) we know that **NP** is strictly contained in **NEXP**, thus we reach a contradiction. □

Lecture 8
Impagliazzo-Kabanets-Wigderson Theorem

December 27th, 2012

Lecturer: Gil Cohen — Scribe: Gil Cohen, Igor Shinkar

In this lecture we complete the proof of Theorem 7.1 from Lecture 7, by proving Theorem 7.2, which asserts that $\mathbf{NEXP} \subseteq \mathbf{P/poly} \implies \mathbf{NEXP} = \mathbf{EXP}$. Theorem 7.2 has the "same flavor" as other theorems we have encountered, such as

- $\mathbf{NP} \subseteq \mathbf{P/poly} \implies \mathbf{PH} = \mathbf{\Sigma_2^p}$ (see Theorem 3.1).
- $\mathbf{PSPACE} \subseteq \mathbf{P/poly} \implies \mathbf{PSPACE} = \mathbf{MA}$ (see Corollary 6.10).
- $\mathbf{EXP} \subseteq \mathbf{P/poly} \implies \mathbf{EXP} = \mathbf{MA}$ (see Corollary 6.11).

In all of the above, we assume that **P/poly** contains some uniform complexity class, an assertion that we believe to be false, and conclude a collapse between uniform complexity classes, hoping to get a contradiction. As it turns out, the known proof for Theorem 7.2 is significantly more involved than the proofs of the other, similar in spirit, results. This is said with some reservation, as Corollary 6.10 and Corollary 6.11 are "easy" to prove only given $\mathbf{IP} = \mathbf{PSPACE}$.
In order to prove Theorem 7.2 we need to introduce two notions that are interesting on their own right - the notion of *advice*, and the notion of *infinitely often*.

8.1 Turing Machines that Take Advice

The class **P/poly** was defined in Lecture 2 (see Definition 2.10) in terms of circuits, in order to model non-uniformity. Historically however, **P/poly** was defined in terms of Turing machines that "take advice" - circuits were not involved. Informally, a Turing machine is said to take an advice if for every *input length* n the machine has access to a string α_n on top of its input.

Definition 8.1. Let $t : \mathbb{N} \to \mathbb{N}$ and $a : \mathbb{N} \to \mathbb{N}$ be two functions (which we think of as the *time* and *advice* functions, respectively). We say that a language L is in the complexity class $\mathbf{DTIME}(t(n))/a(n)$ if there exists a Turing machine M that runs in time $t(n)$ on inputs of length n, and a family of strings $\{\alpha_n\}_{n=1}^{\infty}$, with $|\alpha_n| \le a(n)$ for all n, such that

$$x \in L \iff M(x, \alpha_{|x|}) = 1.$$

The name of the class **P/poly** is perhaps clearer at this point: to the left of the slash we have the complexity class **P** and to the right **poly** which represents advice of polynomial length. The following theorem makes this formal.

Theorem 8.2.

$$\mathbf{P/poly} = \bigcup_{a,b\in\mathbb{N}} \mathbf{DTIME}\left(n^a\right)/n^b$$

Proof. We first prove the $\subseteq$ direction. The idea is simple - evaluating a circuit on a given input can be done in time which is polynomial (and even linear) in the description length of the circuit and the input. Thus, one can use the advice to store the circuit description. We make this formal. Let $L \in \mathbf{P/poly}$. Then there exist a constant $c \geq 1$ and a family of circuits $\{C_n\}$ computing L such that (for large enough n) $\mathrm{size}(C_n) \leq n^c$. Given x and a reasonable description of $C_{|x|}$, a Turing machine can compute $C_{|x|}(x)$ in time $O(|x|^c)$. By considering the description of C_n as the advice α_n, we get $L \subseteq \mathbf{DTIME}\left(n^c\right)/n^c$.
As for the other direction, let $L \in \mathbf{DTIME}\left(n^a\right)/n^b$ for some constants a, b. The idea again is simple. Since $\mathbf{P} \subset \mathbf{P/poly}$ the Turing machine for L can be simulated by a circuit family. We then take an advantage of the non-uniformity by hard-wiring the advices, one in each circuit. We make this formal. There exist a Turing machine M that runs in time $O(n^a)$ for inputs of length n, and a family of strings $\{\alpha_n\}$ with $|\alpha_n| \leq n^b$ such that $x \in L \iff M(x, \alpha_{|x|}) = 1$. By Theorem 2.12 there exists a family of circuits $\{C_n\}$ of size $O(n^{2a})$ that agrees with M. By hard-wiring the advice α_n to the circuit C_n we get a family of circuits $\{C_n'\}$ that decides L. This conclude the proof as $\mathrm{size}(C_n') \leq O(n^{2a})$. □

By examining Theorem 8.2 one can see a downside of modeling non-uniformity using circuits - the advice and the computation are mixed, as evaluating a circuit is done in time linear in its size. When considering the Turing machines with advice definition, one can separate the computation time from the advice length. For example, one can consider the class $\mathbf{P}/1$, namely, efficient computation with one bit of advice. This class is already strong enough to solve some version of the Halting Problem!
Recall that $\mathbf{NEXP} = \cup_a \mathbf{NTIME}\left(2^{n^a}\right)$ and that $\mathbf{P/poly} = \cup_b \mathbf{SIZE}\left(n^b\right)$. In Theorem 7.2 we assume that $\mathbf{NEXP} \subseteq \mathbf{P/poly}$. Therefore one can ask whether for every a there exists a $b = b(a)$ such that $\mathbf{NTIME}\left(2^{n^a}\right) \subset \mathbf{SIZE}\left(n^b\right)$. Of course, for general sets this doesn't hold (take $A = (0,1] = \cup_n (0,1]$ and $B = \cup_n [\frac{1}{n}, 1]$. Although $A = B$, there is no n such that $A \subseteq [\frac{1}{n}, 1]$). Nevertheless, since we are dealing with very structured sets (complexity classes), this assertion (and more) is true, and will be useful for us.

Lemma 8.3 (Impagliazzo et al. [2001]). *If* $\mathbf{NEXP} \subseteq \mathbf{P/poly}$ *then for every* $a \in \mathbb{N}$ *there exists* $b = b(a)$ *such that*

$$\mathbf{NTIME}\left(2^{n^a}\right)/n \subset \mathbf{SIZE}\left(n^b\right).$$

Proof. For a given $a \in \mathbb{N}$ consider a universal non-deterministic Turing machine $U_a(\cdot,\cdot)$ that on input $(x,i) \in \{0,1\}^* \times \mathbb{N}$ simulates the i'th non-deterministic Turing machine M_i on input x for $2^{|x|^a}$ steps. Note that $L(U_a) \in \mathbf{NEXP}$, and hence, by the assumption of the lemma we have $L(U_a) \in \mathbf{P/poly}$. Therefore, there exists a family of circuits $\{C_n\}$ of size $|C_n| = n^c$ such that $C_{|x,i|}$ computes $L(U_a)$, i.e., $C_{|x,i|}(x,i) = U_a(x,i)$.
We now prove $\mathbf{NTIME}\left(2^{n^a}\right)/n \subset \mathbf{SIZE}\left(n^b\right)$. Take a language $L \in \mathbf{NTIME}\left(2^{n^a}\right)/n$. Then, there is a sequence of advices $\{\alpha_n\}_{n\in\mathbb{N}}$ with $|\alpha_n| = n$, and an index $i = i_L$ such that for every $x \in \{0,1\}^*$ we have $x \in L$ if and only if $M_i(x, \alpha_{|x|})$ has an accepting computation path, where M_i is the i'th non-deterministic Turing machine. Taking the family of circuits $\{C_n\}$ as above we have $C_{|x,\alpha_{|x|},i_L|}(x, \alpha_{|x|}, i_L) = L(x)$. Therefore, by fixing the inputs $\alpha_{|x|}, i_L$ we obtain the desired family of circuits that computes L whose size is at most most $(2n + |i_L|)^{c+1}$. The lemma follows. $\square$

8.2 Infinitely Often

Another notion we use in the proof of Theorem 7.2, which is also quite common in complexity theory, is the notion of *infinitely often*. Roughly speaking, given a complexity class $\mathbf{C}$, the infinitely often version of $\mathbf{C}$ contains all languages that agree with some language from $\mathbf{C}$ on infinitely many input lengths.

Definition 8.4. Let $\mathbf{C}$ be a complexity class. Define the class $\mathbf{io{-}C}$ to contain all languages L for which there exist a language $L' \in \mathbf{C}$ and an infinite set $I \subseteq \mathbb{N}$ such that for every $n \in I$, $L \cap \{0,1\}^n = L' \cap \{0,1\}^n$.

One can easily verify that

Lemma 8.5. *Let* $\mathbf{C}_1, \mathbf{C}_2$ *be two complexity classes. Then*

$$\mathbf{C}_1 \subseteq \mathbf{C}_2 \quad \Longrightarrow \quad \mathbf{io{-}C}_1 \subseteq \mathbf{io{-}C}_2.$$

We will also make use of the following lemma.

Lemma 8.6 (Impagliazzo et al. [2001]). *For any fixed* $c \in \mathbb{N}$ *it holds that*

$$\mathbf{EXP} \not\subset \mathbf{io{-}SIZE}\left(n^c\right).$$

Proof. By the Size Hierarchy Theorem (Theorem 2.11), there exists $n_0 = n_0(c)$ such that for every $n > n_0$ there exists a function f_n on n inputs that cannot be computed by circuits of size n^c yet can be computed by circuits of size at most $4 \cdot n^c$. Given n, one can find, say, the first lexicographic such function and simulate it in exponential time. Denote the resulting language by L_c.
If $L_c \in \textbf{io}-\textbf{SIZE}\left(n^c\right)$ then there exists a family of circuits $\{C_n\}$ such that size$(C_n) \leq n^c$, where infinitely many of them computes f_n (that is, L_c on the respective input length) correctly. This contradicts the fact that at all but the first n_0 circuits in the family cannot compute L_c correctly. □

As a corollary we obtain

Corollary 8.7. *If* $\textbf{NEXP} \subseteq \textbf{P/poly}$ *then for every fixed* $a \in \mathbb{N}$ *it holds that*

$$\textbf{EXP} \not\subseteq \textbf{io}-[\textbf{NTIME}\left(2^{n^a}\right)/n].$$

Proof. By the assumption that $\textbf{NEXP} \subseteq \textbf{P/poly}$ and by Lemma 8.3, there exists $b = b(a)$ such that

$$\textbf{NTIME}\left(2^{n^a}\right)/n \subset \textbf{SIZE}\left(n^b\right).$$

By Lemma 8.5 it follows that

$$\textbf{io}-[\textbf{NTIME}\left(2^{n^a}\right)/n] \subset \textbf{io}-\textbf{SIZE}\left(n^b\right).$$

However, by Lemma 8.6,

$$\textbf{EXP} \not\subset \textbf{io}-\textbf{SIZE}\left(n^b\right),$$

which concludes the proof. □

8.3 A Proof for the IKW Theorem

Recall that we wish to prove Theorem 7.2, which asserts that $\textbf{NEXP} \subseteq \textbf{P/poly} \implies \textbf{NEXP} = \textbf{EXP}$. In other words, we want to show that $\textbf{NEXP} \subseteq \textbf{P/poly}$ and $\textbf{NEXP} \neq \textbf{EXP}$ cannot both hold together. Corollary 8.7 states that under the assumption that $\textbf{NEXP} \subseteq \textbf{P/poly}$ it holds that

$$\forall a \in \mathbb{N} \quad \textbf{EXP} \not\subseteq \textbf{io}-[\textbf{NTIME}\left(2^{n^a}\right)/n]. \tag{9}$$

Given that, all that is left is to prove is

Lemma 8.8. *If* **NEXP** $\neq$ **EXP** *then*

$$\exists a \in \mathbb{N} \quad \mathbf{MA} \subseteq \mathbf{io}\text{−}[\mathbf{NTIME}\left(2^{n^a}\right)/n]. \tag{10}$$

This would conclude the proof of Theorem 7.2. Indeed, since we assume that **NEXP** $\subseteq$ **P/poly** (and thus **EXP** $\subseteq$ **P/poly**), by Corollary 6.11, **EXP** $=$ **MA**. Hence, Equation 9 and Equation 10 stand in contradiction to each other. Therefore, we are left to prove Lemma 8.8. The proof idea is very elegant, and is based on the "easy witness" method introduced by Kabanets [2000].[*]

Proof of Lemma 8.8. Under the assumption **NEXP** $\neq$ **EXP** there exists a language $L^* \in \mathbf{NEXP} \setminus \mathbf{EXP}$ (any complete language for **NEXP** will do). Since $L^* \in \mathbf{NEXP}$, there exist a constant $c^* = c^*(L^*)$ and a non-deterministic Turing machine M^*, that runs in time $O(2^{n^{c^*}})$ on inputs of length n, such that

$$z \in L^* \quad \iff \quad \exists y \in \{0,1\}^{2^{|z|^{c^*}}} \quad M^*(z,y) = 1.$$

What is the implication of $L^* \notin \mathbf{EXP}$? Well, any attempt at deciding L^* in deterministic exponential time is bound to fail! How are we to take advantage of this hardness of L^*? We will suggest a specific attempt at deciding L^* in deterministic exponential time, and benefit from it failing. Clearly, we need double-exponential time in order to simulate the non-determinism of M^* by enumerating over all possible witnesses y. The key idea is to consider only "easy" witnesses, namely, y's that are the truth table of functions that can be computed by small circuits. We make this formal.

For any constant d, consider the following deterministic Turing machine M_d: On input z of length $|z| = n$, enumerate over all circuits of size n^d with n^{c^*} inputs. For any such circuit C, consider its truth table $y = \mathtt{tt}(C)$, which is a string of length $2^{n^{c^*}}$, and check whether $M^*(z,y) = 1$. If we found no such y, the machine rejects z. Otherwise, the machine accepts z.

