Advanced Complexity Notes

Caleb Koch

June 2019

These were notes taken during an independent study course with Christoph Haase in advanced complexity theory during Trinity Term 2019 at Oxford.

Contents

Counting Complexity

For this topic we cover

- Definitions of PP and #P, alternative definitions of PP, parsimonious reductions for #P-complete problems.
- Natural problems complete for PP and #P.
- Toda's theorem
- Closure of PP under intersection and complement.

Definitions

Fix $\Sigma = \{0,1\}$. Our main sources are [Koz06] and [AB09].

Definition 0.1 (#P [AB09])**.** The set #P is the set of functions $f : \Sigma^* \to \mathbb{N}$ such that there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time DTM M such that $\forall x \in \Sigma^*$

$$f(x) = \left|\left\{y \in \Sigma^{p(|x|)} : M(x,y) = 1\right\}\right|$$

An equivalent definition from [Koz06] is the following: #P is the set of functions $f_M : \Sigma^* \to \mathbb{N}$ such that M is a polynomial-time NDTM and $f_M(x)$ is the number of accepting computations paths of M on x. The equivalence follows from the fact that if M is a NDTM then there is a polynomial-time DTM verifier V which on input (x,y) verifies that y is a valid certificate to $x \in L(M)$ given that y is of length poly$(|x|)$.

One example of a problem in #P is #SAT which counts the number of satisfying assignments of a boolean formula. In fact many natural NP problems have #P analogues.

The next definition establishes a class that can be thought of as the "decision problem analogue" of #P.

Following Kozen's presentation we define the function

$$W(p,L,x) \stackrel{\text{def}}{=} \left|\left\{y \in \Sigma^{p(|x|)} : (x,y) \in L\right\}\right|$$

for a function $p : \mathbb{N} \to \mathbb{N}$, a language L and a string x.

Definition 0.2 (PP or probabilistic polynomial time [AB09, Koz06])**.** A language L is in PP if there exists a language $A \in$ P and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$,

$$x \in L \Leftrightarrow |W(p,A,x)| \geq \frac{1}{2}(2^{p(|x|)}).$$

In the case of NP, the size of the set $W(p,A,x)$ must be at least 1 in order for x to be in L, but in the case of PP we require that at least half of the certificates are (since the total number of certificates is $2^{p(|x|)}$). Intuitively PP corresponds to computing the most significant bit of functions in #P since if $f \in$ #P maps to $[0, N-1]$ on x one has to decide whether $f(x) \geq N/2$.

Definition 0.3 (Alternative definition of PP)**.** A language L is in PP if there is a probabilistic TM for which yes-instances are accepted with probability $> 1/2$ and no instances are accepted with probability $\leq 1/2$.

To see why these are equivalent, note that Definition 0.2 is equivalent to stating that there is a NDTM M such that $x \in L$ iff the number of accepting computation paths is at least as large as the number of rejecting paths. We can define a probabilistic TM M' that simulates M on x and if M rejects then with probability $1/2$ M' will reject and with probability $1/2$ M' will accept. Otherwise if M accepts then M' accepts. This will give a TM that satisfies Definition 0.3.

It is a fairly straightforward exercise to show that $\mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$. We mentioned that PP is the "decision-problem analogue" for #P. There is a nice lemma to motivate this identification but first we need to define FP.

Definition 0.4 (FP)**.** A function $f : \Sigma^* \to \mathbb{N}$ is in FP if there exists a polynomial-time DTM M such that $M(x)$ computes $f(x)$.

Lemma 0.5 ([AB09])**.** $\mathrm{PP} = \mathrm{P} \Leftrightarrow \#\mathrm{P} = \mathrm{FP}$

If #P = FP then we can count the number of accepting/rejecting computation paths in a NTM in polynomial time so PP = P. The other direction is a bit trickier and proven in Lemma 17.7 of [AB09]. The idea is to show that testing if $f(x) > N$ for $f \in$ #P and N with poly($|x|$) bits can be done with a PP procedure. Thus if PP = P we can use a polynomial procedure to test $f(x) > N$ and can thus do a binary search to determine the value of N.

Reductions and #P*-Completeness*

In the decision problem setting, a problem L is said to be NP-hard if a polynomial time algorithm for L would imply that P = NP. This motivates how completeness for #P is defined. Intuitively, we want to say that $f \in$ #P is #P-complete if a polynomial-time algorithm for f implies that #P = FP. With this motivation we say that a TM has oracle access to $f : \Sigma^* \to \Sigma^*$ if it has access to the language $\{\langle x, i\rangle : f(x)_i = 1\}$ (so effectively it can't outright ask for $f(x)$ but it can ask for specific bits in $f(x)$). Then we define FP^f to be the set of functions computable by polynomial-time TMs that have oracle access to f. This notion of oracle access gives us the following deinition.

Definition 0.6 (Oracle based #P-completeness [AB09])**.** A function f is #P-complete if it is in #P and if $g \in$ #P then $g \in \mathrm{FP}^f$.

Note then in particular that if f is #P-complete and $f \in$ FP then $\#\mathrm{P} = \mathrm{FP}^f = \mathrm{FP}$.

In some cases it is useful to consider the stricter notion of *parsimonious reductions* which preserve the uniqueness of certificates.

Definition 0.7 (Parsimonious reduction, parsimonious based #P-completeness)**.** We say that there is a parsimonious reduction from $f : \Sigma^* \to \Sigma^*$ to $g : \Sigma^* \to \Sigma^*$ denoted $f \leq_{\mathrm{pars}} g$ if there is a polynomial-time computable function $h : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$ we have

$$f(x) = g(h(x)).$$

A function $g \in$ #P is said to be #P-complete if $f \leq_{\mathrm{pars}} g$ for all $f \in$ #P.

The following are a few #P-complete problems

- #SAT: Given a boolean formula how many satisfying assignments does it have?
- #2SAT: Given a boolean formula in 2-CNF, how many satisfying assignments does it have?
- #CYCLE: Given a directed graph G, how many simple (i.e. cannot visit a vertex twice) cycles are there?
- PERM: Given an $n \times n$ 0-1 matrix A, compute the permanent of A
- #k-COLORABILITY: Given a graph G how many k-colorings are there?
- #SUBSETSUM: Given $X = \{x_1, ..., x_n\} \subseteq \mathbb{N}$ and $T \in \mathbb{N}$ how many subsets of X sum up to T [HK16]?

The permanent of A is given by

$$\sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where S_n is the symmetric group of order n. This result requires a particularly clever construction and was proven by Valiant in his original 1979 paper that introduced #P. For this reason, the result is sometimes known as Valiant's theorem. One can interpret A as the adjacency matrix of a graph G in which case $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ is a perfect matching. Hence the permanent is simply the number of perfect matchings of G.

There are also a few natural PP-complete problems including

- MAJSAT: For a boolean formula φ decide if more than half the assignments make φ true
- Given an $n \times n$ 0-1 matrix A is Perm(A) $> n!/2$?
- KTH LARGEST SUBSET: Given $X = \{x_1, ..., x_n\} \subseteq \mathbb{N}$ and $K, B \in \mathbb{N}$ decide whether $K \leq |\{Y \subseteq B : \sum_{y \in Y} y \leq B\}|$ [HK16]
- KTH LARGEST m-TUPLE: Given finite $X_1, ..., X_m \subseteq \mathbb{N}$ and $K, B \in \mathbb{N}$ decide if $K \leq |\{x \in X_1 \times \cdots \times X_m : \sum x_i \geq B\}|$ [HK16]

Toda's Theorem

We begin with a brief review of the polynomial hiearchy, PH.

Definition 0.8 (Polynomial hierarchy)**.** The polynomial hierarchy, PH is defined inductively as

$$\begin{aligned} \Sigma_0 &= P \\ \Sigma_{i+1} &= \mathrm{NP}^{\Sigma_i} \\ \mathrm{PH} &= \bigcup_{i \in \mathbb{N}} \Sigma_i. \end{aligned}$$

Toda's theorem is a statement relating #P and PH. It is often stated as $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$ which intuitively states that one can solve any problem in PH given an oracle to a #P problem.

Theorem 0.9 (Toda's theorem)**.**

$$\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$$

There are many ways to prove Theorem 0.9. We follow most closely the exposition by Fortnow [For09].

There are two main lemmas:

Lemma 0.10. $\mathrm{PH} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}}$.