Observe that if $z \notin L^*$ then there is no witness for z being in L^*, and thus certainly there is no *easy* witness for this false claim. Thus, M_d rejects z. Observe further that

[*]It is certainly worth mentioning that in Kabanets [2000], among other results, the author shows that any **RP** algorithm can be simulated by a subexponential zero-error probabilistic algorithm with some reservations that we choose to omit here.

the running time of M_d is

number of circuits of size n^d with n^{c^*} inputs $\times$
time to evaluate each such circuit on all inputs (so to compute the truth table) $\times$
time to run M^* on the resulting truth table

$$\leq O\left(\left((n^{2d})^{n^d}\right) \cdot \left(2^{n^{c^*}} \cdot n^d\right) \cdot \left(2^{n^{c^*}}\right)\right) = \exp(n).$$

That is, for every fixed d, M_d runs in exponential time, and thus cannot compute L^* correctly. Moreover, we can assume that M_d fail to compute L^* correctly on infinitely many inputs, as otherwise we could have "correct" M_d by adding to it the finite table of the inputs it fails to compute correctly. That is, for every d, there exists an infinite sequence of inputs $Z_d = \{z_i^{(d)}\}_{i \in I_d}$ for which $M_d(z_i^{(d)}) \neq L^*(z_i^{(d)})$, where $I_d \subseteq \mathbb{N}$ is the set of lengths for which there are "bad inputs" (notice that we may take one input per length and Z_d would still remain infinite).
Moreover, we note that M_d makes only one-sided error. Namely, if $z \notin L^*$ then, for every d, M_d would correctly reject z. The only mistakes are false-negative, namely, rejecting inputs that should have been accepted. This may (and will) occur for inputs that has only hard witnesses, that is, witnesses that cannot be computed by circuits of size $|z|^d$.
The conclusion of all of this is that for every d, there exists a non-deterministic Turing machine M'_d that given n, runs in time $2^{n^{c^*}}$ and uses n bits of advice, such that on infinitely many n's prints the truth table of a function that cannot be computed by circuits of size n^d. We now explain this last assertion. The machine M'_d will work properly for the input set I_d, which is infinite. For an input $n \in I_d$, and advice string $z_n^{(d)}$, the machine M'_d guesses a string $y \in \{0,1\}^{2^{n^{c^*}}}$ and checks whether $M^*(z_n^{(d)}, y) = 1$. If the answer is true, the machine M'_d prints y.
Note that the machine uses n bits of advice (the string $z_n^{(d)}$), and runs in time $O(2^{n^{c^*}})$ - the time required to guess y, check whether $M^*(z_n^{(d)}, y) = 1$ and print y. Moreover, if $n \in I_d$ then $z_n^{(d)}$ is an input that is falsely rejected by M_d, and thus, by the above discussion, $z_n^{(d)} \in L^*$, though any witness for this fact - and there are such witnesses - cannot be computed by circuits of size n^d. This implies that the machine M'_d would guess and print a $y \in \{0,1\}^{2^{n^{c^*}}}$ that is the truth table of a function that cannot be computed by circuits of size n^d, as long as n belongs to the infinite set I_d, which was our assertion.
With $\{M'_d\}_d$ in hand, we are ready to prove Equation 10. Let $L \in \mathbf{MA}$. Then there exists a constant $d = d(L)$ such that for any input x, Merlin (the non-determinism) sends Arthur (the probabilistic verifier) a proof $y \in \{0,1\}^{|x|^d}$ for the claim "$x \in L$".

Arthur then tosses $|x|^d$ random bits, and decides (deterministically, given the random bits) in time $|x|^d$ whether to accept x given y.
We now derandomize Arthur. We restrict ourselves to the case where $n = |x| \in I_d$. By the above, there exists a Turing machine M'_d that runs in time $O(2^{n^{c^*}})$, which is exponential in this universal constant c^*, and is independent of d. The machine M'_d prints the truth table of an n^d-hard function. This hard function can be used with the Nisan-Wigderson PRG (Theorem 6.1), which in turn, allows us to derandomize Arthur. This simulation of Arthur takes time $n^{O(d)}$. Since we are using n-bits of advice, runs in non-deterministic time $2^{n^{c^*}} + n^{O(d)} = O\left(2^{n^{c^*}}\right)$ [*], and correctly computes L for all inputs with length in I_d, we get that $L \in \mathbf{io}-[\mathbf{NTIME}\left(2^{n^{c^*}}\right)/n]$. The proof then follows since this holds for every $L \in \mathbf{MA}$ with the *same* constant c^* in the right hand side. $\square$

As mentioned, Theorem 7.2 is used not only to prove Theorem 7.1, but also to prove Williams' Theorem - our main goal in the course. In the next two lectures we finally give a full proof for Williams' Theorem.

[*] At this point a magic / cheating of asymptotic is crucial - c^* is fixed before d, as c^* is some function of a fixed language in $\mathbf{NEXP} \setminus \mathbf{EXP}$, while d may vary depending on the language $L \in \mathbf{MA}$. Nevertheless, both c^* and d are constants in n. Thus, the expression $2^{n^{c^*}}$ asymptotically dominates $n^{O(d)}$.

Lecture 9
$\mathbf{NEXP} \not\subset \mathbf{ACC^0}$ - Part 1

January 3rd, 2013

Lecturer: Gil Cohen   Scribe: Dean Doron

In this lecture we finally prove Williams' theorem.

Theorem 9.1 (Williams [2011b]). **$\mathbf{NEXP} \not\subset \mathbf{ACC^0}$**

Though the result is interesting by itself, what is perhaps more interesting is the conceptual message of Williams' work (which already appeared in Williams [2010]) - a non-trivial algorithm for satisfiability can be used to prove circuit lower bounds.
Consider a Boolean circuit C of size s on n inputs. Checking whether C is satisfiable, namely, whether there exists an input $x \in \{0,1\}^n$ for which $C(x) = 1$, can be done naively in time $O(s \cdot 2^n)$. In particular, we haven't used the output of the circuit C on one input for the other inputs, and therefore we could not have avoided the 2^n factor in the running time. A key step in the proof of Theorem 9.1 is the following theorem, that states that for $\mathbf{ACC^0}$ circuits, one can do slightly better than time 2^n.

Theorem 9.2 (Williams [2011b]). *For every depth d there exists a $\delta = \delta(d) > 0$ and an algorithm, that given an $\mathbf{ACC^0}$ circuit C on n inputs with depth d and size at most 2^{n^δ}, the algorithm solves the circuit satisfiability problem of C in 2^{n-n^δ} time.*

Thus, Williams' idea of proving lower bounds is through improving algorithms for circuit satisfiability for different classes of circuits. This proof technique is quite general and works for all "natural" circuit classes. In fact, the reason it currently applies only to $\mathbf{ACC^0}$ (and not to, say, $\mathbf{NC^1}$ or $\mathbf{P/poly}$) is that we don't have an analog of Theorem 9.2 for other classes of circuits.
We begin this lecture by proving Theorem 9.1 modulo Theorem 9.2. We start proving the latter theorem at the end of this lecture and continue to do so in Lecture 10.

9.1 A NEXP-Complete Language

The class **NEXP** is a key player in Williams' theorem. In particular, we want to show that $\mathbf{NEXP} \not\subset \mathbf{ACC^0}$. A natural attempt at proving that is to focus on some **NEXP** complete language, and show that this language is not in $\mathbf{ACC^0}$. If one throws a rock in a descent computer science faculty (this is ill-advised), he is likely to hit a student that knows of half a dozen **NP** complete problems. We now present a natural **NEXP** complete problem.

Consider a 3CNF formula φ on 2^n variables. Assuming there are no two equal clauses in the formula (namely, clauses that contain the same literals), the number of clauses in φ is at most $(2 \cdot 2^n)^3 = 2^{3(n+1)}$ (there are 2^n variables and so $2 \cdot 2^n$ literals). Consider a function $f : \{0,1\}^{3(n+1)} \to \{0,1\}^t$ that gets as input (the binary representation of) a clause number and outputs the clause description in some natural encoding. That is, $f(k)$ outputs the indices of the three variables that appears in the clause as well as the respective negations. Clearly, $t = 3(n+1)$.
The function f can be computed by a circuit of size at most $2^{O(n)}$ (every output bit of f can, Theorem 2.8, and there are $O(n)$ of them), and the size of the smallest circuit for computing f depends on the complexity of φ (maybe it is better to say that the size of such circuit can be used as a definition for the complexity of φ). Let C_φ be the smallest circuit for computing f (if there is more than one such circuit, we take the first in lexicographic order, say). We say that C_φ encodes φ. On the other hand, every circuit encodes some 3CNF formula, and for a circuit C we denote by φ_C the 3CNF formula encoded by C.
We now describe the **NEXP** complete problem that we will work with. The problem SUCCINCT-3SAT is the following. Given a circuit C on $3(n+1)$ inputs, of size $\mathrm{poly}(n)$, decide whether $\varphi_C \in \mathrm{SAT}$. Clearly, SUCCINCT-3SAT $\in$ **NEXP** as one can reconstruct φ in time $2^{O(n)}$ by applying C on each clause number, and then guess an exponentially long satisfying assignment to φ, and check whether it satisfies φ, in time $2^{O(n)}$. It turns out that SUCCINCT-3SAT is in fact **NEXP** complete in a very strong sense.

Theorem 9.3 (Papadimitriou and Yannakakis [1986]). *For every language $L \in$ **NTIME** $\left(\frac{2^n}{n^{10}}\right)$ there exists an algorithm that given $x \in \{0,1\}^n$, outputs a circuit C on $n + O(\log n)$ inputs, in time $O(n^5)$ (and thus C has size $O(n^5)$) such that*

$$x \in L \iff C(x) \in \text{SUCCINCT-3SAT}.$$

We leave Theorem 9.3 without a proof both due to lack of time and space but also due to the fact it is somewhat technical. The idea is to obtain more efficient Cook-Levin proofs. We refer the reader to the discussion following Theorem 2.2 in Williams [2011c].
Intuitively, if φ is a 3CNF formula on 2^n variables such that $\mathrm{size}(C_\varphi) = \mathrm{poly}(n)$ then φ must be very structured, so to allow for such a compression. One might suggest that if $\varphi \in \mathrm{SAT}$ then, perhaps, a satisfying assignment for φ is also compressible. More precisely, there exists a circuit W on n inputs, with size $\mathrm{poly}(n)$, that on input $i \in \{0,1\}^n$, the circuit W interprets i as a number in $\{1, 2, \dots, 2^n\}$ and outputs the bit value assigned to the i^{th} variable. As it turns out, this assertion is true assuming

NEXP $\subseteq$ **P/poly**. We state here the theorem that captures this in a somewhat informal manner.

Theorem 9.4 (Williams [2010]). *If* **NEXP** $\subseteq$ **P/poly** *then* SUCCINCT-3SAT *has a succinct witness.*

Proof. The proof of this theorem is in fact implicit in the proof of Theorem 7.2. Recall how we proved Theorem 7.2. Our goal was to show that **NEXP** $\subseteq$ **P/poly** and **NEXP** $\neq$ **EXP** cannot live together. In particular, we showed that **NEXP** $\subseteq$ **P/poly** implies Equation 9 while **NEXP** $\neq$ **EXP** implies the contradicting Equation 10.
We don't really care about these equations at this point. What is important is how did we prove that **NEXP** $\neq$ **EXP** implies Equation 10? Well, we took some language L^* in **NEXP** $\setminus$ **EXP** (such a language exists by the assumption) and considered a specific attempt at solving L^* in exponential time. Such attempt, of course, cannot work and we then found a way to benefit from its failure and to conclude Equation 10. For the proof of this theorem, lets recall what was that specific attempt? Well, given x, for deciding if $x \in L^*$, we iterated over all "easy witnesses" $y \in \{0,1\}^{2^{|x|^{c^*}}}$, namely y's that has a succinct representation in terms of a small circuit. Therefore, to deduced Equation 10 one doesn't have to assume that **NEXP** $\neq$ **EXP**, but rather it is enough to assume that SUCCINCT-3SAT does not have succinct witnesses. This concludes the proof of the theorem. $\square$

9.2 Proof of Theorem 9.1

Proof of Theorem 9.1. We prove the theorem by contradiction. We assume that **NEXP** $\subseteq$ $\mathbf{ACC^0}$ and deduce that

$$\mathbf{NTIME}\left(\frac{2^n}{n^{10}}\right) \subseteq \mathbf{NTIME}\left(2^{n-n^{\delta}}\right)$$

for some constant $\delta > 0$. This stands in contradiction to the non-deterministic time-hierarchy theorem (Theorem 1.7). Let $L \in \mathbf{NTIME}\left(\frac{2^n}{n^{10}}\right)$ and $x \in \{0,1\}^*$. Let C be the (efficiently computable) instance of SUCCINCT-3SAT guaranteed by Theorem 9.3 with respect to L, x. We need to design an algorithm that decides whether or not φ_C is satisfiable in $\mathbf{NTIME}\left(2^{n-n^{\delta}}\right)$.
The following claim states that if **P** $\subseteq$ $\mathbf{ACC^0}$ (which is implied by our assumption **NEXP** $\subseteq$ $\mathbf{ACC^0}$) then there is an $\mathbf{ACC^0}$ circuit C_0 that is equivalent to C and has size comparable to that of C.

Claim 5. *If* $\mathbf{P} \subseteq \mathbf{ACC^0}$*, then there exists an* $\mathbf{ACC^0}$ *circuit* C_0 *that is equivalent to* C *and such that* $\mathrm{size}(C_0) = \mathrm{poly}(\mathrm{size}(C))$. [*].

Proof. since evaluation of a circuit can be done in polynomial (even linear) time, CIRCUIT-EVAL $\in \mathbf{ACC^0}$. Hence, there exist constants c_0, d and an $\mathbf{ACC^0}$ circuit family EVAL_n such that the depth of each circuit is at most d and the size of EVAL_n is at most n^{c_0}. Given C, the circuit C_0 can be obtained by hardwiring the constants corresponding to the description of C into the $\mathbf{ACC^0}$ circuit for CIRCUIT-EVAL, keeping the input of EVAL_n that corresponds to the input of C, free. □

The claim above states that there exists an $\mathbf{ACC^0}$ circuit C_0 equivalent to C of comparable size. Hence we can *guess* it. We now want to show how to check that our guess C_0 is indeed equivalent to the the circuit C. One attempt would be to consider the circuit that on input x outputs 1 iff $C(x) \neq C_0(x)$ and run Theorem 9.2 on it to check whether it is satisfiable or not (it is satisfiable iff C and C_0 are not equivalent). Although C, C_0 are small circuits, C is not an $\mathbf{ACC^0}$ circuit and so the described circuit is also not an $\mathbf{ACC^0}$ circuit, which means that Theorem 9.2 does not apply. The problem with the attempt above is that it treated C, C_0 in a black-box fashion. The reason we like circuits so much is their nice structure that allows for some white-box analysis. This is exactly what we will exploit. More precisely, we will do the following. Label the wires of C from 0 to t, where 0 is the label of the output wire. For every wire i of C we guess an $\mathbf{ACC^0}$ circuit C_i that is supposed to compute the value on the i^{th} wire of C. Notice the nice notational trick - for $i = 0$ we get our original guess C_0. More formally, if the i^{th} wire in C is the result of the AND of the j^{th} and k^{th} wires then for every $x \in \{0,1\}^n$ it should hold that $C_i(x) = C_j(x) \wedge C_k(x)$. A similar condition holds for OR and NOT gates, as well as wires that are connected to the inputs.

Now, consider the $\mathbf{ACC^0}$ circuit C' that outputs the AND of all those conditions over all wires i of C. This circuit also has a constant depth (the maximal depth over all C_i's plus some constant) and size polynomial in the size of C. Moreover, if it outputs 1 for every x, then for every i the circuit C_i is indeed equivalent to the i^{th} wire of C. Hence, since the output wire of C is labeled by 0, C is equivalent to C_0 iff C' is not satisfiable. Since C' is an $\mathbf{ACC^0}$ circuit, we can check if it is satisfiable using the algorithm from Theorem 9.2.

One important thing to notice is that C' has the same number of inputs as C, which according to Theorem 9.3 is $n + O(\log n)$, where n is the length of the string that we

[*]In order to be more formal one should consider the circuit family from which C is taken (as a function of L, x) and talk about the circuit family to which C_0 belongs to. However, as custom, we will speak of a specific circuit and understand that there is a circuit family in the background.

are trying to decide whether it belongs to some language in $L \in \mathbf{NTIME}\left(\frac{2^n}{n^{10}}\right)$. It is crucial that the number of inputs is $n + O(\log n)$ as it is and not just, say, $O(n)$ as the saving that we are exploiting in Theorem 9.2 is very modest.

Now that we have the $\mathbf{ACC^0}$ circuit C_0, which we verified to be equivalent to C, we apply Theorem 9.4 to deduce the existence of a polynomial-size circuit W that encodes a satisfying assignment for φ_C. All that is left is therefore to guess such circuit and verify that it indeed encodes a satisfying assignment for φ_C. Note that we might as well guess our easy witness W to be an $\mathbf{ACC^0}$ circuit, as we know such exists by following the same argument as in Claim 5.

How do we verify that the $\mathbf{ACC^0}$ circuit W encodes a satisfying assignment for φ_C? Again we reduce this problem to an instance of $\mathbf{ACC^0}$ circuit satisfiability and then apply Theorem 9.2. For that we construct the circuit D, which as input gets a (binary encoding of a) clause number k. The output of D on k is 0 iff the assignment encoded by W to the variables in the k^{th} clause are such that the clause is satisfied. Since C_0, W are $\mathbf{ACC^0}$ circuit, the circuit D can be constructed as an $\mathbf{ACC^0}$ circuit as well. To complete the proof, note that D is satisfiable iff W does not encode a satisfying assignment for φ_C. □

9.3 SYM+ Circuits and $\mathbf{ACC^0}$ Circuits.

We are left with proving Theorem 9.2. The proof of this theorem relies on a structural result for $\mathbf{ACC^0}$ circuits. To describe this structural result we give the following definition.

Definition 9.6. A SYM+ circuit of degree d, size s and n input bits is a pair (P, Θ), such that $P : \{0,1\}^n \to \mathbb{Z}$ is a multilinear polynomial of degree d with coefficients in $\mathbb{Z}$ of magnitude at most s and $\Theta : \mathbb{Z} \to \{0,1\}$.

We say that a SYM+ circuit computes $f : \{0,1\}^n \to \{0,1\}$ if $f(x) = \Theta(P(x))$ for every $x \in \{0,1\}^n$. As it turns out, the above model of computation captures the strength of $\mathbf{ACC^0}$, in the following sense:

Theorem 9.7 (Beigel and Tarui [1994])**.** *There is an algorithm, that given an* $\mathbf{ACC^0}$ *circuit C of depth d, size s and n input bits, runs in time $2^{\mathrm{polylog}(s)}$ and outputs a* SYM+ *circuit (P, Θ) of degree at most* $\mathrm{polylog}(s)$ *and size at most $2^{\mathrm{polylog}(s)}$ such that $C(x) = \Theta(P(x))$ for every $x \in \{0,1\}^n$. The implicit quasi-polynomial depends on the constant d.*

In the next lecture we prove Theorem 9.7 and deduce from it Theorem 9.2. We end this lecture by proving the following claim that we will use next time.

Claim 8. *Given a multi-linear polynomial $p : \{0,1\}^n \to \mathbb{Z}$ such that every coefficient is of magnitude at most s, one can evaluate p on all 2^n points in time $2^n \cdot \mathrm{poly}(n, \log(s))$.*

Note that the trivial algorithm requires $2^n \cdot 2^n \cdot \mathrm{poly}(n, \log(s))$ time. The 2^n factor that we save stands for the potential number of monomials.

Proof. By induction on a recursive algorithm. For $n = 1$ the claim is trivial. Assume the claim holds for $n-1$ and take p of n input bits. As p is multilinear, we can write

$$p(x_1, \dots, x_n) = x_1 q_1(x_2, \dots, x_n) + q_2(x_2, \dots, x_n) \tag{11}$$

Now, run in recursion on q_1 and q_2. The running time $R(n)$ is thereby

$$R(n) \leq 2R(n-1) + 2^n \mathrm{poly}(n, \log(s)) \tag{12}$$

where $2^n\mathrm{poly}(n, \log(s))$ is the time that takes to merge the two arrays of evaluations. The merging technique (Lemma 4.2 of Williams [2011b], first appearing in Yates [1937]) can be done by dynamic programming. The result immediately follows. □

Lecture 10

NEXP $\not\subset$ ACC^0 - Part 2

January 17th, 2012

Lecturer: Gil Cohen Scribe: Ami Mor, Or Lotan

In this lecture we complete the two missing parts of Williams' proof from Lecture 9. In the first part part (Section 10.1) we present Yao-Beigel-Tarui's theorem for translating an **ACC^0** circuit into a pseudo-polynomial sized SYM+ circuit (Theorem 9.7). In the second part (Section 10.2) we present a non-trivial satisfiability algorithm for **ACC^0** (Theorem 9.2).

10.1 Yao-Beigel-Tarui Theorem

We first present a structural result for the **ACC^0** circuits. This result states that we can efficiently translate an **ACC^0** circuit into a small SYM+ circuit, where the "small" refers to the coefficients and the degree. The strength of **ACC^0** is in its unbounded fan-in. Without the unobounded fan-in it would not have been surprising that we could express low fan-in circuits with a low degree polynomial (maybe with some weak function in the end to eliminate the MOD gates). The SYM+ circuit's strength is in its symmetric function.
In the proof we will first show how to reduce the fan-in by embedding the "hard" part of the computation inside a symmetric function. After that we will modify an **ACC^0** circuit with small fan-in into a low degree polynomial with a symmetric function on top.

Theorem 10.1. *[A.Yao [1990]; Beigel and Tarui [1994]] There exists a function φ and an algorithm A that given an* **ACC^0** *circuit C on n variables, of depth d and size s, runs in time $2^{O\left(\log^{\varphi(d)} s\right)}$, and outputs a SYM+ circuit (P, Θ) of a poly-logarithmic degree $O\left(\log^{\varphi(d)} s\right)$ and pseudo-polynomial size $2^{O\left(\log^{\varphi(d)} s\right)}$ such that $\Theta(P(x)) = C(x)$ for every $x \in \{0,1\}^n$.*

Proof. **Step 0: Tidying up the circuit.** We can assume w.l.o.g that C has the following three properties, as it can be modified to satisfy them at a polynomial cost in size and a constant cost in the depth. Modifying C to satisfy all of these properties takes $O(s^{d+2})$ time which is small compared to the $2^{O\left(\log^{\varphi(d)} s\right)}$ operations we are willing to make in the entire algorithm. Hence, we can use these modifications whenever we need.

1. C is a *formula*. A formula F is a circuit with a "tree structure" - every inner gate has fan-out 1, every input gate has fan-out at least 1, and every output gate has fan-out 0. The following claim argues that every constant-depth circuit can be translated into a formula without paying any cost in the circuit depth, and paying only a polynomial cost in size:

   Claim 2. *For every circuit C of depth d and size s there exists an equivalent formula F of depth d and size at most s^{d+1}. Moreover, the formula can be computed in time $O(s^{d+2})$.*

   Proof. For every gate g, we denote its level by the longest (directed) path from an input node to g (gates at level i depend only on gates at level at most $i-1$, considering the input gates at level 0). The construction of F is done inductively on the gates levels, bottom-up. Assume all gates up to level $i-1 \in [d]$ have fan-out 1 (i.e., levels $1, 2, \ldots, i-1$ consist of the formulas $F_1, \ldots, F_k$) and let g_j be the root of F_j with fan-out d_j. Then in the next step of the construction each formula F_j is replicated d_j times and each copy is connected to a different parent out of the d_j parents of g_j.

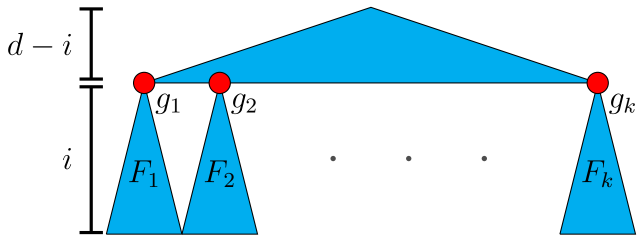


Figure 1: The construction after $i-1$ steps. All inner nodes in $F_1, \ldots, F_k$ have fan-out 1. In the i^{th} step every formula is replicated so that the roots will have fan-out 1 as well.

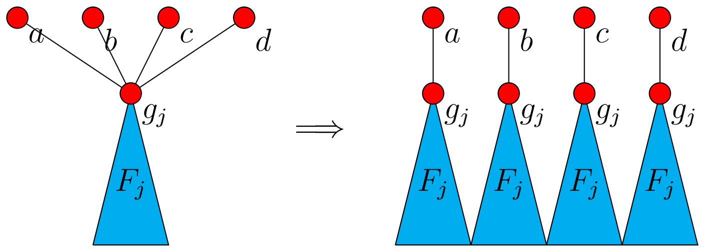


Figure 2: The formula F_j is rooted at g_j and is connected to $d_j = 4$ parents from level $i+1$. At step i, F_j is replicated 4 times so that every copy of g_j (and its sub-tree) has fan-out 1.

The result of the aforementioned construction is a formula $F \equiv C$ of depth d. As for the formula's size, denote by $s(i)$ the size of the new circuit after i iterations. Since s upper bounds the amount of gates at each untouched level then $s(i+1) \leq s(i) \cdot s$. Since size$(F) = s(d)$, and $s(0) = s$, the size of F is upper bounded by s^{d+1}. The running time of the construction is the time required to write down the circuits at all levels. Since at time i the circuit's size is upper bounded by s^i, the total running time is $O\left(\sum_{i=0}^{d} s^i\right) = O\left(s^{d+2}\right)$. $\square$

2. C has no AND gates and no NOT gates. This can be done by replacing every AND gate by a De-Morgen gadget of OR and NOT gates, and then replacing every NOT gate with a single MOD_2 gate with fan-in 2 and a constant 1 hard-wired to one of the inputs. This transformation increases both the size and the depth of the circuit by a factor at most 3. Moreover, it keeps the circuit an $\mathbf{ACC^0}$ circuit even after introducing the MOD_2.

3. C is *layered*. We can assume that every level in C consists of a single type of gate, by inserting fan-in 1 (and fan-out 1) $\mathsf{OR}, \mathsf{MOD}_{p_1}, \ldots, \mathsf{MOD}_{p_k}$ gates as dummy gates that just propagate the input bit up (assuming the circuit family $C \in \mathbf{ACC^0}[p_1, \ldots, p_k]$).

As mentioned, in the following steps we apply the above transformations whenever one of the assumptions is violated. The amount of steps will depend only on the circuit depth (the constant d) and on the amount of types of MOD gates (constant with respect to the family) and thus the asymptotic of the above construction stays the same.

Step 1: Reducing the fan-in of the OR gates. As seen later, the (possibly) large fan-in of the OR gates is a bottle-neck for the proof to work. In the next step of the construction we reduce the fan-in of all OR gates, which is currently bounded only by the current formula size $s^{O(d)}$. The process results in a $2^{O\left(\log^2 s\right)}$-size formula (with a majority gate at its top), in which every OR gate has fan-in at most $O(\log s)$. The proof put in use the ε-biased sets, introduced in Lecture 5 (see Section 5.5), and it slightly deviates from the original proof of the theorem (which uses hash functions). We find it a bit simpler and it slightly improves the parameters (in the original proof, the fan-in of the OR gates is $O(\log^2 s)$ while in this proof it is only $O(\log s)$).

We first recall Definition 5.9 for readability:

Definition 10.3 (Naor and Naor [1990])**.** A set $S \subset \{0,1\}^n$ is called ε-*biased*, if $\forall x \in \{0,1\}^n$ such that $x \neq 0$, $\left|\mathbb{E}_{y \sim S}\left[(-1)^{\langle x,y \rangle}\right]\right| \leq \varepsilon$.

In Theorem 5.11 we saw that an ε-biased set can be constructed efficiently (given n and ε) and has size $O\left(\left(\frac{n}{\varepsilon}\right)^2\right)$. Given an efficiently constructable $\frac{1}{5}$-biased set $S \subseteq \{0,1\}^{s^{O(d)}}$, we know that for any non-zero vector $\alpha \in \{0,1\}^{s^{O(d)}}$, $\langle \alpha, y\rangle = 1$ w.p. at least 0.4:

$$
\begin{aligned}
\left|\mathbb{E}_{y\sim S}\left[(-1)^{\langle \alpha,y\rangle}\right]\right| \le 0.2 \;&\Rightarrow\; \left|\sum_{y\in S}\Pr\left[Y=y\right]\left[(-1)^{\langle \alpha,y\rangle}\right]\right| \le 0.2\\
&\Rightarrow\; \left|\Pr\left[\langle \alpha,y\rangle = 0\right](-1)^0 + \Pr\left[\langle \alpha,y\rangle = 1\right](-1)^1\right| \le 0.2\\
&\Rightarrow\; \left|\Pr\left[\langle \alpha,y\rangle = 0\right] - \Pr\left[\langle \alpha,y\rangle = 1\right]\right| \le 0.2\\
&\Rightarrow\; \left|1 - 2\Pr\left[\langle \alpha,y\rangle = 1\right]\right| \le 0.2\\
&\Rightarrow\; \Pr\left[\langle \alpha,y\rangle = 1\right] \ge \frac{0.2-1}{(-2)} = 0.4.
\end{aligned}
$$

We first show how to generate a distribution of circuits, such that each circuit in the distribution has $O(\log s)$ fan-in OR gates. The probability to sample a circuit from this distribution which answers correctly on any input is greater than 0.9.

Claim 4. *There exists a randomized efficient algorithm A which works as follows. Given an* $\mathbf{ACC^0}$ *formula C on n inputs, with depth $O(d)$ and size $s' = s^{O(d)}$, the algorithm outputs an* $\mathbf{ACC^0}$ *formula C' on n inputs, with depth $O(d)$ and size $s^{O(d)}$. Every OR gate in C' has fan-in at most $O(\log s)$ and w.p. 0.9, $C(x) = C'(x)$ for every $x \in \{0,1\}^n$.*

Proof. Let $S \subseteq \{0,1\}^{s'}$ be a $\frac{1}{5}$-biased set. Fix an input $x \in \{0,1\}^n$ and an OR gate g in C. We denote the input gates of g by $g_{i_1}, \ldots, g_{i_k} \in \{g_1, \ldots, g_{s'}\}$, and define a vector $v_g \in \{0,1\}^{s'}$ such that $v_g[i] = 1$ iff $i \in \{i_1, \ldots, i_k\}$ and $g_i = 1$ (in the calculation $C(x)$. We note that g outputs 1 iff v_g is not the zero vector. Thus, if g outputs 0, for any randomly sampled element $y \in S$ we get $\langle v_g, y\rangle = 0$ w.p. 1. On the other hand, if g should output 1, a random element $y \in S$ yields $\langle v_g, y\rangle = 1$ w.p. at least 0.4.

Let $y \in S$ be some arbitrary chosen element. Calculating the inner product $\langle v_g, y\rangle$ can be implemented using a circuit, by hard wiring into a MOD_2 gate all the input gates of g for which the corresponding bit in y equals 1.