Note that since BPP $\subseteq$ PP by definition, we have $\mathrm{BPP}^A \subseteq \mathrm{PP}^A$ for any oracle A

Lemma 0.11. $\mathrm{PP}^{\oplus \mathrm{P}} \subseteq \mathrm{P}^{\#\mathrm{P}}$.

To prove Lemma 0.10 we make use of the following theorems.

Theorem 0.12 (Valiant-Vazirani Theorem)**.** *There is a probabilistic polynomial-time procedure that, given a Boolean formula φ will output formulas $\psi_1, ..., \psi_k$ such that*

- *if φ is unsatisfiable then each ψ_i is unsatisfiable*
- *otherwise, with high probability, for some i, ψ_i has a unique satisfying assignment.*

Note that $\oplus$P (Parity-P) is the set of languages of polynomial-time NDTMs where x is in the language iff the number of accepting paths is odd.

Corollary 0.13. $\mathrm{NP} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}}$

Proof. Given a boolean formula φ run the procedure from Theorem 0.12 to get $\psi_1, ..., \psi_k$. Accept if any of the ψ_i have an odd number of satisfying assignments, indicating that the formula is satisfiable. This shows that $\mathrm{SAT} \in \mathrm{BPP}^{\oplus \mathrm{P}}$ and hence $\mathrm{NP} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}}$. □

Theorem 0.14 (Papadimitriou-Zachos Theorem)**.** $\oplus \mathrm{P}^{\oplus \mathrm{P}} = \oplus \mathrm{P}$.

Theorem 0.15 (Zachos)**.** *If* NP $\subseteq$ BPP *then* PH $\subseteq$ BPP.

Proof of Lemma 0.10. We relative Corollary 0.13 and apply Theorem 0.14 to get

$$\mathrm{NP}^{\oplus \mathrm{P}} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}^{\oplus \mathrm{P}}} = \mathrm{BPP}^{\oplus \mathrm{P}}$$

Then by relativizing Theorem 0.15 we get

It is critical here that the proof of Valiant-Vazirani relativizies. To properly verify such a claim, one would have to go back through the proof and ensure all claims still hold if the TMs in question have access to a $\oplus P$ oracle.

$$\mathrm{NP}^{\oplus \mathrm{P}} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}} \Rightarrow \mathrm{PH}^{\oplus \mathrm{P}} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}}$$

and hence combining with above we have

$$\mathrm{PH}^{\oplus} \subseteq \mathrm{BPP}^{\oplus}$$

which shows that $\mathrm{PH} \subseteq \mathrm{BPP}^{\oplus \mathrm{P}}$. □

Thus it remains to prove Lemma 0.11. We will need to following definition.

The class GapP is precisely the closure of #P under subtraction

Definition 0.16 (GapP)**.** A function $f : \Sigma^* \to \Sigma^*$ is in GapP if there is a polynomial-time NDTM M such that for any $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths minus the number of rejecting computation paths of M on x.

The following lemma gives a useful GapP characterization of $\oplus P$.

Lemma 0.17 (Fenner-Fortnow-Kurtz)**.** *A language L is in $\oplus P$ iff there is a Gap*P *function f satisfying*

- *if $x \in L$ then $f(x)$ is odd*
- *if $x \notin L$ then $f(x)$ is even.*

We can also define PP^A using P^A predicates.

Lemma 0.18. *A language L is in PP^A iff there is a language $L' \in \mathrm{P}^A$ and a polynomial p such that*

- *if $x \in L$ then*

$$\left|\left\{y \in \Sigma^{p(|x|)} : (x,y) \in L'\right\}\right| \geq \left|\left\{y \in \Sigma^{p(|x|)} : (x,y) \notin L'\right\}\right|$$

- *if $x \notin L$ then*

$$\left|\left\{y \in \Sigma^{p(|x|)} : (x,y) \in L'\right\}\right| < \left|\left\{y \in \Sigma^{p(|x|)} : (x,y) \notin L'\right\}\right|.$$

Then combining the above two lemmas with the fact that $\oplus\mathrm{P}^{\oplus\mathrm{P}} = \oplus\mathrm{P}$ (and hence $\mathrm{P}^{\oplus\mathrm{P}} = \oplus\mathrm{P}$) we have that

Corollary 0.19. *A language L is in $\mathrm{PP}^{\oplus\mathrm{P}}$ iff there is a Gap$\mathrm{P}$ function f and a polynomial p such that*

- *if $x \in L$ then*

$$\left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is odd}\right\}\right| \geq \left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is even}\right\}\right|$$

- *if $x \notin L$ then*

$$\left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is odd}\right\}\right| < \left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is even}\right\}\right|$$

We proceed by giving an $\mathrm{FP}^{\mathrm{GapP}}$ algorithm to compute

$$\left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is odd}\right\}\right|$$

and

$$\left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is even}\right\}\right|.$$

The result then follows since $\mathrm{FP}^{\mathrm{GapP}} = \mathrm{FP}^{\#\mathrm{P}}$ and we can simulate an $\mathrm{FP}^{\#\mathrm{P}}$ algorithm in $\mathrm{P}^{\#\mathrm{P}}$.

Proof of Lemma 0.11. Consider the polynomial $g(m) = 3m^2 - 2m^3$. Let $g^k(m)$ denote the composition of g with itself k times. Then algebra and induction show

1. if $m \equiv 0 \pmod{2^j}$ then $g(m) \equiv 0 \pmod{2^{2j}}$
2. if $m \equiv 1 \pmod{2^j}$ then $g(m) \equiv 1 \pmod{2^{2j}}$
3. if $m \equiv 0 \pmod{2}$ then $g^k(m) \equiv 0 \pmod{2^{2^k}}$
4. if $m \equiv 1 \pmod{2}$ then $g^k(m) \equiv 1 \pmod{2^{2^k}}$.

Let $h(x,y) = g^{1+\lceil \log p(|x|)\rceil}(f(x,y))$. Then h is a GapP function. By the above we have

Here, we use the fact that GapP functions are closed under exponential-size sums and polynomial-size products.

- if $f(x,y) \equiv 1 \pmod 2$ then $h(x,y) \equiv 1 \pmod{2^{p(|x|)+1}}$ and
- if $f(x,y) \equiv 0 \pmod 2$ then $h(x,y) \equiv 0 \pmod{2^{p(|x|)+1}}$.

Define

$$r(x) = \sum_{y \in \Sigma^{p(|x|)}} h(x,y)$$

which is also a GapP function. We have

$$r(x) \pmod{2^{p(|x|)+1}} = \left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is odd}\right\}\right|$$

and

$$2^{p(|x|)} - r(x) \pmod{2^{p(|x|)+1}} = \left|\left\{y \in \Sigma^{p(|x|)} : f(x,y) \text{ is even}\right\}\right|$$

□

Closure properties of PP

The class PP was first introduced by Gill in 1977 [Gil77] who conjectured that the class was closed under complement and intersection. However, it remained unknown whether this was the case until 1991 when three researchers affirmed the conjecture [BRS95].

Theorem 0.20 (PP is closed under complement)**.** *Let* $L \in$ PP *then* $\overline{L} \in$ PP *where* $\overline{L}$ *is the complement of* L*.*

Proof. Let M be a probabilistic polynomial-time TM that decides L such that $x \in L \Rightarrow P(M \text{ accepts } x) > 1/2$ and $x \notin L \Rightarrow P(M \text{ accepts } x) \leq 1/2$. We define a new probabilistic polynomial-time TM M' such that $x \in L \Rightarrow P(M' \text{ accepts } x) > 1/2$ and $x \notin L \Rightarrow P(M' \text{ accepts } x) < 1/2$.

Suppose $x \in L$. Let $P(M \text{ accepts } x) = 1/2 + \varepsilon$ for $\varepsilon > 0$. The key insight is that we can lower bound ε by exploiting the fact that the probabilities of accepting/rejecting are discrete. Specifically let p be the polynomial that bounds the computation time of M. Hence $p(|x|)$ is an upper bound on the total number of coin tosses M can make on x. In particular, $1/2^{p(|x|)}$ is a lower bound on the "granularity" of the probability that M accepts.[1] Thus, we have $P(M \text{ accepts } x) \geq 1/2 + 1/2^{p(|x|)}$.

[1] By "granularity" we mean that the true probability is going to be a multiple of something at least as big as $1/2^{p(|x|)}$ and hence the true probability will be a multiple of $1/2^{p(|x|)}$.