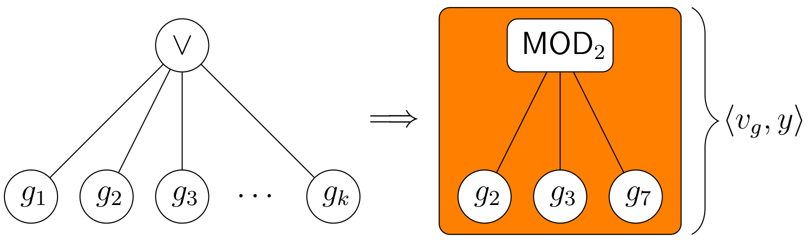
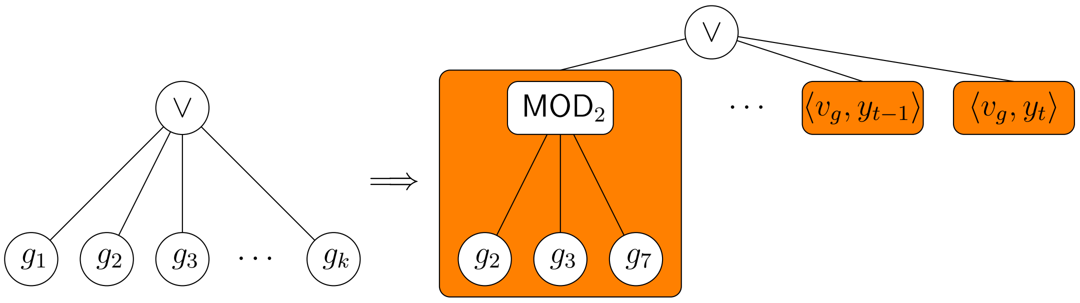


Figure 3: On the left an OR gate g with input gates $g_1, \ldots, g_k$. On the right a circuit implementation of the inner product $\langle v_g, y\rangle$ where the only input gates of g, whose corresponding bit in y is 1, are g_2, g_3, g_7.

Repeating the above construction for t samples $y_1, \ldots, y_t \in S$ chosen independently and taking the disjunction of all the inner products (using an OR gate) we get a circuit which answers like g w.p. $1 - 0.6^t$. Solving for $0.6^t = \frac{1}{10s'} = \frac{1}{10s^{O(d)}}$ we get that t, the fan-in of the OR gate, is $O(\log s)$.



C' is constructed by randomly choosing $y_1, \ldots, y_t$ and replacing each OR gate of C by the afformentioned gadget. Let A_g be the event that the output of the OR gate g is wrongly calculated by its replacement gadget in C'. Using the union bound we can bound the probability that any OR gate is wrongly predicted, which is a satisfying condition for an accurate calculation of the entire circuit on every input:

$$\Pr[C' \not\equiv C] \le \Pr[\exists \mathsf{OR} \text{ gate } g \text{ s.t } A_g] \le \sum_{g \in C} \Pr[A_g] \le \frac{1}{10s'} s' = \frac{1}{10}.$$

□

Corollary 10.5. *There exists an algorithm A that given an* $\mathbf{ACC^0}$ *formula C on n inputs, with depth $O(d)$ and size $s^{O(d)}$ outputs an equivalant* $\mathbf{ACC^0}$ *formula C' with a* MAJORITY *gate on top. C' has depth $O(d)$, size $2^{O(\log^2 s)}$, and every OR gate in it has fan-in at most $O(\log s)$. The running time of A is upper bounded by $2^{O(\log^2 s)}$.*

Proof. The previous claim yields a circuit equivalent to C with probability strictly greater than 0.5. Derandomizing the process by taking the majority of every possible

t tuple of samples $y_1, \ldots, y_t \in S$ yields a circuit of size $2^{O\left(\log^2 s\right)}$ which answers like C on every input. A needs to construct an 0.2-biased set which takes poly(s) time and then write the new circuit C' which takes $2^{O\left(\log^2 s\right)}$ time. □

In the three following steps (steps 2-4) we want to turn the $\mathbf{ACC^0}$ formula (including the majority gate) into a SYM+ circuit (a polynomial and a symmetric function). A polynomial can be viewed as an algebraic circuit with only SUM and PROD gates. The canonical representation of the polynomial is such that all the PROD gates are at the bottom, and they represent the monomials participating in the polynomial. In this representation, the degree of the polynomial is the largest fan-in of any PROD gate. The largest coefficient is smaller than the sum of coefficients which is the SYM+ circuit size. Since our goal is to create a polynomial with low degree and small coefficients, it is reflected by a SUM and PROD circuit with a small fan-in to the PROD gates and a small size.

Step 2: Replacing the OR and MOD gates with arithmetic gates. Our next goal is to transform the formula into an *almost algebraic circuit* contaning SUM, PROD and temporary eq_p gates (along with the symmetric majority gate at the root).

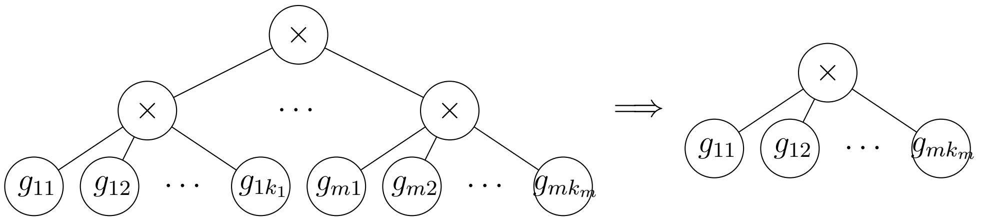
1. **Eliminating the OR gates.** Recall that $\bigvee_{i=1}^{k} g_i = 1 - \prod_{i=1}^{k} (1 - g_i)$, thus any OR gate can be replaced by a triplet of SUM-PROD-SUM gates which maintains the layered structure of the formula. Due to step 1 the fan-in of the new introduced PROD gate is $O(\log s)$.

2. **Eliminating the MOD$_p$ gates.** For every prime p, we introduce a new gate eq_p that gets as an input an integer n and returns $n \mod p$. Fermat's little theorem tells us that $0^{p-1} = 0 \mod p$, while $x^{p-1} = 1 \mod p$ for every $x \in \mathbb{Z}_p^*$. Thus, we can replace every **MOD**$_p$ gate on inputs $g_1, \ldots, g_k$ by $eq_p\left((\sum g_i)^{p-1}\right)$. The latter is implemented by a series of $p-2$ PROD gates followed by the new eq_p and thus increase the MOD$_p$ layer by a factor of p. Since $\mathbf{ACC^0} = \bigcup AC^0[m]$ then the original circuit family uses counters of at most m (constant with respect to the input size and s). Since $p \leq m$ the above procedure increase the circuit depth by a factor at most m.
   We stress that due to Fermat's little theorem all eq_p gates always output 0 or 1 (even though we relate to them as modulo p gates over the integers). Also note that these gates have fan-in 1 compared to the unbounded fan in of the original MOD$_p$ gates.

Step 3: Pushing the multiplication gates down. Recall that our goal is to represent a polynomial using circuit with canonical form in which all multiplication

gates are at the bottom level (i.e., representing the polynomial as a sum of monomials). In this step we inductively push the PROD gates from the top-most level towards the bottom-most layer (which gets only inputs).
The induction step is done by switching between PROD and some other layer without affecting the size of the circuit by much.

1. PROD gate on top of a PROD gate: Assume a PROD layer that precedes a PROD layer, we will unite them into one product.



   Since every PROD gate in the circuit has fan-in at most $O(\log s)$ (from step 2) uniting two layers of PROD gates results in a PROD gate with fan-in at most $O\left(\log^2 s\right)$. In the worst case all $O(d)$ layers consists of PROD gates, so by the end of the induction we could end up with a $\log^{O(d)} s$ fan-in PROD gate . The size of the circuit may only shrink.

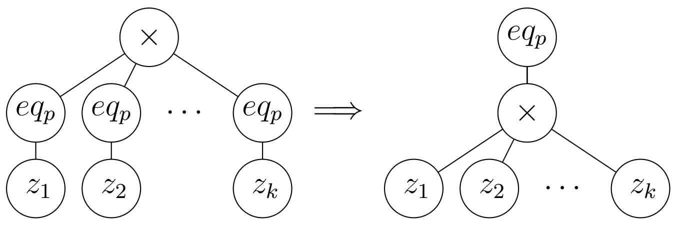
2. PROD gate on top of a SUM gate: Let g be a PROD gate and let $g_1, \ldots, g_l$ be its input gates. Denote by $y_1^i, \ldots, y_{k_i}^i$ the inputs of the i^{th} SUM gate g_i. Having this notations, the calculation made by g is given by

$$\left(y_1^1 + \cdots + y_{k_1}^1\right)\left(y_1^2 + \cdots + y_{k_2}^2\right)\cdots\left(y_1^l + \cdots + y_{k_l}^l\right).$$

   Switching between the two layers can be simply done by opening all brackets.

   We first note that in the resulting circuit the fan-in of the new PROD gates remains the same as we take one element from each bracket. The circuit size though, may increase from $2^{O\left(\log^2 s\right)}$ up to $2^{\log^{O(d)} s}$. Denote by $k = \max\{k_i\}$ the largest fan-in, the number of introduced PROD gates is bounded by k^l. Using the fact that the induction is up to bottom we have that $k \le 2^{O\left(\log^2 s\right)}$ and thus the circuit size is increased by factor of at most $k^l \le \left(2^{O\left(\log^2 s\right)}\right)^{\log^{O(d)}(s)} = 2^{\log^{O(d)} s}$.

3. PROD gate on top of a eq_p gate: In this case the product PROD gate g outputs $(z_1 \mod p)(z_2 \mod p)\cdots(z_k \mod p)$ (as the circuit is layered). By step 2 we know the eq_p gates output only 0 and 1 so we can apply the equality $(z_1 \mod p)(z_2 \mod p)...(z_k \mod p) = z_1 z_2 ... z_k \mod p$ by switching between the two layers



Note that generally it is not true that $(z_1 \mod p)(z_2 \mod p)\ldots(z_k \mod p) = z_1 z_2 \ldots z_k \mod p$ because the outer product in the left hand side is not done modulo p and we can get a result which is bigger than p. This transformation does not change the circuit size, nor change its depth.

We conclude that after this step we have an equivalent circuit with the following properties:

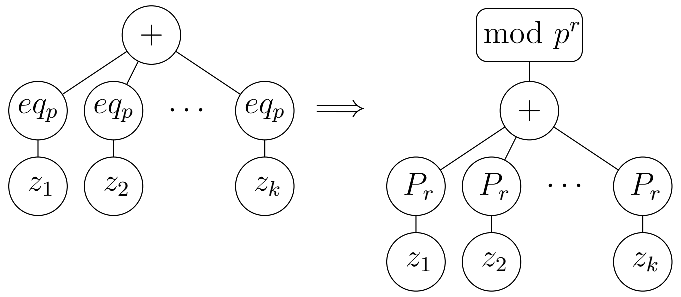
1. All the PROD gates are located at the bottom-most level and have fan-in at most $\log^{O(d)}(s) = \text{polylog}(s)$.
2. The size of the circuit is primarily determined by the number of times we used the PROD after SUM manipulation, but the size is increased by the power of $\log^{O(d)}(s)$ at most d times, so the size is at most $2^{\log^{O(d)}(s)}$.

Step 4 removing the eq_p to get a SYM+ circuit: By now we are left with a circuit with the following criteria: All of the PROD gates or located at its bottom layer, SUM and eq_p gates or located in the mid-layers, and a symmetric majority gate stands at its top layer. In this last step we use the fact that a SYM+ circuit consists of an easy to calculate polynomial on which a hard symmetric function is composed. In this last step we consider the top symmetric gate as θ (initialized as the majority function), and our goal is to get rid of the eq_p. Since our calculation is $\theta(p(x))$ we have the obligation to leave the harder calculations to the function θ.
In order to get a polynomial in the canonical form we want to take the SUM gates down and embed the modulus calculation inside the θ function. For the substitution between layers of eq_p and layers of SUM we will use a set of polynomials called *Modulus amplifying polynomials.* These polynomials have a parameter r that represents the new modulus we amplify to. The properties we want from these polynomials:

- If $(x \mod p) = 0$ then $P_r(x) \mod p^r = 0$
- If $(x \mod p) = 1$ then $P_r(x) \mod p^r = 1$

With these two properties we can conclude that for r large enough:

$$\sum (z_i \mod p) = \sum (P_r(z_i) \mod p^r) = \left(\sum P_r(z_i)\right) \mod p^r$$



The first equality holds due the modulus amplification while the last equality holds because if p^r is larger than the size s' of the circuit, then $\sum (P_r(z_i) \mod p^r)$ is actually a sum of at most s' 0's and 1's and therefor the sum doesn't reach the modulo. Hence, $\sum z_i \mod p = (\sum P_r(z_i)) \mod p^r$.
We will first present the polynomials and then discuss how to use them and how it affects the polynomial coefficients and degree.

Claim 6. *[Beigel and Tarui [1994]; Gopalan et al. [2008]] For every p and r there exist a polynomial P_r of degree $2r-1$ such that:*

- *If $(x \mod p) = 0$ then $P_r(x) \mod p^r = 0$*
- *If $(x \mod p) = 1$ then $P_r(x) \mod p^r = 1$*

Proof. To find the polynomial we will use an algebraic trick:

$$\begin{aligned} 1 &= (x+(1-x))^{2r-1} = \sum_{i=0}^{2r-1}\binom{2r-1}{i}x^i(1-x)^{2r-1-i} \\ &= (1-x)^r\sum_{i=0}^{r}\binom{2r-1}{i}x^i(1-x)^{r-1-i} + x^r\sum_{i=r}^{2r-1}\binom{2r-1}{i}x^{i-r}(1-x)^{2r-1-i} \end{aligned}$$

The last term can be separated into 2 polynomials: $1 = (x-1)^r r(x) + x^r u(x)$. We define $P_r(x) = x^r u(x)$.

- If $x = 0(\mod p)$ then we can write $x = kp$ for some integer k and $P_r(x) = P_r(kp) = p^r k^r u(x)$ which is divisible by p^r. Therefore $P_r(x) = 0(\mod p^r)$.
- If $x = 1(\mod p)$ then we can write $x = kp+1$ for some integer k and $P_r(x) = 1-(1-x)^r r(x) = 1-(-kp)^r r(x)$. Since $(-kp)^r r(x)$ is divisible by p^r we get that $P_r(x) = 1(\mod p^r)$.

These polynomials also have low degree $(2r-1)$ and the maximal coefficient is bounded by $2^{2r} = 4^r$. □

We will use these polynomials each time we want to transform $\sum z_i \mod p$ into $\left(\sum P_r(z_i)\right) \mod p^r$ where $r = \lceil \log_p(s') \rceil$. Note that this operation essentially switches between a SUM layer and an eq_p layer. This can happen at most $O(d)$ times because at the start we have at most $O(d)$ layers of eq_p gates and after every switch we embed one layer of eq_p gates from the circuit into θ.
After we plug P_r in the circuit and switch between the layers we have again PROD gates in the mid-layers so we need to apply step 3 again. Since P_r has maximal coefficient $4^r = 2^{2\log s}$ and degree $2r = O(\log(s))$ it can multiply the maximal coefficient of the circuit by $2^{2\log s}$ and increase its degree by $2r$. Therefore the degree is at most $\log^{O(d)} s + O(d \log s) = O\left(\log^{\phi(d)}(s)\right)$. and the maximal coefficient $2^{\log^{f(d)} s} \cdot \left(2^{2\log s}\right)^d = 2^{\log^{\phi(d)}(s)}$ for some function ϕ. At the end the θ function will be composed by the majority operation along with $O(d) \mod p^r$ operations.

□

10.2 Non Trivial Satisfiability Algorithm for $\mathbf{ACC^0}$

Our goal now is to solve the circuit satisfiability problem for $\mathbf{ACC^0}$ circuits. Given an $\mathbf{ACC^0}$ circuit find whether there exists an input on which the circuit outputs 1 (in such case we say that the circuit has a satisfying assignment, even though circuits have no assignments). Naively, we can check satisfiability of any circuit of size s on n inputs by going over all inputs and simulating the circuit. This procedure runs in time $O(s \cdot 2^n)$. In particular, if the size is $s = 2^{n^\delta}$ then the running time is $O(2^{n+n^\delta})$. We will show an algorithm that runs in time $O(2^{n-n^\delta})$ which is fast enough for our missing part from Williams' proof.
This proof relies heavily on the aforementioned structural result for $\mathbf{ACC^0}$ circuits. This result shows that it is possible to translate an $\mathbf{ACC^0}$ circuit into a small SYM+ circuit. That translation is important because we know of a fast way to check whether a SYM+ is satisfiable using the SYM+ evaluation lemma presented in the previous lecture.

Proof of Theorem 9.2. The algorithm works in three steps:

1. Efficiently constructing an $\mathbf{ACC^0}$ circuit on less inputs but with bigger size, which is satisfiable if and only if C is satisfiable.

2. Use the new circuit to create another equivalent SYM+ circuit whose maximal coefficient is not to large.

3. Use the evaluation lemma presented in Lecture 9 (Claim 8) to try to find a satisfying assignment for the SYM+ circuit.

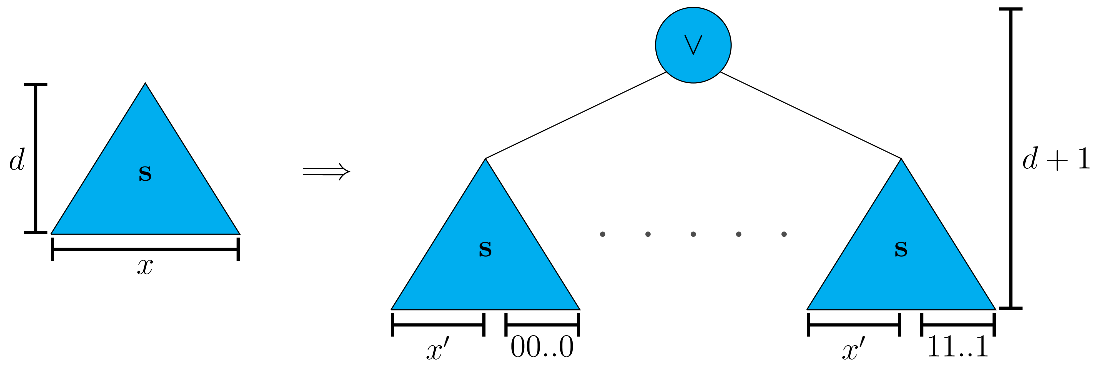
Let $\delta = \delta(d) > 0$ to be defined later. Let C be an $\mathbf{ACC^0}$ circuit C on n inputs with depth d and size $s \leq 2^{n^\delta}$. We construct a new $\mathbf{ACC^0}$ circuit C' on $n' = n - 2n^\delta$ inputs with the following properties:

1. C' has size $s' = 2^{2n^\delta} s \leq 2^{3n^\delta}$.
2. C' has depth $d' = d + 1$.
3. C is satisfiable if and only if C' is satisfiable.

C' computes the following function on an input $x' = (x_1, \ldots, x_{n'})$

$$C'(x') = \bigvee_{x_{n'+1},\ldots,x_n \in \{0,1\}} C(x_1, \ldots, x_{n'}, x_{n'+1}, \ldots, x_n)$$

and is obtained by producing 2^{2n^δ} copies of the circuit C, each wired with a different assignment of the last $2n^\delta$ bits of the input $x = (x_1, \ldots, x_n)$, and taking the OR of all these circuits' outputs. An illustration of the construction of C' is given in the following figure:



Clearly C' is a $(d+1)$-depth $\left(2^{2n^\delta} s + 1\right)$-size $\mathbf{ACC^0}$ circuit. Also, C' is satisfiable if and only if at least one of the copies of C is satisfiable which then again happens if and only if C is satisfiable. We stress that this transformation can be done in $O(2^{3n^\delta}) = o(2^{n-n^\delta})$ time since there are 2^{2n^δ} circuits and writing each down takes time $s \leq 2^{n^\delta}$.

Now, using Yao-Beigel-Tarui structural result (Theorem 10.1) we can calculate in time $2^{\log^{\varphi(d')} s'}$ a SYM+ circuit (P, Θ), equivalent to C', of size at most $2^{\log^{\varphi(d')} s'} \leq 2^{(3n^\delta)^{\varphi(d')}}$ and of degree $\log^{\varphi(d')} s' \leq (3n^\delta)^{\varphi(d')}$ Setting $\delta = \delta(d) = \frac{1}{2\varphi(d')}$ we get

$$\begin{aligned} \deg(P,\Theta) &\leq 3^{\varphi(d')} n^{\delta\varphi(d')} = 3^{\varphi(d+1)}\sqrt{n} = O(\sqrt{n}) \\ \text{time (structural result)} &\leq 2^{3^{\varphi(d')} n^{\delta\varphi(d')}} = 2^{3^{\varphi(d+1)}\sqrt{n}} = 2^{O(\sqrt{n})} = o(2^{n-n^\delta}) \end{aligned}$$

Using the evaluation lemma presented in Lecture 9 (Claim 8), (P,Θ) can be evaluated on all $2^{n'}$ possible inputs in time

$$2^{n'}\text{poly}\left(n' \log\left(\text{size}\left(P,\Theta\right)\right)\right) = 2^{n-2n^\delta}\text{poly}\left(\left(n-2n^\delta\right)\sqrt{n}\right) \leq 2^{n-n^\delta}.$$

The output is in the form of an $O\left(s'n^{d'}\right) = O\left(2^{\sqrt{n}}n^d\right)$ sized vector V holding the possible values of Θ. Hence the satisfiability of (P,Θ) (and equivalently C' and C) can be determined in 2^{n-n^δ} queries to V. The proof follows. $\square$

Lecture 11
Natural Proofs

Lecturer: Gil Cohen Scribe: Ron D. Rothblum

Beautiful results discovered in the 80's and early 90's, such as Theorem 4.3, showed dramatic lower bounds on the power of Boolean circuits. Initially, following the discovery of these results there was high hope that this line of research would lead us to proving that $\mathbf{NP} \not\subset \mathbf{P/poly}$ and as a consequence, to separate $\mathbf{P}$ from $\mathbf{NP}$. Unfortunately, this line of research gradually halted and in fact, the state of the art results (e.g., $\mathbf{NEXP} \not\subset \mathbf{ACC^0}$) are far weaker than what we actually believe to be true (e.g., that $\mathbf{NP} \not\subset \mathbf{P/poly}$).
Given this unfortunate state of affairs, researchers tried to pinpoint what exactly is the reason for our lack of success. One direction that has proved very fruitful and is the focus of this write-up, is to show that certain, very natural proof techniques cannot be used to prove circuit lower bounds. Such results are sometimes called *barriers.* Recall that already in Lecture 1 we encountered the first barrier for separating $\mathbf{P}$ from $\mathbf{NP}$ and the optimistic way around it was to study circuits. Now we have a barrier for circuit lower bounds.
Before proceeding, we point out that while the initial interpretation of such barriers is quite negative and in particular, that the quest for proving lower bounds is somewhat hopeless. A different, more useful interpretation is that such barriers give us a deep insight that may serve as a guide toward finding a novel proof technique. We suggest Barak's employment of non-blackbox techniques in cryptography to avoid blackbox impossibility results (which were thought to be an unsurmountable barrier at the time) as evidence for the more positive interpretation.
We now turn to present the *natural proof barrier*, discovered by Razborov and Rudich [1997] and for which Razborov and Rudich shared the 2007 Gödel prize. Our presentation loosely follows that of Arora and Barak [2009], Chapter 23.
Consider the following natural approach to proving that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. As a first step we identify a particular property that is shared by most functions and in particular by all functions that are in $\mathbf{NP}$. Our interpretation is that this property points to some high degree of complexity (in some undefined sense). Then, we show that a particular function in $\mathbf{P/poly}$ does not share this property. Hence, we can conclude that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$.
Now let us consider this property in slightly more detail. Since, we as human beings, are (in a sense) computationally limited, it seems reasonable that this property will

not be overly complicated. Razborov and Rudich suggested to consider properties for which, given as input a truth table of a function (of size 2^n), it is possible to determine whether the function has the property or not in polynomial-time (i.e., in time poly(2^n)). Loosely speaking, the natural proofs barrier shows that such a property cannot exist, unless some widely believed cryptographic conjectures are false.
More formally, a property is a subset of all Boolean functions. A "natural property" is defined by Razborov and Rudich as follows.

Definition 11.1. A property Π is a *natural property useful against* **P/poly** if it satisfies the following three conditions:

- Usefulness: for every $f \in \mathbf{P/poly}$ it holds that $f \notin \Pi$.
- Constructivity: membership in Π is computable in polynomial-time. That is, there exists an algorithm A that given as input the 2^n-bit long truth table of a function $f : \{0,1\}^n \to \{0,1\}$, runs in time poly($2^n$) and outputs 1 if $f \in \Pi$ and 0 otherwise.
- Largeness: for every $n \in \mathbb{N}$, at least a $1/2^n$ fraction of all n-bit functions have property Π.

The main result of Razborov and Rudich [1997] is that if a particular cryptographic object exists, then natural properties that are useful against **P/poly** cannot exist. The specific cryptographic object that Razborov and Rudich [1997] use is a *sub-exponentially hard pseudo-random function*, an object first defined and constructed by Goldreich et al. [1986].

Definition 11.2. A *sub-exponentially hard pseudo-random function family (*PRF*)* is an efficiently computable ensemble of functions $\mathbb{F} = \{f : \{0,1\}^{n^c} \times \{0,1\}^n \to \{0,1\}\}_{n \in \mathbb{N}}$, where $c > 2$ is a fixed constant, such that for every algorithm A that given as input a truth table of a function $f : \{0,1\}^n \to \{0,1\}$ runs in time $2^{O(n)}$ it holds that

$$\left| \Pr_{k \in_R \{0,1\}^s} [A(f(k,\cdot))] - \Pr_{h \in_R R_n} [A(h)] \right| < 2^{-n^2} \tag{13}$$

where $f(k,\dots)$ denotes the truth table of the function $f_k : \{0,1\}^n \to \{0,1\}$, defined as $f_k(x) = f(k,x)$, and R_n denotes the set of all functions from $\{0,1\}^n$ to $\{0,1\}$.

We remark that the definition above is stronger than that given by Goldreich et al. [1986] in a fundamental way (which is crucial for the proof of Razborov and Rudich

[1997]). Specifically, in Goldreich et al. [1986] the pseudorandomness condition of the PRF is defined with respect to a polynomial-time adversary that only has black-box access to the function. In other words, the adversary can only view the value of the function at some polynomial number of points, rather than seeing the whole truth table (of exponential size) as above. Nevertheless, PRFs as above exist based on widely believed cryptographic conjectures.

Theorem 11.3 (Razborov and Rudich [1997]). *If there exist a sub-exponentially hard* PRF *then there is no natural proof useful against* $\mathbf{P/poly}$.

Proof. Let $\mathbb{F}$ be a PRF and suppose that there exists a property Π that is a natural proof useful against $\mathbf{P/poly}$. Indeed, since $\mathbb{F}$ can be computed by a family of polynomial-size circuits, by the usefulness of Π, for every $k \in \{0,1\}^n$ it holds that $f(k,\cdot) \notin \Pi$. On the other hand, by the largeness condition, $\Pr_{h \in_R R_n}[A(h) \in \Pi] \geq 2^{-n}$. Let A be the algorithm guaranteed by the constructivity condition. Then,

$$\Pr_{h \in_R R_n}[C(h)] - \Pr_{k \in_R \{0,1\}^s}[A(f(k,\cdot))] \geq 2^{-n} \tag{14}$$

in contradiction to the subexponential hardness of the PRF. □

11.1 More on the Largeness Condition

One of the requirements in a natural property useful against $\mathbf{P/poly}$ is that the property hold for a non-negligible fraction of all functions. This requirement seems sensible since if we think of the property Π as measuring the hardness of functions (in some sense), then we would definitely want such hardness to capture most functions (since, as we saw in the course, most functions are very hard to compute). Still it remains a viable possibility that the hardness of $\mathbf{NP}$ will be demonstrated via some combinatorial property of a *specific* problem.

Additional insight in favor of the largeness condition is given by the following argument. We show that a particular class of very natural properties is quite large. Specifically we refer to properties that can be described as *formal complexity measures.*

Recall that a *formal complexity measure* is a function μ that maps every Boolean function on $\{0,1\}^n$ to an non-negative integer, such that:

1. For the (trivial) dictator and anti-dictator functions $\mu(x_i), \mu(1-x_i) \leq 1$.

2. For every two functions f, g it holds that $\mu(f \wedge g), \mu(f \vee g) \leq \mu(f) + \mu(g)$.

Claim 4. *Let μ be a formal complexity measure and suppose that there exists a function f such that $\mu(f) \geq m$. Then, for at least $1/4$ of the function $g : \{0,1\}^n \to \{0,1\}$ it holds that $\mu(g) \geq c/4$.*

Proof. Suppose toward a contradiction that for less than $1/4$ of the functions g it holds that $\mu(g) \geq c/4$.
For every function $g : \{0,1\}^n \to \{0,1\}$, let $h_g : \{0,1\}^n \to \{0,1\}$ be the function defined as $h_g = g\text{XOR}f$. Hence $f = g\text{XOR}h_g = (g \wedge \neg h_g) \vee (\not g \wedge \neg h_g)$, and therefore $\mu(f) \leq \mu(g) + \mu(\neg g) + \mu(h_g) + \mu(\neg h_g)$.
If g is chosen at random then (by our assumption), with probability at least $1/4$ it holds that $\mu(g) \geq (c-2)/4$. Similarly, since h_g is a random function (individually), with probability less than $1/4$ it holds that $\mu(h_g) \geq (c-2)/4$. Similar statements hold for $\neg g$ and $\neg h_g$. Hence, by the union bound, there exists a function g such that $\mu(g), \mu(\neg), \mu(h_g), \mu(\neg h_g) < c/4$ and therefore $\mu(f) < c$, a contradiction. □

In a recent paper Williams [2013] proves, among other things, that in some sense, the constructiveness is unavoidable.

Lecture 12

Linear-Circuit Lower Bounds via Matrix Rigidity

January 24th, 2013

Lecturer: Gil Cohen Scribe: Bharat Ram Rangarajan, Eylon Yogev

In this lecture and the next we discuss a classical circuit lower bound problem that is still wide open.

12.1 Linear-Circuits

Consider an $n \times n$ matrix over a field $\mathbb{F}$. Such a matrix represents a linear transformation from the vector space $\mathbb{F}^n$ to the vector space $\mathbb{F}^n$. A natural question concerns the complexity of computing this transformation.
The model of linear-circuits is a natural model for the computation of linear transformations. Informally, the gates of a linear-circuit compute linear combinations of their input nodes. Thus, they are restricted models of arithmetic circuits (arithmetic circuits additionally have multiplication gates too, see Section 4.2).