With this new bound we define $M'(x)$ as follows:

Suppose $x \notin L$ then

$$\begin{aligned} P(M' \text{ accepts } x) &\leq \underbrace{\frac{1}{2}}_{P(M \text{ accepts})} \cdot \underbrace{\left(1 - \frac{1}{2^{p(|x|)+1}}\right)}_{P(y \neq 0^{p(|x|)+1})} \\ &< \frac{1}{2} \end{aligned}$$

1: Run $M(x)$
2: **if** $M(x) = 0$ **then**
3: **reject**
4: **else**
5: Guess a string $y \in \Sigma^{p(|x|)+1}$
6: **if** y is all 0s **then reject**
7: **else** accept
8: **end if**

Conversely if $x \in L$ then

$$\begin{aligned} P(M' \text{ accepts } x) &\geq \left(\frac{1}{2} + \frac{1}{2^{p(|x|)}}\right)\left(1 - \frac{1}{2^{p(|x|)+1}}\right) \\ &= \frac{1}{2} + \frac{1}{2^{p(|x|)}} - \frac{1}{2^{p(|x|)+2}} - \frac{1}{2^{2p(|x|)+1}} \\ &> \frac{1}{2} \end{aligned}$$

which follows since

$$\begin{aligned} 1 &> \frac{1}{4} + \frac{1}{2^{p(|x|)+1}} \\ \frac{1}{2^{p(|x|)}} &> \frac{1}{2^{p(|x|)+2}} + \frac{1}{2^{2p(|x|)+1}} \\ \frac{1}{2^{p(|x|)}} - \frac{1}{2^{p(|x|)+2}} - \frac{1}{2^{2p(|x|)+1}} &> 0. \end{aligned}$$

Let M'' be the probabilistic polynomial time TM that runs M' but reverts the accept/reject decision (i.e. if M' accepts then M'' rejects and vice versa). Then M'' satisfies $x \in \overline{L} \Rightarrow P(M' \text{ accepts } x) > 1/2$ and $x \notin \overline{L} \Rightarrow P(M' \text{ accepts } x) < 1/2$. It follows that $\overline{L}$ is in PP. □

The proof that PP is closed under intersection and union is slightly more involved. We follow the exposition given by Fortnow in [For97]. The main idea is to approximate the sign function with some rational functions.[2] Let m be a positive even integer. Define

$$\begin{aligned} P_m(z) &= (z-1)\prod_{i=1}^{m}(z-2^i)^2 \\ A_m(z) &= (P_m(-z))^{m/2+1} - (P_m(z))^{m/2+1} \\ B_m(z) &= (P_m(-z))^{m/2+1} + (P_m(z))^{m/2+1} \\ S_m(z) &= \frac{A_m(z)}{B_m(z)}. \end{aligned}$$

[2] The sign function is the function $\text{sign}(x)$ is 1 if $x > 0$, 0 if $x = 0$, and -1 if $x < 0$. A rational function is a quotient of two polynomials.

We also use the following classification of PP:

Definition 0.21 (GapP classification of PP)**.** A language L is in PP if there is a function $f \in \text{GapP}$ such that $x \in L \Rightarrow f(x) > 0$ and $x \notin L \Rightarrow f(x) < 0$.

Lemma 0.22 ([BRS95]). *The following results hold with respect to the above functions*

- *If* $1 \leq z \leq 2^m$ *then* $1 \leq S_m(z) < 1 + 1/2^m$.
- *If* $-2^m \leq z \leq -1$ *then* $-1 - 2^{-m} < S_m(z) \leq -1$.

We are now in a position to prove the main theorem.

Theorem 0.23. *Let* $D, E \in$ PP, *then* $D \cup E \in$ PP.

Proof. Let D, E be as in the theorem statement. Let f_D, f_E be the functions from the GapP definition of PP. Let $m = m(|x|)$ be a polynomial with even coefficients such that $2^{m(|x|)}$ bounds the maximum absolute value of $f_D(x)$ and $f_E(x)$. Define $A_D(x) = A_m(f_D(x))$. Similarly define B_D, S_D, A_E, B_E and S_E. The functions A_D, B_D, A_E, B_E are GapP functions. Let $H = S_D + S_E + 1$. By Lemma 0.22 we have

- if $x \in D, E$ then $H(x) \geq 3$
- if $x \in D$, $x \notin E$ then $H(x) \geq 1 - 2^{-m}$
- if $x \notin D$, $x \in E$ then $H(x) \geq 1 - 2^{-m}$
- if $x \notin D, E$ then $H(x) \leq -1$.

Thus we have $x \in D \cup E$ iff $H(x) > 0$. This would complete the proof if H were a GapP but H may have nonintegral values. So we have

$$H = \frac{A_D}{B_D} + \frac{A_E}{B_E} + 1 = \frac{A_D B_E + A_E B_D + B_D B_E}{B_E B_D}.$$

If p, q are nonzero integers and $p/q > 0$ then $pq > 0$ and hence we define

$$H' = (A_D B_E + A_E B_D + B_D B_E)(B_E B_D).$$

Hence we have

- H' is a GapP function
- For all $x \in \Sigma^*$, $H(x) > 0$ iff $H'(x) > 0$.
- $x \in D \cup E$ iff $H'(x) > 0$.

Thus by the GapP characterization of PP we have that $D \cup E$ is in PP. □

Note that this relies on the construction to show that PP is closed under intersection. In particular, we can simply take f to be the number of accepting paths minus the number of rejecting paths.

Interactive Proofs

Topics we cover

- Definition of interactive proof system, dIP, IP
- Graph nonisomorphism and quadratic nonresiduosity is in IP
- IP = PSPACE

Questions from Last Time

Question 0.24. *What was the motivation for defining* PP*?*

In [Gil77], Gill was inspired by recent results giving fast probabilistic algorithms for problems that have no fast deterministic solution. The motivation for PP was thus partially to define a unifying model of computation which seemed to capture these properties.

Question 0.25. *If L is* #P*-complete under parsimonious reductions, then the corresponding majority problem is* PP*-complete.*

In general, this may not be the case. The reason is that parsimonious reductions may not preserve the majority. For example, #3SAT is #P-complete under parsimonious reductions - as can be shown by reducing from #SAT. However in performing this reduction, new variables are introduced that increase the total number of assignments. Hence the majority may not be preserved.

Definitions

An interactive proof system is, informally, a model of computation that captures an interaction between two parties where one party wants to be convinced of some statement while the other party provides responses to the first party's queries. The party that wants to be convinced of some statement is the *verifier* while the other party is called the *prover*. The verifier and the prover pass strings between each other until the verifier decides whether it will accept the given input or not. A language $L \subseteq \Sigma^*$ is recognized by an interactive proof system if there is a verifier that satisfies the completeness and soundness properties:

- **Completeness**: If $x \in L$ then there is a prover P which following the protocol should be able to satisfactorily convince V that $x \in L$
- **Soundness**: If $x \notin L$ then there is no prover that can convince V that $x \in L$.

Observe that in this definition there is no limit on the computational power of the prover g. We want this property since intuitively a false assertion should by unprovable regardless of how smart the prover may be.

Some of the most well known classes that originate from interactive proof systems are IP, MA, AM, MIP, and PCP$(r(n), q(n))$. Here, we focus on the class IP.

Given the notion of an interactive proof system, one may define a protocol as follows. Let $f, g : \Sigma^* \to \Sigma^*$ be functions and $k \geq 0$ an integer. Then a k-round interaction on input $x \in \Sigma^*$ is a sequence of strings $a_1, ..., a_k \in \Sigma^*$ such that $a_1 = g(x), a_2 = f(x, a_1), ...$ and the final output $f(x, a_1, \ldots, a_k)$ is a bit in $\{0, 1\}$. Let out$(f, g)(x)$ denote this output bit. Then we can define

Definition 0.26 (dIP [AB09])**.** A language L has a deterministic interactive proof (dIP) there is a polynomial p and a DTM V such that on input $x, a_1, ..., a_{p(|x|)}$ runs in polynomial time and satisfies soundness and completeness after a $p(|x|)$-round interaction with a prover:

Completeness $x \in L \Rightarrow \exists g : \Sigma^* \to \Sigma^*$ such that $\text{out}(V, g)(x) = 1$

Soundness $x \notin L \Rightarrow \forall g : \Sigma^* \to \Sigma^*$ we have $\text{out}(V, g)(x) = 0$.