Definition 12.1. A *linear-circuit* over a field $\mathbb{F}$ is a directed acyclic graph L in which each directed edge is labeled by a non-zero element of $\mathbb{F}$. If g is a gate with in-coming edges labeled by $\lambda_1, \dots, \lambda_k \in \mathbb{F}$ from gates $g_1, \dots, g_k$, then g computes $v(g) = \lambda_1 v(g_1) + \cdots + \lambda_k v(g_k)$, where $v(g_i) \in \mathbb{F}$ is the value computed at gate g_i. We shall consider linear-circuits with fan-in 2 gates.
Suppose L has n input gates (nodes with no in-coming edges) and m output gates (nodes with no out-going edges). If we denote by $y_1, \dots, y_m \in \mathbb{F}$ the values computed at the output gates of L starting with the values $x_1, \dots, x_n \in \mathbb{F}$ at the input gates, then we will have $y = A_L x$, where $A_L \in \mathbb{F}_{m \times n}$; in other words, the circuit L computes the linear transformation given by the matrix A_L.
The size of a linear-circuit L is defined to be the number of edges in L. The depth of L is defined to be the number of gates on a longest path from an input node to an output node in L.

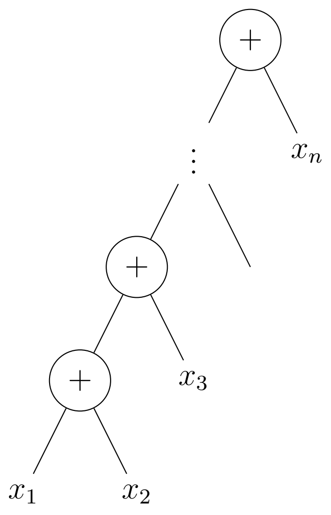
When working over $\mathbb{F}_2$, the gates of any linear-circuit are simply PARITY gates. From here on, unless otherwise stated, we shall work over the field $\mathbb{F}_2$. Let us consider some examples.

Example 12.2. Consider the identity matrix I_n. In this case the circuit simply passes all the input bits as the corresponding output bits, with no intermediate gates.

Example 12.3. Consider the matrix

$$A_{n,n} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

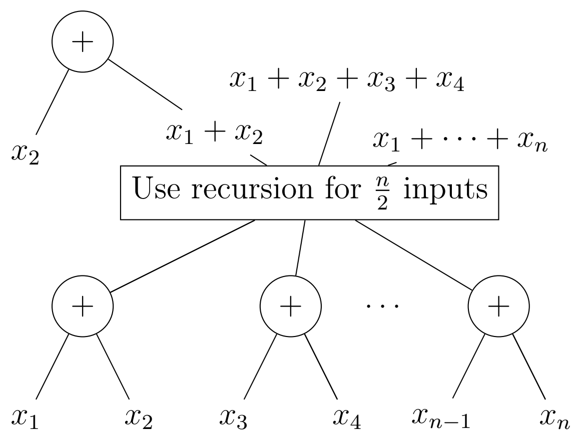
The following linear-circuit computes the transformation given by the matrix $A_{n,n}$. It has $O(n)$ depth and $O(n)$ size.



A shallower circuit can be constructed by the following recursive approach (as suggested to us by Ron Rothblum). Suppose we have already computed the $n/2$ pair sums

$$\begin{aligned} & x_1 + x_2 \\ & x_1 + x_2 + x_3 + x_4 \\ & \qquad \vdots \\ & x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1} + x_n, \end{aligned}$$

then we can XOR them with $x_2, x_4, x_6, \ldots, x_n$ respectively to obtain the remaining sums that we need. This is illustrated by the following figure.



The size and depth of this circuit C_n, on n inputs, is given by the following recursive relations:

$$\text{size}(C_n) = \text{size}(C_{n/2}) + n$$
$$\text{depth}(C_n) = \text{depth}(C_{n/2}) + 2.$$

Solving these recursive relations yields $\text{size}(C_n) = O(n)$ and $\text{depth}(C_n) = O(\log n)$.

In fact, insisting on $O(\log n)$ depth, one can compute any linear transformation from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ by a linear-circuit of size $O(n^2)$. On the other hand, by a counting argument, we can show that most $n \times n$ matrices over $\mathbb{F}_2$ require linear-circuits of size at least $\Omega(\frac{n^2}{\log n})$. This is because the number of $n \times n$ matrices over $\mathbb{F}_2$ is 2^{n^2}, and the number of circuits of size s is upper bounded by $(s^2)^s$. Thus, one must require that $2^{n^2} \leq (s^2)^s$ in order to have enough circuits of size s to compute all these linear transformations. By isolating s, we get that $s = \Omega(\frac{n^2}{\log n})$.
Clearly, a typical circuit has size $\Omega(n)$ as otherwise, since the fan-in is 2, the linear transformation computed by the linear-circuit will not depend on all inputs. It is a major challenge in complexity theory to prove super-linear lower bounds on the size of linear-circuits, even of logarithmic depth, for an explicit family of matrices. That is, we would like to design an algorithm that given n, runs in time $\text{poly}(n)$ and outputs an $n \times n$ matrix that cannot be computed by a linear-circuit of size $O(n)$. Currently there is only one route for resolving this problem - Matrix Rigidity.

12.2 Matrix Rigidity

Recall that the (column) *rank* of a matrix A is the maximum number of linearly independent column vectors of A. We can similarly define the row rank of A. A fundamental result in linear algebra is that the column rank and the row rank of a matrix are always equal, and this is referred to as the rank of the matrix. Another equivalent definition of the rank of A is the largest positive integer r such that A has an $r \times r$ submatrix that is invertible.

Definition 12.4. For an $n \times n$ matrix A over a field $\mathbb{F}$ and a positive integer r, the rigidity of A, denoted $R_A(r)$, is the least number of entries of A that must be changed so that the rank of the resulting matrix reduces to r or less:

$$R_A(r) = \min\{|C| : \mathsf{rank}(A + C) \leq r\}.$$

Here $|C|$ denotes the number of non-zero entries of C.

Let us consider some examples.

Example 12.5. Consider the identity matrix I_n. It has rigidity $R_I(r) = n - r$. Thus, although I_n has full rank, it is quite non-rigid.

Example 12.6. Again, consider the matrix

$$A_{n,n} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

For reducing the rank of A from n to $n/2$, we can change the last 1 in every alternate row to 0, giving $n/2$ total changes. This would reduce the number of linearly independent rows of the matrix to $n/2$, and thus $R_A(n/2) \leq n/2$. To reduce the number of rank of A to $n/4$, divide the set of rows into $n/4$ divisions of 4 rows each, and modify entries of A so that all rows in each division are the same, giving a total of at most $n/4$ linearly independent rows. Since 4 changes are sufficient to make the rows in a division the same, $R_A(n/4) \leq n$. The same approach can be generalized in the case of reducing the rank of A to r: Divide the set of rows into r divisions of n/r rows each, and modify entries of A so that all rows in each division are the same. This would involve making $O(\frac{n^2}{r^2})$ changes within each division, giving $R_A(r) \leq (\frac{n^2}{r})$.

We shall see an upper bound (communicated to us by Stasys Jukna) on $R_A(r)$ for any matrix A.

Lemma 12.7. *For every $n \times n$ matrix A over $\mathbb{F}_2$, $R_A(r) \leq (n-r)^2$.*

Proof. Clearly we may assume $\mathsf{rank}(A) \geq r$. Then it has an $r \times r$ submatrix of full rank. Without loss of generality, let B be this matrix obtained from the first r rows and first r columns of A.

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

Note that the columns of the $r \times (n-r)$ submatrix C are linear combinations of the r columns of B. That is, the i^{th} column of C is given by Bx_i, for some $x_i \in \{0,1\}^r$. Now replace the i^{th} column of E by Dx_i. This way, every column of the new matrix is a linear combination of the first r columns, thus reducing the rank to r. Since we changed only the entries of E, the number of changed entries of A is at most $(n-r)^2$. □

Using a counting argument, we can show that most $n \times n$ matrices have close to maximum rigidity. That is, for almost all matrices A, $R_A(r) = \Omega((n-r)^2/\log n)$.

Lemma 12.8. *For most $n \times n$ matrices A over $\mathbb{F}_2$, it holds that*

$$R_A(r) \geq \frac{n^2 - 3nr}{2\log n}.$$

Proof. The number of $n \times n$ matrices over $\mathbb{F}_2$ with at most c nonzero entries is

$$\sum_{i=0}^{c} \binom{n^2}{i} = O(n^{2c}).$$

The number of $n \times n$ matrices over $\mathbb{F}_2$ of rank at most r is at most

$$\binom{n}{r} \cdot 2^{nr} \cdot (2^r)^{n-r}.$$

Here $\binom{n}{r}$ comes from choosing r rows out of n to be the linearly independent rows, 2^{nr} is an upper bound on the choice of r linearly independent vectors for those rows, and $(2^r)^{n-r}$ is an upper bound on the number of linear combinations of those r vectors deciding the remaining $n-r$ rows. This quantity is upper bounded by 2^{3nr}. We shall get a bound on c using the following inequality:

$$\#(\text{Matrices of rank } r) \cdot \#(\text{c-sparse matrices}) < \frac{1}{2} \cdot (\text{Total number of matrices}).$$

Here, a c-sparse matrix is a matrix with at most c non-zero entries. That is, we want a value of c so that at least half the total number of $n \times n$ matrices over $\mathbb{F}_2$ require modifications in more than c positions to reduce the rank to r or less. Hence,

$$2^{3nr} \cdot n^{2c} < 2^{n^2} \implies c < \frac{n^2 - 3nr}{2 \log n}.$$

□

This leads us to the problem of explicitly constructing matrices of high rigidity. We shall now see an explicit construction of an $n \times n$ matrix A with rigidity

$$R_A(r) = \Omega\left(\frac{n^2}{r} \log\left(\frac{n}{r}\right)\right).$$

A first step could be to ask how many entries of an $n \times n$ matrix need to be modified so that every $r \times r$ submatrix has been affected. Let $C(n, r)$ denote this quantity. Then, suppose we had started with a matrix A which has the property that every $r \times r$ submatrix has full rank, then one would require at least $C(n, r)$ entries to be changed in A so that every $r \times r$ submatrix has been modified, giving us a lower bound of $C(n, r)$ on the rigidity $R_A(r)$ of this matrix.

The quantity $C(n, r)$ has a graph-theoretic interpretation. Given an $n \times n$ matrix A, we can construct a bipartite graph $G = (V_1, V_2, E)$ associated with A as follows: the rows of the matrix A correspond to V_1, and the columns of A correspond to V_2. Then the entry of the matrix in the i^{th} row and j^{th} column can be associated with the edge from vertex i to vertex j in the bipartite graph G. In this case, given the complete bipartite graph $K_{n,n}$, $C(n, r)$ is the number of edges which need to be removed so that the resulting graph has no complete bipartite subgraph $K_{r \times r}$. Computing the value of $C(n, r)$ is known as the *Zarankiewicz problem.*
We shall now see a lower bound on $C(n, r)$.

Lemma 12.9.

$$C(n, r) \geq n(n - r + 1)\left(1 - \left(\frac{r-1}{n}\right)^{1/r}\right)$$

Proof. Consider the complete bipartite graph $K_{n,n} = (V_1, V_2, E)$. Suppose we have removed c edges such that there is no complete bipartite subgraph $K_{r,r}$. We shall use a counting argument to give a lower bound on c.
Note that the number of edges is now $n^2 - c$. For a vertex $v \in V_1$ and a set of vertices $R \in V_2$, we call the pair (v, R) *good* if $R \subseteq N(v)$ and $|R| = r$. Here $N(v)$ denotes

the set of neighbours of vertex v. Now, since the resulting graph has no $K_{r,r}$ as a subgraph, for a particular $R \subseteq V_2$ with $|R| = r$, we can have at most $r-1$ choices of vertices $v \in V_1$ such that (v, R) is good. Thus, the number of good pairs (v, R) is at most $(r-1)\binom{n}{r}$.
Counting in another way, for a fixed vertex $v \in V_1$, the number of good pairs (v, R) is $\binom{d(v)}{r}$, so the total number of good pairs is $\sum_{v\in V_1}\binom{d(v)}{r}$. Here $d(v)$ denotes the degree of vertex v. Thus,

$$\sum_{v\in V_1}\binom{d(v)}{r} \le (r-1)\binom{n}{r}.$$

Note that

$$\sum_{v\in V_1} d(v) = n^2 - c$$

and $\sum_{v\in V_1}\binom{d(v)}{r}$ is a convex function of $\{d(v) : v \in V_1\}$. Hence $\sum_{v\in V_1}\binom{d(v)}{r}$ has minimum value $n\binom{n-\frac{c}{n}}{r}$ (achieved by setting $d(v) = \frac{n^2-c}{n}$ for all $v \in V_1$). Thus, we have

$$n\binom{n-\frac{c}{n}}{r} \le (r-1)\binom{n}{r}.$$

Now,

$$\frac{\binom{n-\frac{c}{n}}{r}}{\binom{n}{r}} \ge \frac{(n-\frac{c}{n}-r+1)^r}{(n-r+1)^r}.$$

Thus,

$$\left(\frac{r-1}{n}\right)^{1/r} \ge \frac{n-\frac{c}{n}-r+1}{n-r+1}.$$

From here it is easy to isolate c to obtain the desired lower bound on c, and thus $C(n,r)$. □

We shall now study the asymptotic behaviour of the bound on $C(n,r)$ obtained above.

Lemma 12.10. *Let* $\log^2 n \le r \le n/2$ *and let* n *be sufficiently large. Then*

$$n(n-r+1)\left(1-\frac{r-1}{n}\right)^{\frac{1}{r}} \ge \frac{n(n-r+1)}{2r}\log\frac{n}{r-1} = \Omega\left(\frac{n^2}{r}\log\frac{n}{r}\right)$$

Proof. As $n(n-r+1) \ge n^2/2$ for $r \le n/2$, it suffices to show that

$$1-\left(\frac{r-1}{n}\right)^{1/r-1} \ge \frac{1}{2r}\log\frac{n}{r-1}.$$

This is equivalent to showing that

$$\left(1-\frac{1}{2r}\log\frac{n}{r-1}\right)^{r}\geq\frac{r-1}{n},$$

or equivalently

$$\left(1-\frac{1}{2r}\log\frac{n}{r-1}\right)^{r/\log\frac{n}{r-1}}\geq\left(\frac{r-1}{n}\right)^{1/\log\frac{n}{r-1}}=\frac{1}{2}.$$

Now note that for large values of n and $r \geq \log^2 n$, the left hand side of the above inequality converges to $e^{-1/2}$ which is greater than $1/2$. Hence the above inequality is true for large n. □

Thus, suppose we can explicitly construct an $n \times n$ matrix which has the property that every $r \times r$ submatrix has full rank, then we have explicit constructions of matrices A with rigidity $R_A(r) = C(n, r) \geq \Omega(\frac{n^2}{r}\log(\frac{n}{r}))$. We shall see (in Lecture 13) that such matrices can be constructed using asymptotically good algebraic-geometric codes, thereby giving us a construction of matrices with the above rigidity. This method gives us the best construction of rigid matrices achieved so far.

12.3 Valiant's Theorem

Our interest in rigid matrices stems from a result of Valiant, which states that matrices which are rigid enough require linear-circuits of super-linear size. Unfortunately, as we shall see, the explicit construction of rigid matrices discussed earlier is not sufficiently strong for this purpose.