This definition has similarities with the certificate/verifier definition of NP. We can think of the prover g as providing a certificate for membership to a particular NP problem and the verifier V as the polynomial-time DTM that verifies the certificate. In fact, it is essentially this line of reasoning that shows the two classes are equivalent.

Proposition 0.27. NP = dIP.

Proof. By the above comment, every $L \in$ NP has a 1-round interaction protocol - simply have the prover provide a certificate if $x \in L$ and have V verify it. Otherwise if $x \notin L$ no such certificate can be produced. For the other direction, suppose that $L \in$ dIP. Then a certificate for $x \in L$ is simply a transcript of messages $(a_1, ..., a_k)$ which is polynomail in $|x|$ and can be verified in polynomial time. □

The moral of this example is that deterministic interactive proof systems are not terribly interesting. A natural step then is to allow the verifier to access random bits.

The class IP was first introduced in [GMR89] for cryptographic applications.

Definition 0.28 (IP [AB09])**.** For an integer $k \geq 0$ (or a function on the size of input strings), a language L is in IP$[k]$ if there is a probabilistic polynomial-time TM V that can have a k-round interaction with a function $g : \Sigma^* \to \Sigma^*$ such that

Completeness $x \in L \Rightarrow \exists g : \Sigma^* \to \Sigma^*$ such that $P(\text{out}(V, g)(x) = 1) \geq 2/3$

Soundness $x \notin L \Rightarrow \forall g : \Sigma^* \to \Sigma^*$ we have $P(\text{out}(V, g)(x) = 1) \leq 1/3$.

We then define IP $= \cup_{c \geq 0}$IP$[n^c]$.

We observe then that NP $\subseteq$ IP$[1]$ and BPP = IP$[0]$ (since a verifier is itself a polynomial-time DTM).

We make some observations about the class IP.

- The prover is required to be deterministic, but the class isn't changed by allowing it to also access random bits. In particular, if a prover uses random bits and makes V accept with some probability then there must be some sequence of random bits that we can fix to make V accept with the same probability.
- Using the same probability amplification as with BPP one can replace the completeness parameter with $1 - 2^{-p(n)}$ and the soundness parameter by $2^{-p(n)}$ for any polynomial p.

- The prover does not have access to the random bits used by the verifier. Thus, this type of protocol is called a *private coin* interaction proof. Alternatively one can consider *public coin* interaction proofs which lead to the classes MA and AM.

Problems in IP

Graph nonisomorphism

Two undirected graphs $G = (V, E), G' = (V', E')$ are isomorphic if one can relabel the vertices of one to get the other. The class GI $= \{\langle G, G' \rangle : G \text{ and } G' \text{ are isomorphic}\}$, the class of isomorphic graphs, is in NP since we can use the isomorphism between G and G' as a certificate. It is unknown whether GI is NP-hard, but there are reasons to believe it may be an NP-intermediate problem. Let $\overline{\text{GI}}$ be the complement of GI. Then $\overline{\text{GI}} \in$ IP. The following is a protocol in IP[2].

The first interactive proof for graph nonisomorphism was introduced in [GMW86], again for cryptographic reasons.

Require: Two undirected graphs G_0, G_1

V: Randomly pick $i \in \{0, 1\}$. Pick a random permutation of the vertices of G_i and send H to prover, P.

P: Identify which of G_1, G_2 was picked out by i to produce H. If G_j is the identified graph, send j to verifier.

V: Accept if $i = j$ and reject otherwise.

Note that the verifier runs in polynomial time in the length of $\langle G_0, G_1 \rangle$ and so in order to show this procedure puts $\overline{\text{GI}}$ in IP we need to analyze the probability of acceptance. If $\langle G_0, G_1 \rangle \in \overline{\text{GI}}$ then there exists a prover that can, when given H, simply check which of G_0, G_1, H is isomorphic to. This check will always yield one result because G_0, G_1 are not isomorphic. Hence $P(V \text{ accepts } \langle G_0, G_1 \rangle) = 1$. However, suppose $\langle G_0, G_1 \rangle \notin \overline{\text{GI}}$. Then H will always be isomorphic to both G_0, G_1. Then the best the prover can do is guess because a random permutation of G_0 looks exactly like a random permutation of G_1. Hence regardless of the prover we have $P(V \text{ accepts } \langle G_0, G_1 \rangle) \leq 1/2$. Repeating this interaction twice (i.e. picking a random bit twice and only accepting if the prover guesses H correctly twice) gets the probability bound down to $1/4$ so we have

$$\langle G_0, G_1 \rangle \in \overline{\text{GI}} \Rightarrow \exists P : P(V \text{ accepts } \langle G_0, G_1 \rangle) = 1$$
$$\langle G_0, G_1 \rangle \notin \overline{\text{GI}} \Rightarrow \forall P : P(V \text{ accepts } \langle G_0, G_1 \rangle) \leq 1/4.$$

Thus $\overline{\text{GI}} \in \text{IP}[2] \subseteq \text{IP}$.

Since it is widely believed that $\overline{\text{GI}}$ is not in NP, this interactive proof gave initial evidence that IP is stronger than NP.

Quadratic nonresiduosity

A similar technique as in the protocol for graph nonisomorphism can be used to show that quadratic nonresiduosity is in IP. A num-

ber a is a quadratic residue modulo p if there is a b such that $a \equiv b^2 \pmod{p}$. The number b is the square root of a. The language $\text{QR} = \{(a,p) : a \text{ is a quadratic residue modulo a prime } p\}$ is in NP since the square root of a can serve as a certificate that $(a,p) \in \text{QR}$. Conversely the language $\overline{\text{QR}} = \{(a,p) : a \text{ is a quadratic nonresidue modulo a prime } p\}$ is not known to be in NP. However, it can be shown to be in IP via the following protocol

---

Require: Two integers (a,p)
V: Randomly pick $r \in \{0,1,\ldots,p-1\}$.
Pick a random bit $b \in \{0,1\}$. If $b = 0$, send $r^2 \pmod{p}$ otherwise send $ar^2 \pmod{p}$
P: Identify whether $b = 0$ or $b = 1$, send answer to V.
V: Accept if P correctly identifies b and reject otherwise.

To show this protocol is in IP first assume that $(a,p) \in \overline{\text{QR}}$. Then note that ar^2 is a quadratic nonresidue since if $ar^2 \equiv b^2 \pmod{p}$ then $a \equiv (br^{-1})^2 \pmod{p}$ is a quadratic residue - a contradiction (note we use here that r is a unit since $\mathbb{Z}_p$ is a field as p is prime). In contrast, r^2 is always a quadratic residue with r as the square root. Hence the prover which determines whether the number sent by the verifier is a quadratic residue will make the verifier accept with probability 1. Conversely if a is a quadratic residue then

$$\{ar^2 \pmod{p} : r \in \{0,\ldots,p-1\}\} = \{r^2 \pmod{p} : r \in \{0,\ldots,p-1\}\}$$

since if c is a quadratic residue then $c = as^2$ where $s^2 \equiv ca^{-1} \pmod{p}$, as $ca-1$ is also a quadratic residue since c and a^{-1} are. Hence the best the prover can do is randomly guess whether $b = 0$ or $b = 1$ which shows that the verifier accepts with probability at most $1/2$. Thus, by repeating this procedure twice as in the graph nonisomorphism case, we get that $\overline{\text{QR}} \in \text{IP}$.

IP = PSPACE

We break the result into two lemmas: first showing that IP $\subseteq$ PSPACE then showing that IP $\supseteq$ PSPACE.

Lemma 0.29. IP $\subseteq$ PSPACE

Proof. Let $L \in$ IP. Given a particular prover P and verifier V, we write $P(P \leftrightarrow V \text{ accepts } x)$ for the probability that V accepts x after following a protocol for communicating with P. Our strategy will be to show that

$$m = \max_P P(P \leftrightarrow V \text{ accepts } x)$$

is computable in PSPACE. Then we can simply check if $m \geq 2/3$ in which case we accept x and reject otherwise. Let m_k denote the kth message (sent by either the verifier or the prover) and let M_j denote the message history after j rounds of interaction (the message history is simply the sequence of strings sent by the prover and the verifier to each other during the execution of the protocol). Note that m_1 is a message sent to the prover from the verifier and hence in general if k is odd then m_k is a message from the verifier to the prover. Let $m_{p(|x|)}$ denote the last message where p is the polynomial bounding the number of interactions and without loss of generality assume that if $M_{p(|x|)} = 0$ then V rejects x and if $M_{p(|x|)} = 1$ then V accepts.