Theorem 12.11 (Valiant [1977]). *For any constants $\epsilon, \delta > 0$, if A is a matrix with rigidity $R_A(\epsilon n) \geq n^{1+\delta}$, then any linear-circuit of logarithmic depth that computes the linear transformation given by A, has size $\Omega(n \cdot \log\log n)$.*

Before proving Valiant's Theorem, we will give some intuition. Suppose the transformation is computed by a circuit of size s and of depth $d = c\log n$, where $c = 1/2$. Then each output gate depends on at most $2^d = 2^{c\log n} = \sqrt{n}$ variables, and since the circuit is linear we can write it as a linear combination of $\sqrt{n}$ inputs. Hence, A is a matrix with at most $\sqrt{n}$ nonzero entries at each row, and thus any non-sparse matrices cannot be computed by this circuit.
Valiant's idea is to reduce the depth, which may be $c \cdot \log n$ for $c > 1$ to such case (namely, $c < 1$) by removing not too many edges. Since a small number of edges was

removed, one can say that the linear combination they compute has small rank. For this purpose, we will use the following combinatorial lemma.

Lemma 12.12. *Let $G = (V, E)$ be a directed acyclic graph in which all (directed) paths are of length at most d. Then by removing at most $|E|/\log d$ edges, one can ensure that all paths in the resulting graph have length at most $d/2$.*

Proof. Consider the labeling of each vertex v of the graph G by the length of the longest path that ends at the vertex v. Thus each vertex is now associated with a binary string which is $\log d$ bits long (since any path in the graph is of length at most d). Note that the labeling along a directed edge $e = (v, w)$ strictly increases. That is, the label of w is greater than the label of v.
We shall use these labels to partition the edge set E into $\log d$ sets $E_1, E_2, \ldots, E_{\log d}$, where a directed edge $e = (v, w)$ is in the set E_i if i is the first position from the left in which the labels of the vertices v and w differ.
Let E_i be the set of smallest cardinality. Remove the edges contained in this set E_i from the graph to obtain a graph G'. First note that $|E_i| \leq |E|/\log d$.
We claim that all paths in the resulting graph G' have length at most $d/2$. To see this, consider a new labeling of the vertices of G' obtained by disregarding the position i in the original labels. We claim that this new labeling too retains the property that the labeling along any directed edge in G' strictly increases. To see this, consider an edge $(v, w) \in E_j$ where $j \neq i$. We have two cases:

1. $j > i$: In this case, the i^{th} coordinate belonged to the common prefix of the labels of v and w, and hence after removing it, the labels are still strictly increasing along the edge.

2. $j < i$: In this case, the j^{th} coordinates of the labels of v and w in the original labeling must be 0 and 1 respectively. Since $j < i$, after removing the i^{th} coordinate, the labels are still strictly increasing along the edge.

Thus, since the new labeling requires only $\log d - 1$ bits, any path in the resulting graph has length at most $d/2$. □

We shall now prove Valiant's Theorem which states that matrices with sufficient rigidity require super-polynomial size linear-circuits of logarithmic depth.

Proof. Suppose the transformation is computed by a circuit of size s and of depth $d = c \log n$ where c is some constant. For simplicity, let us assume that d is a powers of 2. Apply Lemma 12.12 to the circuit. Lemma 12.12 assures that by removing at most $\frac{s}{\log d}$ edges, the depth decreases to at most $\frac{d}{2}$. Let $t = \frac{c}{\delta}$ and apply the lemma

$\log t$ times so that we have removed at most $r = \log t \cdot \frac{s}{\log d}$ edges, and the depth is at most $\frac{d}{t}$. Since the circuit is linear, each removed edge is a linear combination of the inputs x_i. Let $b_1, \dots, b_r$ be the corresponding linear forms of the r removed edges. Consider the subcircuit computing the i^{th} output and let $a_i(x)$ be its value on input x. Initially, $a_i(x)$ was a linear combination of at most 2^d input bits, however, now it is a linear combination of at most $m = 2^{\frac{d}{t}}$ original input bits, $x_{i_1}, \dots, x_{i_m}$ and r new variables $b_1, \dots, b_r$ (created by removing the r edges). Hence

$$a_i(x) = \ell_i(b_1, \dots, b_r) + \ell'_i(x_{i_1}, \dots, x_{i_m})$$

where ℓ_i and ℓ'_i are linear transformations. Considering all the outputs i, we get that we can write this in matrix notation. Let ℓ and ℓ' be the matrix with ℓ_i and ℓ'_i as its rows respectively. Let B_1 be the $n \times r$ matrix representing the transformation computed by ℓ and let B_2 be the $r \times n$ matrix that has b_i as its rows. Let C be the $n \times n$ matrix representing ℓ' and let $B = B_1 B_2$. Then we can write

$$A = B_1 B_2 + C = B + C.$$

Note that $\mathsf{rank}(B) \leq r = \frac{s \log t}{\log d}$ and that

$$|C| \leq nm = n \cdot 2^{\frac{d}{t}} = n \cdot 2^{\frac{d}{c/\delta}} = n \cdot 2^{\frac{c \log n}{c/\delta}} = n^{1+\delta}.$$

Thus, we can turn A into a rank $\leq r$ matrix B by changing only $|C| \leq n^{1+\delta}$ entries. Therefore, we get $R_A(\frac{s \log t}{\log d}) \leq n^{1+\delta}$. However, we assumed that $R_A(\varepsilon n) \geq n^{1+\delta}$ and hence we get that $\frac{s \log t}{\log d} \geq \varepsilon n$ or

$$s \geq \frac{\varepsilon}{\log \frac{c}{\delta}} n \log \log n = \Omega(n \log \log n)$$

as required.

$\square$

Lecture 13

Relations between Matrix Rigidity and Coding Theory

January 31th, 2013

Lecturer: Gil Cohen | Scribe: Uri Sherman, Tal Wagner

In this lecture we explore ways to derive matrix rigidity results from coding theory. In the first part, we use a *positive* result in coding theory - namely, a highly non-trivial explicit construction of a code - in order to construct an explicit family of rigid matrices. In the second part, we present an approach suggested by Zeev Dvir, that turns *negative* coding theory results (that is, the non-existence of certain codes) into proofs for the rigidity of certain explicit matrices (which are also derived from codes).

13.1 Basics of Error Correcting Codes

Error correcting codes have many non-trivial applications in theoretical computer science[*], but their "original purpose" is quite simple: Say we want to send a message to another party. Sending the message as is might be problematic, since noise etc. might corrupt part of the message, i.e. flip some 0's to 1's and vice versa. So, instead, we would like to send an encoded version of our message that has the property that the original message can be decoded even though some bits of the encoded message got flipped.
The idea is that the codewords (the encoded version of the message) live in a larger world (say our messages are 50 bits long, then the actual codeword sent might be 100 bits long), in a way that each two legal codewords are far from one another (in Hamming distance). This ensures us that upon receiving a possibly corrupted codeword - as long as not too many bits got flipped - there would be no ambiguity as to which codeword was actually sent, since there would be a unique closest legal codeword. Thus, the original message can be decoded with absolute certainty.
So, now formally: An *error correcting code* (ECC) is an injective map $C : \Sigma^n \to \Sigma^m$ where $n < m$, and Σ is some finite set (thought of as an "alphabet"). The *distance* parameter of an ECC is the minimal Hamming distance between any two codewords. Observe that we can decode with absolute certainty as long as the number of corrupted entries is less than half the distance (as then, there would be a unique nearest non-

[*]For an introductory, see the book "Essential Coding Theory" by Guruswami, Rudra and Sudan; an online draft (currently under preparation) is available at: http://www.cse.buffalo.edu/~atri/courses/coding-theory/book/.

corrupted codeword). The *rate* of an ECC is the ratio between the original message length, and the codeword length: n/m (the rate can be thought of as the proportion of the data sent that actually "contains data"). We will be interested in *linear* codes, that is, ECCs for which the alphabet Σ is some finite field $\mathbb{F}$, and the map $C : \mathbb{F}^n \to \mathbb{F}^m$ defining the ECC is a linear transformation. The corresponding matrix called the *generating matrix* of the code.
As usual, we focus our interest on the asymptotic setting, so we consider families of codes parameterized by the message length n. The size of the alphabet is required to be constant. As clear from the above definitions, the asymptotically optimal possible rate and distance are $\Omega(1)$ and $\Omega(n)$, respectively. An ECC achieving both simultaneously is usually called *asypmtotically good*, and such codes are indeed known to exist. A fairly simple existential proof may be given based on probabilistic arguments (see for example Spielman [2009], Section 11.5). However, as with many combinatorial objects, for computational purposes we need *explicit* constructions, and these are considerably harder to achieve. In our context of linear ECCs, *explicitness* means that the generating matrix of the code can be computed efficiently - that is, in time poly(n) on input n, the desired message length.

The following theorem, which we state here without proof, establishes the existence of an explicit, linear, asymptotically good family of codes. It is heavily based on Algebraic Geometry, following a line of work initiated by Goppa in Goppa [1983]. The proof is due to Tsfasman et al. [1982], and the explicitness is analyzed in Vladuts and Manin [1985]. For a somewhat clearer reference, see Stichtenoth [2009], Section 8.4 and in particular Theorem 8.4.7 [*].

Theorem 13.1. *Let $q = p^2$ for any prime p, and let $\mathbb{F}_q$ denote the finite field over q elements. For any n, there exists an explicit linear ECC $C : \mathbb{F}_q^n \to \mathbb{F}_q^{2n}$ with distance $d \geq (1-\epsilon)n$, where $\epsilon = \frac{2}{\sqrt{q}-1}$.*

Observe that the code described in the theorem has rate exactly $\frac{1}{2}$, and distance arbitrarily close to n. This is nearly tight: A simple result known as Singleton bound states that any code must satisfy $n \leq m - d + 1$, and by plugging $m = 2n$ and rearranging, we see that any code with rate $\frac{1}{2}$ may have distance at most $n + 1$.

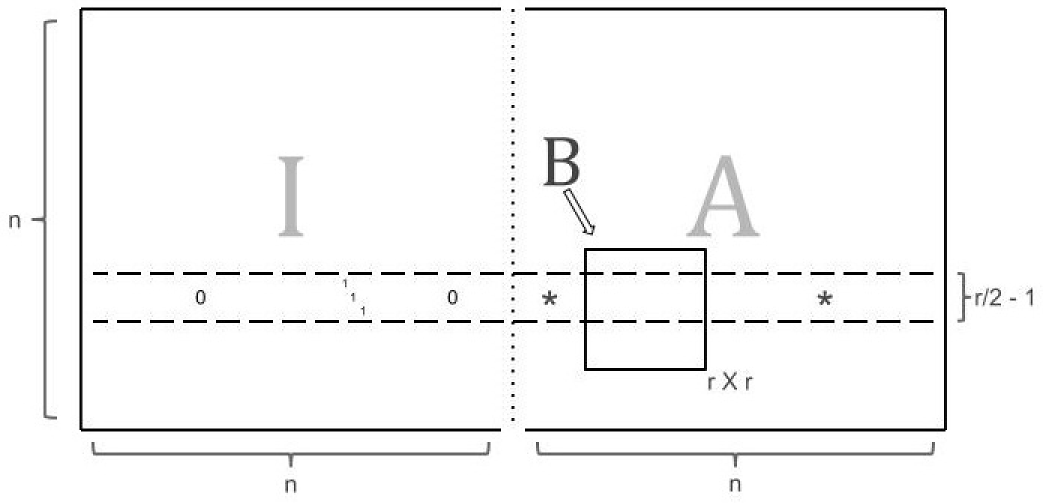
We now show how to use the generating matrices of the above code to construct rigid matrices: In Lecture 12, it was shown that given an $n \times n$ matrix A, if each $r \times r$

[*] In their notation, δ is the *relative distance* - that is, the distance divided by the codeword length - and $\alpha_q(\delta)$ is the rate. So plugging $\delta = \frac{1}{2} - \frac{1}{\sqrt{q}-1}$ into their Theorem 8.4.7 gives the code described in Theorem 13.1. However, their proof does not address the explicitness of this code.

submatrix of A has full rank, then A is (r, s)-rigid with $s = O(\frac{n}{r} \log \frac{n}{r})$. We will produce a matrix that satisfies this property but with half full rank (i.e. every $r \times r$ submatrix will have rank $\geq \frac{r}{2}$).

13.2 Rigid Matrices from Algebraic-Geometry Codes

Consider the transposed $n \times 2n$ matrix representation of C. Notice that any linear combination of the rows of this matrix is a codeword.
By Gaussian elimination (and a permutation of the coordinates), we can bring this matrix to the form $\begin{bmatrix} I & A \end{bmatrix}$, where I is the $n \times n$ identity matrix and A is some $n \times n$ matrix. We claim that A satisfies our property discussed above with $r = 2\epsilon n$. Let B be some $r \times r$ submatrix of A.
Assume by contradiction that $\mathsf{rank}(B) < \frac{r}{2} - 1$. Then any subset of $\frac{r}{2} - 1$ rows of B is linearly dependent, i.e. there exists a linear combination of them that gives the zero vector. Now consider the linear combination with the same coefficients, but with each row of B replaced with its corresponding row in the matrix $\begin{bmatrix} I & A \end{bmatrix}$ (which is an extension of the same row vector). This is depicted in the following figure.



Let v denote the result vector of this linear combination. As mentioned above, v is a legal codeword. Observe the following: The I-side contributes to v exactly one non-zero entry for each of the $\frac{r}{2} - 1$ rows taken. As for the A-side, the B portion of v vanishes (recall that this is the property by which this linear combination was chosen), and the rest of the entries on the A-side of v (marked with "*" in the above figure) may be non-zero, but there are no more than $n - r$ of them. Thus, we have that the number of non-zero entries of v is $\leq \frac{r}{2} - 1 + n - r = n - \epsilon n - 1$. Here we have our contradiction: Since the zero vector is also a codeword, the codeword v we have produced has distance $\leq (1-\epsilon)n - 1$ from it, contradicting the distance bound ensured by our code.

13.3 Dvir's Approach to Matrix Rigidity

In Dvir [2011], Dvir suggests a new approach to the problem matrix rigidity, which is essentially a reduction to a (by then unrelated) sub-domain in coding theory, which concerns ECCs with *local* propetries. He shows that the non-rigidity of a certain family of matrices implies the existence of codes which are not otherwise known to exist, and arguably seem unlikely.

We begin with the necessary coding theory background, then proceed to proving the main result, and conclude with a discussion of its implications.

13.3.1 Local Decodability and Local Correction

We restrict our attention to linear codes, which are sufficient for our purposes, even though the definitions in this section may be given without the linearity requirement.

In general, *decoding* means recovering the original message from a possibly corrupted codeword. In *local decoding*, we wish to recover only a single letter of the message, and do so by reading only a small fraction of the codeword. A *locally decodable code* (LDC) is one that allows us that. The formal definition follows.

Definition 13.2. A linear code $C : \mathbb{F}^n \to \mathbb{F}^m$ is a (q, δ, ϵ)-LDC, if there exists a probabilistic algorithm D mapping $\mathbb{F}^m \times [n]$ to $\mathbb{F}$, with the following guarantee: For every $x \in \mathbb{F}^n$, $v \in \mathbb{F}^m$ with $|v| \leq \delta m$, and $i \in [n]$,

- $\Pr\left[D(C(x) + v, i) = x_i\right] \geq 1 - \epsilon$
- D reads at most q letters from $C(x) + v$.

The guarantee in the definition should be read as follows: For every codeword corrupted at no more than a δ-fraction of its letters, D recovers the i^{th} letter of the original message with high probability, while making only q queries to the corrupted codeword.