Then consider the following recursive function

$$N(M_j) = \begin{cases} 0 & j = p(|x|), m_{p(|x|)} = 0 \\ 1 & j = p(|x|), m_{p(|x|)} = 1 \\ \max_{m_{j+1}} N(M_{j+1}) & j < p(|x|), j \text{ is odd} \\ \text{avg}_{m_{j+1}} N(M_{j+1}) & j < p(|x|), j \text{ is even} \end{cases}$$

where

$$\text{avg}_{m_{j+1}} N(M_{j+1}) \stackrel{\text{def}}{=} \sum_{m_{j+1}} N(M_{j+1}) P(V \text{ sends } m_{j+1} \text{ after } M_j)$$

which is the average of $N(M_{j+1})$ weighted by the probability that M_{j+1} actually occurred, i.e. that V actually sent m_{j+1}. Note here that the probability is taken over all the random bits that V may access when deciding what to send.

Note then that a recursive algorithm can compute $N(M_0)$ in PSPACE since the depth of recursion is $p(|x|)$, a polynomial in $|x|$ (and both $\max_{m_{j+1}} N(M_{j+1})$ and $\text{avg}_{m_{j+1}} N(M_{j+1})$ can be computed in PSPACE). Moreover induction shows that $N(M_j) = \max_P P(P \leftrightarrow V \text{ accepts } x \text{ starting at } M_j)$. Thus $N(M_0)$ gives us the desired probability which we can check to determine whether $x \in L$ or not. □

The proof that PSPACE $\subseteq$ IP took a bit longer to be developed. The first major development came in the form of showing that $\#\text{SAT}_D = \{\langle \varphi, k \rangle : \varphi \text{ has exactly } k \text{ satisfying assignments}\}$ is in IP. Shortly after, it was shown that the language TQBF $= \{\langle F \rangle : F \text{ is satisfiable and } F = Q_1 x_1 \cdots Q_n x_n \varphi(x_1, \ldots, x_n), Q_i \in \{\exists, \forall\}\}$ is in IP.[3] The result then follows since TQBF is PSPACE-complete.

[3] TQBF stands for true quantified Boolean formulas.

Proposition 0.30 ([AB09]). $\#\text{SAT}_D \in \text{IP}$.

This result shows that coNP is in IP since it shows $\overline{\text{SAT}}$ is in IP.

Proof. The main idea is to represent Boolean formulas as polynomials. Specifically suppose that φ is a Boolean formula in CNF with variables $x_1, \ldots, x_n$ and $C_j = \ell_1 \vee \cdots \vee \ell_k$ is a clause. Then we define

$p_j(X_1, \ldots, X_n) = 1 - L_1 \cdots L_k$ where $L_i = 1 - X_i$ if $\ell_i = x_i$ and $L_i = X_i$ if $\ell_i = \neg x_i$. So for example if $C_j = x_1 \vee \neg x_2 \vee \neg x_3$ then $p_j(X_1, \ldots, X_n) = 1 - (1 - X_1)X_2X_3$. We can represent any assignment of truth values to the variables of φ as a bit-vector $b \in \{0,1\}^n$ where $b_i = 0$ if x_i is assigned false and $b_i = 1$ otherwise. Then we have that $p_j(b) = 1$ iff the clause C_j is satisfied (otherwise $p_j(b)$ is 1). We then define

$$P_\varphi(X_1, \ldots, X_n) = \prod_{j \leq m} p_j(X_1, \ldots, X_n)$$

where m is the number of clauses in φ. Then note that P_φ evaluates to 1 on satisfying assignments and 0 otherwise. Note also that the degree of P_φ is at most mn and in order to write down P_φ we need only $O(mn)$ space. The function P_φ is called the arithmetization of φ.

Now let $\langle \varphi, k \rangle$ be a potential instance of $\#\mathrm{SAT}_D$ and let P_φ be as above. Then the number of satisfying assignments is given by

We abuse notation throughout the proof by writing $P_\varphi(b)$ where b is a vector instead of $P_\varphi(b_1, \ldots, b_n)$.

$$\#\varphi = \sum_{b \in \{0,1\}^n} P_\varphi(b).$$

Let p be a prime in $\{2^n + 1, \ldots, 2^{2n}\}$. Then observe that the above $\#\varphi \leq 2^n$ and hence the above is equal to k iff it is equal to k modulo p. For the rest of the proof then we work over the field $\mathbb{F}_p$ and give an IP protocol for determining the value of $\#\varphi$.

Consider the following univariate polynomial

$$h(X_1) = \sum_{b \in \{0,1\}^{n-1}} P_\varphi(X_1, b).$$

Note that if the boolean formula φ does in fact have k satisfying assignments then $h(0) + h(1) = k$. Inspired by this, we have the following protocol:

Require: An integer k and a polynomial $g(X_1, ..., X_n)$.

V: If $n = 1$, check that $g(0) + g(1) = k$. If yes, then accept; if no, then reject. If $n \geq 2$, ask P to send an expanded form (modulo p) of

$$h(X_1) = \sum_{b \in \{0,1\}^{n-1}} g(X_1, b).$$

P: Sends some polynomial $f(X_1)$.

V: If f and h have different degrees then reject. Check if $f(0) + f(1) \neq k$. If yes, then reject. Otherwise pick a random number r in $\mathbb{F}_p$ and recursively repeat this procedure with the integer $f(r)$ and the function $g'(X_2, \ldots, X_n) = g(r, ..., X_n)$.

The protocol starts with $g = P_\varphi$. We claim that if $\sum_{b \in \{0,1\}^n} g(b) \neq k$ in the above procedure then the probability that V rejects is at least

$(1 - d/p)^n$ where $d = mn$ is the degree of P_φ regardless of the prover behavior. Our proof is by induction on n. If $n = 1$, then we directly check that $g(0) + g(1) = k$ and reject with probability 1. Otherwise, assume the hypothesis is true for degree d polynomials in $n - 1$ variables. Then in the first round if P sends the correct polynomial f such that $f = h$, then the evaluation $f(0) + f(1) = h(0) + h(1) \neq k$ yields a yes response (since by assumption $h(0) + h(1) \neq k$) and thus V rejects, with probability 1. Otherwise, P returns something different from h. Degree d polynomials f and h agree on at most d values as $f - h$ has at most d roots. Thus $P(f(r) = h(r)) \leq d/p$ since $p = |\mathbb{F}_p|$. Hence $P(f(r) \neq h(r)) \geq 1 - d/p$. If $f(r) \neq h(r)$ then in the recursive step, the verifier is left with an incorrect claim and by induction rejects with probability at least $(1 - d/p)^{n-1}$. Hence overall we have inductively,

$$P(V \text{ rejects}) \geq \left(1 - \frac{d}{p}\right)\left(1 - \frac{d}{p}\right)^{n-1} = \left(1 - \frac{d}{p}\right)^n.$$

This completes the inductive proof. Notice then that $p \geq 2^n$ and WLOG assume that the formula φ is in 3-SAT form so that $d \leq n^3$. Then we have

$$\left(1 - \frac{d}{p}\right)^n \geq \left(1 - \frac{n^3}{2^n}\right)^n.$$

Notice then that by making p be a large enough prime, we can force $1 - \frac{n^3}{2^n}$ to be arbitrarily close to 1 so that the verifier rejects with a probability high enough to satisfy the definition of IP.

Hence, it remains to consider the case when $k = \sum_{b \in \{0,1\}^n} g(b)$ is true. In this case, we can pick a prover that simply always returns the correct function h. Then, V will accept with probability 1.

□

Using a similar technique we can show that TQBF $\in$ IP.

We present a simplified proof due to [She92] and follow the exposition from [ABo09].

Proposition 0.31 ([Sha90])**.** TQBF $\in$ IP.

Proof. Let $\Phi = \forall x_1 \exists x_2 \cdots \exists x_n \varphi(x_1, \ldots, x_n)$. We use arithmetization to construct a polynomial P_φ such that

$$\prod_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\varphi(b_1, \ldots, b_n) \neq 0$$

iff $\Phi \in$ TQBF. We may try to use the same protocol as in the case of #SAT, however due to the $\prod$ terms, the degrees of the polynomial h may become too large. So instead we define the following operators on polynomials:

The L_i operator is known as a linearization operator and is linear in X_i. It agrees with p whenever $X_i = 0, 1$.

$$\begin{aligned}
L_i(p) =& X_i p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n) + \\
& (1 - X_i) p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) \\
\exists_i(p) =& p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) + \\
& p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n) \\
\forall_i(p) =& p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) \cdot \\
& p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n).
\end{aligned}$$

Thus we have $\Phi \in$ TQBF iff $\forall_1 \exists_2 \cdots \exists_n P_\varphi \neq 0$. Since the claim only concerns values in $\{0, 1\}$ we can add linearization operators as we please. We do so to ensure that the polynomials arising in the interactive proof all have low degree. In particular we are interested in the expression

$$\forall_1 L_1 \exists_2 L_1 L_2 \cdots \exists_n L_1 L_2 L_3 \cdots L_n P_\varphi.$$

From this representation, we can give an inductive definition of the protocol. Suppose that we start with a function $g(X_1, ..., X_k)$ such that whenever there are are values $a_1, ..., a_k$ and C, with $g(a_1, ..., a_k) = C$ then there is a prover that convinces the verifier of this fact with probability 1 and otherwise if $g(a_1, ..., a_k) \neq C$, for all provers, the verifier accepts it (falsely) as true with probability less than ε. Let $U(X_1, ..., X_j)$ be the polynomial obtained as

$$U(X_1, ..., X_j) = O(g)$$

where $O \in \{\exists_i, \forall_i, L_i\}$ is some operator for X_i. Let d be an upper bound on the degree of U with respect to X_i. We demonstrate that a prover can convince the verifier with probability 1 that $U(a_1, ..., a_j) = C'$ if it is true for any $a_1, ..., a_k$ and C' and that if it is false, the probability of acceptance is bounded above by $\varepsilon + d/p$ where p is a prime selected as in the $\#\text{SAT}_D$ protocol. We now break down into cases depending on what O is. Without loss of generality, assume that $i = 1$.

1. $O = \exists_1$. The prover is asked to provide a degree d polynomial $f(X_1)$ that is supposed to be $g(X_1, a_2, ..., a_k)$. The verifier checks if $f(0) + f(1) \neq C'$. If yes, it rejects; otherwise randomly picks $r \in \mathbb{F}_p$ and recurses with the new integer $f(r)$ and the function $g'(a_2, \ldots, a_k) = g(r, a_2, \ldots, a_k)$.

2. $O = \forall_1$. Same as above but with $f(0)f(1)$ instead of $f(0) + f(1)$.

3. $O = L_1$. The verifier asks for a degree d polynomial $f(X_1)$ that is equal to $g(X_1, a_2, \ldots, a_k)$. Verifier checks if $a_1 f(0) + (1 - a_1) f(1) \neq C'$. If yes then reject. Otherwise, pick a random $r \in \mathbb{F}_p$ and recurse with $f(r)$ and $g'(a_2, \ldots, a_k) = g(r, a_2, \ldots, a_k)$.

The same analysis of correctness as in the proof that $\#\text{SAT}_D \in \text{IP}$ shows that this protocol is in IP. In particular if $f(X_1)$ is not the polynomial that is asked for then with probability $(1 - d/p)$ the prover is left with an incorrect statement in the next recursive call.

□

Probabilistically checkable proofs

Probabilistically checkable proofs are similar to interactive proofs except that a true input is required to be accepted with probability 1. Specifically L has *probabilistically checkable proofs* if there is an interactive protocol (P, V) such that V is a probabilistic polynomial-time TM and P is a prover and

- $x \in L \Rightarrow (P, V)$ accepts with probability 1; and
- if $x \notin L \Rightarrow \forall P', (P', V)$ accepts with probability at most $1/2$

and the prover is not allowed to modify its answers based on previous queries. We can without loss of generality assume that the verifier only queries for a single bit at a time from the prover P.[4] Hence, the verifier only sends binary strings of polynomial length to receive some response from the prover P. Thus, if the prover isn't allowed to modify its strategy based on the queries from V, then we can regard the prover as a single exponentially long proof string and each query from V as a request to receive a single bit from that string.

We can also place bounds on how many random bits V uses and the number of queries it makes to the proof string. We thus have the following definition

Definition 0.32 (PCP, PCP verifier [Koz06, AB09])**.** A language L with probabilistically checkable proofs has a $(r(n), q(n))$-PCP verifier if there is a verifier V for the language that satisfies *efficiency*: on input $x \in \Sigma^n$ and given access to a proof string π of length at most $q(n)2^{r(n)}$, V uses at most $r(n)$ random bits and makes at most $q(n)$ queries (each which is of the form π_i, the ith bit of π).

A language L is in $\text{PCP}(r(n), q(n))$ if there are constants $c, d > 0$ such that L has a $(cr(n), dq(n))$-PCP verifier.

As an example, we show that graph nonisomorphism, $\overline{\text{GI}}$ is in $\text{PCP}(n, 1)$.

Proposition 0.33 ([AB09])**.** $\overline{\text{GI}} \in \text{PCP}(n, 1)$

Proof. Let $\langle G_0, G_1 \rangle$ be two graphs each with n vertices. We suggest the following protocol to determine whether $\langle G_0, G_1 \rangle \in \overline{\text{GI}}$. The verifier expects a proof string π, where index i represents the ith n-vertex graph.[5] If H is isomorphic to the graph G_k ($k \in \{0, 1\}$) then

[4] This statement is without loss of generality since for any polynomial length answer from P, it only costs a polynomial number of steps to make a polynomial number of queries to extract the answer one bit at a time.

[5] We can think of an n-vertex graph as the bit string generated by the entries of its adjacency matrix representation.

$\pi[H] = k$, and otherwise $\pi[H]$ is arbitrary. Note then that $|\pi| = 2^{n^2}$. The verifier operates by picking a random $r \in \{0,1\}$ and then generates a random permutation of the vertices of G_r to get a graph H. The verifier then queries $\pi[H]$ and accepts iff $\pi[H] = r$. If the graphs are in fact nonisomorphic then H will always be isomorphic to only G_b and hence the verifier will always accept. Otherwise, if G_0, G_1 are isomorphic then regardless of what the proof string π is, as r is picked randomly, the probability the verifier accepts is at most $1/2$. □

Note that $\mathrm{PCP}(r(n), q(n)) \subseteq \mathrm{NTIME}(2^{O(r(n))}q(n))$. This is because we can without loss of generality assume that the proof has length at most $q(n)2^{r(n)}$ since the verifier can look at $\leq q$ bits for each possible $r(n)$-long sequence of coin tosses. And since there are $2^{r(n)}$ such sequences of coin tosses there are at most $q(n)2^{r(n)}$ bits the verifier could look at (hence we can delete bits that are looked at with probability 0). It follows that a nondeterministic TM could guess a proof in time $2^{O(r(n))}q(n)$ and verify it by simulating the verifier on all possible $2^{O(r(n))}$ sequences of coin tosses and accepting iff the verifier accepts in all sequences of coin tosses. Hence as a special case we have $\mathrm{PCP}(\log n, 1) \subseteq \mathrm{NTIME}(2^{O(\log n)}) = \mathrm{NP}$.

In fact the reverse inclusion also holds, but requires a more involved proof. We thus have the following theorem, which is perhaps the most prominent in the area of probabilistic checkable proofs.

Theorem 0.34 (The PCP theorem [AS98, ALM+98])**.**
$\mathrm{NP} = \mathrm{PCP}(\log n, 1)$.

We start by proving a weaker version of the theorem which utilizes a number of ideas similar to those used in the full proof.

Theorem 0.35 (Weak PCP Theorem [Koz06])**.** $\mathrm{NP} = \mathrm{PCP}(n^3, 1)$.

There are four main ideas that go into the proof.

1. Arithmetization of Boolean formulas over $\mathbb{F}_2$ (as in e.g. the proof that IP = PSPACE).
2. Probabilistic tests for whether a vector v is 0.
3. Probabilistic tests for whether $f : \{0,1\}^n \to \{0,1\}$ is linear
4. Random self-correction for linearity testing.