We refer the reader to Yekhanin [2012] for a survey of existing LDC constructions. We will use the following construction, stated here without proof. For a proof sketch, see Corollary 3.3 in Dvir [2011].

Fact 13.3. Let $\mathbb{F}$ be a finite field. For every $\gamma, \epsilon > 0$, there is an explicit construction of a linear $(n^\gamma, \delta, \epsilon)$-LDC mapping $\mathbb{F}^n \to \mathbb{F}^m$, with $\delta = \delta(\epsilon) > 0$ and $m = O(n)$.

A related notion to local decoding is *local correction*, in which we wish to recover a portion not of the original message, but rather of the codeword itself, from its corrupted version. That is, given a corrupted codeword, the goal is to correct a target letter of it while reading only a small fraction of all letters. Note that no decoding is necessarily involved at all. A code that allows it is called a *locally correctable code* (LCC), as formalized in the next definition.

Definition 13.4. A linear code $C : \mathbb{F}^n \to \mathbb{F}^m$ is a (q, δ, ϵ)-LCC, if there exists a probabilistic algorithm D mapping $\mathbb{F}^m \times [m]$ to $\mathbb{F}$, with the following guarantee: For every $x \in \mathbb{F}^n$, $v \in \mathbb{F}^m$ with $|v| \leq \delta m$, and $j \in [m]$,

- $\Pr\left[D(C(x) + v, j) = C(x)_j\right] \geq 1 - \epsilon$
- D reads at most q letters from $C(x) + v$.

We emphasize the difference from Definition 13.2: The input to D is now an index of a codeword letter (and not of a letter from the message x), and accordingly, it outputs a letter from $C(x)$ (and not from x).

We remark that under the restriction to linear codes, LCCs are stronger than LDCs, in the sense that any linear LCC implies a linear LDC with the same parameters. For a proof, see Lemma 2.3 in Yekhanin [2012].

13.3.2 Main Theorem

Dvir's main result, which we now state and prove, is a general way to construct linear LCCs from linear LDCs with non-rigid generating matrices. Thereafter, by applying it to the LDCs from Fact 13.3, we will infer that their generating matrices are rigid unless some LCCs exist, which are possibly "too good to be true" (in particular as they would have rate arbitrarily close to 1).

Theorem 13.5. *Let $C : \mathbb{F}^n \to \mathbb{F}^m$ be a linear (q, δ, ϵ)-LDC with generating matrix A. If A is not (r, s)-rigid, then for every $\rho > 0$ there exists a linear code $\mathbb{F}^k \to \mathbb{F}^n$ with $k \geq (1 - \rho)n - r$, which is a $(qs, \rho\delta/s, \epsilon)$-LCC.*

Proof. First note that A is an $m \times n$ matrix with entries in $\mathbb{F}$. Since it's not (r, s)-rigid, we may write $A = L + S$ such that L has rank at most r, and S is s-sparse (recall it means that S has at most s non-zero entries in each row).

We call a column of S "heavy" if it has at least $(sm)/(\rho n)$ non-zero entries. Since S has at most sm non-zero entries altogether, by Markov's inequality it has at most ρn heavy columns. We move these columns from S to L, thus rewriting A as $A = L' + S'$. Observe:

- L' is obtained from L by adding at most ρn non-zero columns, and so its rank can increase only by ρn. Hence, $\mathsf{rank}(L') \leq \mathsf{rank}(L) + \rho n \leq r + \rho n$.

- S' has at most s non-zero entries in each row (a property inherited from S, left unharmed as S' is obtained from S by zeroing some columns), and in addition, has at most $(sm)/(\rho n)$ non-zero entries in each column.

Let $k = \dim(\ker L')$. From the above bound on $\mathsf{rank}(L')$, we get $k \geq (1-\rho)n - r$. The (unique) linear transformation mapping $\mathbb{F}^k$ onto $\ker L'$ defines a linear code $C' : \mathbb{F}^k \to \mathbb{F}^n$, and it remains to show that it is a (qs, δ', ϵ)-LCC, for $\delta' = \rho\delta/s$.

We describe the local correction procedure: Let $x \in \ker L'$ be a codeword. Let $v \in \mathbb{F}^n$ be such that $|v| \leq \delta' n$, and let $i \in [n]$ be the target index to correct. We need to recover x_i with at most qs queries to $x + v$, and we will do so by invoking the local decoding algorithm D_C of the LDC C.

Consider the vector $S'v$. Since v satisfies $|v| \leq \delta' n$, $S'v$ is a linear combination of at most $\delta' n$ columns of S'. But each such column has at most $(sm)/(\rho n)$ non-zero entries, so $S'v$ can have at most $\frac{sm}{\rho n}\delta' n = \delta m$ non-zero entries. Consequently, D_C is guaranteed to recover x_i with probability at least $1 - \epsilon$, while querying at most q letters of $C(x) + S'v$.

It remains to simulate queries on $C(x) + S'v$. To this end we observe:

$$C(x) + S'v = Ax + S'v = L'x + S'x + S'v = S'(x + v).$$

Since S' is s-sparse, we can query an entry of $S'(x+v)$ by making only s queries to $x + v$, as follows: Let J_t be the subset indices of non-zero entries in the t^{th} row of S'. Then $(S'(x+v))_t = \sum_{j \in J_t} S'_{tj}(x+v)_j$, so it is sufficient to query the entries of $(x+v)$ in J_t. Since $|J_t| \leq s$, s queries to $x + v$ suffice.

In conclusion, we have obtained a local correction procedure for C' that succeeds with probability $1 - \epsilon$ on inputs corrupted at a δ'-fraction of entries, while making only qs queries to $x + v$ (which are used to simulate q queries to $C(x) + S'v$). So C' is a (qs, δ', ϵ)-LCC, and the proof is complete. $\square$

Now we can plug the LDCs from Fact 13.3 to obtain the following concrete corollary.

Corollary 13.6. *Either the generating matrices of the LDCs from Fact 13.3 form an explicit $\left(\Omega(n), n^{\Omega(1)}\right)$-rigid family, or for every $\epsilon > 0$ there exists a family of $\left(n^{\Omega(1)}, 1/n^{O(1)}, \epsilon\right)$-LCCs with rate arbitrarily close to 1.*

Proof. Let $\gamma, \epsilon > 0$. Fact 13.3 gives a family of $(n^\gamma, \delta, \epsilon)$-LDCs mapping $\mathbb{F}^n$ to $\mathbb{F}^m$. If for any $\alpha, \beta > 0$ their generating matrices are not $\left(\alpha n, n^\beta\right)$-rigid, then Theorem 13.5 gives, for every $\rho > 0$, a family of $\left(n^{\gamma+\beta}, \rho\delta/n^\beta, \epsilon\right)$-LCCs mapping $\mathbb{F}^k$ to $\mathbb{F}^n$, with $k \geq (1-\rho-\alpha)n$. Their rate is k/n, which by proper choice of ρ and α can be made arbitrarily close to 1. □

13.3.3 Discussion

The immediate question arising from Corollary 13.6 is whether the inferred LCCs are likely to exist, or in other words - does Corollary 13.6 supply hard evidence supporting the rigidity of the matrices from Fact 13.3? The answer is unclear, in part because this range of parameters for LCCs is largely unstudied. The reason is that in most applications, LCCs are required to handle a constant fraction of errors (and not just $1/n^{O(1)}$); under this requirement, rate approaching 1 is unachievable for any code, regardless of being LCC or not. Dvir postulates in Dvir [2011] that the local correction property prevents codes from having such good rate, even when required to handle only a significantly lower fraction of errors. After Dvir's work, it was discovered, however, that Dvir's approach cannot yield rigid matrices enough to deduce the desired circuit lower bounds, but it may come very close to that.
From a broader perspective, the connection of rigidity to coding theory appears valuable because the latter is a widespread field in many aspects of computer science, both theoretical and practical, and is being studied by various scientific communities. Hence, a large body of research on it is available, and progress is made constantly and in many directions. Since Dvir's reduction is general, it may be used in the future to translate progress on LDCs to more promising candidates for explicit families of rigid matrices, and progress on LCCs to more solid evidence - and possibly a proof - for the rigidity of these families.

References

1. L. Adleman. Two theorems on random polynomial time. In *FOCS*, pages 75–83. IEEE Computer Society, 1978.

2. N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

3. S. Arora and B. Barak. Computational complexity: a modern approach. Cambridge University Press Cambridge, UK, 2009.

4. A.Yao. On ACC and threshold circuits. In *FOCS*, pages 619–627. IEEE Computer Society, 1990.

5. L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1(1):3–40, 1991.

6. L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Comput. Complex.*, 3(4):307–318, October 1993. ISSN 1016-3328. URL http://dx.doi.org/10.1007/BF01275486.

7. T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

8. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages, in NC^1. volume 38, pages 150–164. Elsevier, 1989.

9. P. Beame, S. Cook A, and H.J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.

10. R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

11. M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 113–131, New York, NY, USA, 1988. ACM. ISBN 0-89791-264-0. URL http://doi.acm.org/10.1145/62212.62223.

12. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

13. H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *Computational Complexity, 1998. Proceedings. Thirteenth Annual IEEE Conference on*, pages 8–12. IEEE, 1998.

14. S.A. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, 1973.

15. Z. Dvir. On matrix rigidity and locally self-correctable codes. *Computational Complexity*, 20(2):367–388, 2011.

16. L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Information Processing Letters*, 28:249–251, 1988.

17. M. L. Furst, James B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

18. O. Goldreich. Computational complexity: a conceptual perspective. *ACM SIGACT News*, 39(3):35–39, 2008.

19. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

20. O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). In *Electronic Colloquium on Computational Complexity*. Citeseer, 1997.

21. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM. ISBN 0-89791-151-2. URL http://doi.acm.org/10.1145/22145.22178.

22. P. Gopalan, V. Guruswami, and R. J. Lipton. Algorithms for modular counting of roots of multivariate polynomials. *Algorithmica*, 50(4):479–496, March 2008.

23. V. D. Goppa. Algebraic-geometric codes. *Mathematics of the USSR-Izvestiya*, 21(1):75, 1983.

24. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the A.M.S*, 117:285–306, 1965.

25. A. Healy. Randomness-efficient sampling within NC^1. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 398–409, 2006.

26. R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *IEEE Conference on Computational Complexity*, pages 2–12, 2001.

27. R. Impagliazzo and A. Wigderson. Randomness vs. Time: De-randomization under a uniform assumption. In *Journal of Computer and System Sciences*, pages 734–743, 1998.

28. K. Iwama, O. Lachish, H. Morizumi, and R. Raz. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In *Proc. of MFCF*, pages 353–364. Springer-Verlag, 2002.

29. S. Jukna. Boolean function complexity: advances and frontiers. volume 27. Springer, 2012.

30. V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Computational Complexity, 2000. Proceedings. 15th Annual IEEE Conference on*, pages 150–157. IEEE, 2000.

31. V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complex.*, 13(1/2):1–46, December 2004. ISSN 1016-3328. URL http://dx.doi.org/10.1007/s00037-004-0182-6.

32. R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

33. R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC*, pages 302–309, 1980.

34. A. Kojevnikov, A. Kulikov, and G. Yaroslavtsev. Finding efficient circuits using SAT-solvers. *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 32–44, 2009.

35. C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4): 215–217, 1983.

36. C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 2–10. IEEE, 1990.

37. O.B. Lupanov. On the synthesis of contact networks. In *Dokl. Akad. Nauk SSSR*, volume 119, pages 23–26, 1958.

38. A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE, 1972.

39. R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge university press, 1995.

40. J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *STOC*, pages 213–223, 1990.

41. N. Nisan and A. Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, October 1994. ISSN 0022-0000. URL http://dx.doi.org/10.1016/S0022-0000(05)80043-1.

42. C. Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

43. C. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

44. A. Razborov. Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function. In *Mat. Zametki*, pages 598–607, 1986.

45. A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition (Russian). *Matematicheskie Zametki*, 41(4):598–607, 1987.

46. A. Razborov and S. Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

47. R. Rubinfeld. Randomness and computation - Lecture 1, 2006. URL http://people.csail.mit.edu/ronitt/COURSE/S06/index.html.

48. A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.

49. C. Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974.

50. J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

51. A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, October 1992. ISSN 0004-5411. URL http://doi.acm.org/10.1145/146585.146609.

52. A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010.

53. M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 330–335, New York, NY, USA, 1983. ACM. ISBN 0-89791-099-0. URL http://doi.acm.org/10.1145/800061.808762.

54. R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 77–82, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. URL http://doi.acm.org/10.1145/28395.28404.

55. D. Spielman. Course on spectral graph theory - lecture notes, Lecture 11: Introduction to error-correcting codes. 2009. URL http://www.cs.yale.edu/homes/spielman/561/2009/.

56. H. Stichtenoth. More about algebraic geometry codes. In *Algebraic Function Fields and Codes*, volume 254 of *Graduate Texts in Mathematics*, pages 289–309. Springer Berlin Heidelberg, 2009. ISBN 978-3-540-76877-7. URL http://dx.doi.org/10.1007/978-3-540-76878-4_8.

57. L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.

58. S. Toda. On the computational power of PP and $\oplus$P. In *FOCS*, pages 514–519, 1989.

59. M.A. Tsfasman, S.G. Vladuts, and Th. Zink. Modular curves, Shimura curves, and Goppa codes, better than Varshamov-Gilbert bound. *Mathematische Nachrichten*, 109(1):21–28, 1982. ISSN 1522-2616. URL http://dx.doi.org/10.1002/mana.19821090103.

60. C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2): 419–440, 2003.

61. C. Umans. The minimum equivalent DNF problem and shortest implicants. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 556–563. IEEE, 1998.

62. L. Valiant. Graph-theoretic arguments in low-level complexity. *Mathematical Foundations of Computer Science 1977*, pages 162–176, 1977.

63. L. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979a.

64. L. Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261. ACM, 1979b.

65. E. Viola. Guest column: correlation bounds for polynomials over {0,1}. *ACM SIGACT News*, 40(1):27–44, 2009.

66. S.G. Vladuts and Yu.I. Manin. Linear codes and modular curves. *Journal of Soviet Mathematics*, 30:2611–2643, 1985. ISSN 0090-4104. URL http://dx.doi.org/10.1007/BF02249124.

67. R. Williams. Topics in circuit complexity course - Lecture 1. 2011a. URL http://www.stanford.edu/~rrwill/cs354.html.

68. R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *STOC*, pages 231–240, 2010.

69. R. Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of the 2011 IEEE 26th Annual Conference on Computational Complexity*, CCC '11, pages 115–125, Washington, DC, USA, 2011b. IEEE Computer Society. ISBN 978-0-7695-4411-3. URL http://dx.doi.org/10.1109/CCC.2011.36.

70. R. Williams. Guest column: a casual tour around a circuit complexity bound. *ACM SIGACT News*, 42(3):54–76, 2011c.

71. R. Williams. Natural proofs versus derandomization. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 21–30. ACM, 2013.

72. A. Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91, 1982.

73. A. Yao. Separating the polynomial-time hierarchy by oracles. In *Foundations of Computer Science, 1985. Proceedings. 39th Annual Symposium on*, pages 1–10. IEEE, 1985.

74. F. Yates. *The Design and Analysis of Factorial Experiments.* Technical communication. Imperial Bureau of Soil Science, 1937. URL http://books.google.co.il/books?id=YW1OAAAAMAAJ.

75. S. Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.

76. R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226, 1979.