Arithmetization

Given a boolean formula φ in 3CNF form with m clauses over variables $x = (x_1, \ldots, x_n)$, we create a vector of polynomials $p_i \in \mathbb{Z}_2[x]$. For the ith clause we define the ith vector to be the product of linear polynomials of the form x_i if $\neg x_i$ is in the clause and $(1 - x_i)$

if x_i is in the clause. So for example $(x_1 \vee \neg x_2 \vee \neg x_3)$ translates to $(1 - x_1)x_2x_3$. Thus $p_i(x) = 0$ iff the clause is satisfied (where an assignment of true corresponds to setting the variable to 1). Thus we get a vector $p = (p_1, \ldots, p_n)$. If a is a satisfying assignment then $p(a)$ is the zero vector. Let $\langle u, v \rangle$ denote the dot product between vectors u and v. Then the main idea is to ask the prover for $\langle r, p(a) \rangle$ where $r \in \mathbb{Z}_2^m$ is chosen at random. We then have the following fact.

Lemma 0.36. *If $p(a) = 0$ then $\langle r, p(a) \rangle = 0$ with probability 1. Otherwise, if $p(a) \neq 0$ then $\langle r, p(a) \rangle = 0$ with probability at most $1/2$.*

Proof. The first part follows from the fact that $\langle r, 0 \rangle = 0$ for all r. Now suppose $p(a) \neq 0$. Then there is some r such that $\langle r, p(a) \rangle \neq 0$. In particular, $r = p(a)$ satisfies $\langle p(a), p(a) \rangle = 1$. Thus the linear map $f : r \mapsto \langle r, p(a) \rangle$ is onto. By rank nullity then we have $m = \dim \ker f + 1$ (since the dimension of the image $\mathbb{Z}_2$ is 1). Hence in particular the kernel is a vector space of dimension $m - 1$ and thus has $|\mathbb{Z}_2^{m-1}| = 2^{m-1}$ elements. So the probability that $\langle r, p(a) \rangle = 0$ is given by $2^{m-1}/2^m = 1/2$. □

Thus one protocol idea is to generate $r \in \mathbb{F}_2^m$ and then ask the prover for $\langle r, p(a) \rangle$. However, this gives no indication as to whether or not the prover is returning a value corresponding to the map $r \mapsto \langle r, p(a) \rangle$. Hence we have an additional test for linearity.

Linearity testing

Note that linear maps $f : F^m \to F$ for some scalar field F are exactly those of the form $f(x) = \langle x, b \rangle$ for some $b \in F^m$ since f is a $1 \times m$ matrix that acts via matrix multiplication on $m \times 1$ vectors (and this matrix multiplication is simply the dot product of two column vectors). Note that over $\mathbb{Z}_p$ we have that $f : \mathbb{Z}_p^m \to \mathbb{Z}_p$ as a map on vector spaces with scalar fields $\mathbb{Z}_p$ preserves additivity only if it preserves scalar multiplication.[6]

[6] It is crucial here that the underlying field is $\mathbb{Z}_p$ so that we can write $a \in \mathbb{Z}_p$ as $1 + \cdots + 1 = a$. Otherwise, it doesn't hold. For example, consider the map $z \mapsto \Re(z)$ from $\mathbb{C} \to \mathbb{C}$ over the field $\mathbb{C}$. This map preserves additivity but we have e.g. that $\Re(ii) = -1 \neq i\Re(i) = 0$.

Proposition 0.37. *Let $f : \mathbb{Z}_p^m \to \mathbb{Z}_p$ be a map of vector spaces each over the field $\mathbb{Z}_p$. If $f(u + v) = f(u) + f(v)$ for all $u, v \in \mathbb{Z}_p^m$ then $f(au) = af(u)$ for all $u \in \mathbb{Z}_p^m, a \in \mathbb{Z}_p$.*

Proof. Let $a \in \mathbb{Z}_p, u \in \mathbb{Z}_p^m$ be arbitrary. Then

$$f(au) = f(\underbrace{u + \cdots + u}_{a \text{ times}}) = \underbrace{f(u) + \cdots + f(u)}_{a \text{ times}} = af(u).$$

□

Hence to test linearity it is sufficient to test whether $f(u + v) = f(u) + f(v)$. In particular, the following is a protocol for testing linearity

Require: A function $f : \mathbb{Z}_p^m \to \mathbb{Z}_p$
V: Pick $u, v \in \mathbb{Z}_p^m$ at random. Ask the prover for $f(u), f(v), f(u+v)$.
P: Sends some values $f'(u), f'(v), f'(u+v)$.
V: If $f'(u) + f'(v) \neq f'(u+v)$ then reject. Otherwise continue.
V: Repeat k times and accept if each trial is a success.

Random self-correction

Unfortunately the above procedure does not guarantee that we can determine whether f is linear or not since a function could agree with a linear function on all but one value and we would only rule out such a function with low probability. However the protocol does convince the verifier V that there is some linear map g (possibly different from f) that at the very least agrees with f on a large fraction on inputs. If this is the case, then we can just ask the prover for $f(u+v), f(v)$ and then the chances will be high that we get $g(u+v), g(v)$ (since f, g agree on a large portion of inputs) and then we can compute $g(u+v) - g(v) = g(u)$. This helps reduce the errors in f and the general technique is known as *random self-correction*.

Formally, we have the following lemma.

Lemma 0.38. *Let V be an n-dimensional vector space over $\mathbb{Z}_p$. Suppose $0 < \varepsilon < 1/6$ and $f : V \to \mathbb{Z}_p$ satisfies*

$$\mathbb{P}_{u,v}\left(f(u+v) = f(u) + f(v)\right) > 1 - \varepsilon$$

then there is some linear map $g : V \to \mathbb{Z}_p$ such that

$$\frac{|\{u \in V : g(u) = f(u)\}|}{|V|} \geq 1 - 3\varepsilon > \frac{1}{2}.$$

Before proving the lemma we show how it is used.
Suppose then that for every linear function g, g agrees with f on strictly less than $1 - 3\varepsilon$ fraction of the inputs. Then the probability that the linearity test $f(u+v) = f(u) + f(v)$ succeeds for k rounds is at most $(1-\varepsilon)^k$ (since by the contrapositive of the above lemma we have that $\mathbb{P}_{u,v}\left(f(u+v) = f(u) + f(v)\right) \leq 1 - \varepsilon$). Hence by picking k large we can make this quantity small to ensure that the probability is small of passing the linearity test when f shouldn't pass.

Now we prove the lemma.

Proof of Lemma 0.38. Assume that f is as in the lemma statement. Let $g(u)$ be the majority value of $f(u+w) - f(w)$ over all possible $w \in V$. We prove that such a majority exists and that g is in fact a linear function. Define an equivalence relation $\equiv_u$ by

$$v \equiv_u w \Leftrightarrow f(v+u) - f(v) = f(w+u) - f(w).$$

Then $\equiv_u$ partitions V as it is an equivalence relation. Let $\delta_1, \ldots, \delta_k$ be the size of these partitions as fractions of $|V|$ such that $\delta_1 \leq \ldots \leq \delta_k$. Hence the ith equivalence class has size $\delta_i|V|$. We compute

$$\begin{aligned}
\mathbb{P}_{v,w}(v \equiv_u w) &= \frac{\sum_{i=1}^{k}(\delta_i|V|)^2}{|V|^2} \\
&= \sum_{i=1}^{k} \delta_i^2 \\
&\leq \delta_k \sum_{i=1}^{k} \delta_i \\
&= \delta_k
\end{aligned}$$

Note that the first equality comes from the fact that there are $(\delta_i|V|)^2$ ways to choose two elements from the ith equivalence class and $|V|^2$ ways to choose two elements from V.

which establishes an upper bound on $\mathbb{P}_{v,w}(v \equiv_u w)$. For a lower bound we compute

$$\begin{aligned}
\mathbb{P}_{v,w}(v \equiv_u w) &= \mathbb{P}_{v,w}(f(u+v) + f(w) = f(u+w) + f(v)) \\
&\geq \mathbb{P}_{v,w}(f(u+v) + f(w) = f(u+v+w) \text{ and} \\
&\qquad f(u+v+w) = f(u+w) + f(v)) \\
&\geq \mathbb{P}_{v,w}(f(u+v) + f(w) = f(u+v+w)) + \\
&\quad \mathbb{P}_{v,w}(f(u+v+w) = f(u+w) + f(v)) - 1 \\
&> (1-\varepsilon) + (1-\varepsilon) - 1 \\
&= 1 - 2\varepsilon.
\end{aligned}$$

The last equality follows from the assumption about f in the statement of the lemma.

Hence we have $1 - 2\varepsilon < \mathbb{P}_{v,w}(v \equiv_u w) \leq \delta_k$. Note by the assumption on ε we have

$$\begin{aligned}
\varepsilon &< \frac{1}{6} \\
2\varepsilon &< \frac{1}{3} \\
1 - 2\varepsilon &> 1 - \frac{1}{3} = \frac{2}{3}.
\end{aligned}$$

Hence we have $2/3 < 1 - 2\varepsilon < \delta_k$ which shows that the largest equivalence class of $\equiv_u$ contains a majority of the elements of V. Thus g is well-defined.

Now we show that g is linear. Let Δ_u be the largest equivalence class under $\equiv_u$. Thus for any $w \in \Delta_u$ we have $g(u) = f(u+w) - f(w)$ and $\mathbb{P}_w(w \in \Delta_u) = \delta_k$. Fix $u, v \in V$. We wish to show that $g(u+v) = g(u) + g(v)$. Suppose there exists some $w \in V$ such that $w \in \Delta_{u+v}$, and $w \in \Delta_v$ and $v + w \in \Delta_u$. Then as $w \in \Delta_{u+v}$ we would have $g(u+v) = f(u+v+w) - f(w)$ and similarly since $w \in \Delta_v$ and $v + w \in \Delta_u$ then likewise $g(v) = f(v+w) - f(w)$ and

$g(u) = f(v+w+u) - f(v+w)$. Note then we would have

$$\begin{aligned} g(u) + g(v) &= f(v+w+u) - f(v+w) + f(v+w) - f(w) \\ &= f(v+w+u) - f(w) \\ &= g(u+v). \end{aligned}$$

We show that such a w exists using the probabilistic method. Suppose that a w is randomly chosen from V. Then the probability that it satisfies the above constraints is given by

$$\begin{aligned} &\mathbb{P}_w(w \in \Delta_{u+v} \text{ and } w \in \Delta_v \text{ and } v+w \in \Delta_u) \\ &\geq \mathbb{P}_w(w \in \Delta_{u+v}) + \mathbb{P}_w(w \in \Delta_v) + \mathbb{P}_w(v+w \in \Delta_u) - 2 \\ &= \frac{|\Delta_{u+v}|}{|V|} + \frac{|\Delta_v|}{|V|} + \frac{|\Delta_u|}{|V|} - 2 \\ &> 1 - 2\varepsilon + 1 - 2\varepsilon + 1 - 2\varepsilon - 2 \\ &= 1 - 6\varepsilon > 0. \end{aligned}$$

Note that we have $|\Delta_u|/|V| > 1 - 2\varepsilon$ for any $u \in V$ since $|\Delta_u|/|V| = \delta_k > 1 - 2\varepsilon$ by the computation earlier in the proof. Hence since the probability that a randomly selected w satisfies the desired property is nonzero, at least one must exist. As u, v were arbitrary this shows that g is linear.

Finally, we wish to show that f and g agree on the majority their inputs.

$$\begin{aligned} \mathbb{P}_u(f(u) = g(u)) &\geq \mathbb{P}_{u,v}(f(u) = g(u) \text{ and } v \in \Delta_u) \\ &= \mathbb{P}_{u,v}(f(u) = f(u+v) - f(v) \text{ and } v \in \Delta_u) \\ &\geq \mathbb{P}_{u,v}(f(u) = f(u+v) - f(v)) + \mathbb{P}_{u,v}(v \in \Delta_u) - 1 \\ &> \frac{\sum_{u \in V} \mathbb{P}_v(v \in \Delta_u)}{|V|} - \varepsilon \\ &> 1 - 2\varepsilon - \varepsilon = 1 - 3\varepsilon > \frac{1}{2}. \end{aligned}$$

□

Proof of weak PCP

Let $y = \begin{pmatrix} y_1 & \cdots & y_m \end{pmatrix}^\top$ be a fresh set of variables. Consider the polynomial $\langle y, p \rangle = \sum_i y_i p_i(x)$. Note that this polynomial is linear in the y_i's and degree 3 in the x_i's. Thus we can rewrite this expression as

Recall that $p(x) = \begin{pmatrix} p_1(x) & \cdots & p_m(x) \end{pmatrix}$ where $p_i(x)$ is the 3-degree polynomial obtained from the ith clause of the SAT instance.

$$\sum_{i \in [m]} y_i p_i(x) = A(y) + \sum_{i \in [m]} B_i(y) x_i + \sum_{i,j \in [m]} C_{ij}(y) x_i x_j + \sum_{i,j,k \in [m]} D_{ijk}(y) x_i x_j x_k.$$

We can write this more neatly using tensor product notation. Let

$$B(y) \stackrel{\text{def}}{=} \Big(B_1(y) \quad \cdots \quad B_m(y)\Big)^\top$$
$$C(y) \stackrel{\text{def}}{=} \Big(C_{11}(y) \quad \cdots \quad C_{mm}(y)\Big)^\top$$
$$D(y) \stackrel{\text{def}}{=} \Big(D_{111}(y) \quad \cdots \quad D_{mmm}(y)\Big)^\top.$$

Then we have

$$\langle y, p(x)\rangle = A(y) + \langle B(y), x\rangle + \langle C(y), x \otimes x\rangle + \langle D(y), x \otimes x \otimes x\rangle.$$

Since $p_i(x) = 0$ iff clause i is satisfied, the verifier wants to be convinced that this is some $a \in \mathbb{Z}_2^m$ such that $\langle r, p(a)\rangle = 0$ for all $r \in \mathbb{Z}_2^n$. To do this, we ask for various values of the linear functions

$$r \mapsto \langle r, a\rangle$$
$$r \mapsto \langle r, a \otimes a\rangle$$
$$r \mapsto \langle r, a \otimes a \otimes a\rangle$$

for some fixed a. We label these functions f, g, and h respectively. The verifier's queries to the prover consist of f, g, or h and the input r of interest.

Using random self-correction the verifier can become convinced that the functions f, g, h are those above with high probability. With the following protocol, the verifier can become convinced that $p(a) = 0$.

V : pick a random $r \in \mathbb{Z}_2^m$ and compute $A(r), B(r), C(r), D(r)$. Ask the prover for $\langle B(r), a\rangle, \langle C(r), a \otimes a\rangle$, and $\langle D(r), a \otimes a \otimes a\rangle$ using random self-correction.[7]

P : sends some values x, y, z

V : compute the sum $x + y + x + A(r)$. If the result is nonzero then reject.

Repeat k times and accept if each trial is a success.

---

[7] For example to get the value of $\langle B(r), a\rangle$, the verifier asks for $f(B(r) + v), f(v)$ for a randomly chosen v and subtracts the two

The protocol uses $3k$ queries and $O(n^3)$ random bits. Thus it remains for the verifier to convince itself that f, g, h are functions of the form $r \mapsto \langle r, a\rangle, r \mapsto \langle r, a \otimes a\rangle$, and so on. By the linearity test above the verifier can become convinced that they are of the form

$$r \mapsto \langle r, a\rangle$$
$$s \mapsto \langle s, b\rangle$$
$$t \mapsto \langle t, c\rangle$$

but we also need to check that $b = a \otimes a$ and $c = a \otimes a \otimes a$. To check that $b = a \otimes a$ we perform the following protocol For an analysis of

---

V : pick a random $u, v \in \mathbb{Z}_2^n$. Ask the prover for $\langle u, a\rangle$, $\langle v, a\rangle$, $\langle u \otimes v, b\rangle$ using self correction with f and g.
P : returns some values purporting to be $\langle u, a\rangle, \langle v, a\rangle, \langle u \otimes v, b\rangle$
V : Check that $\langle u, a\rangle\langle v, a\rangle = \langle u \otimes v, b\rangle$. Reject if this test fails.
Repeat k times and accept if each trial is a success.

---

correctness we start by observing that the map $(u, v) \mapsto \langle u \otimes v, b\rangle$ is bilinear and thus can be written as $(u, v) \mapsto u^\top B v$ for some $n \times n$ matrix B. Note in particular we have that if $b = a \otimes a$ then $B_{ij} = e_i^\top B e_j = \langle (e_i \otimes e_j), b\rangle = \langle e_i, a\rangle\langle e_j, a\rangle = (aa^\top)_{ij}$. Hence $aa^\top = B$ holds iff $b = a \otimes a$. We then our test in the protocol above can be rewritten as

$$\begin{aligned} \langle u, a\rangle\langle v, a\rangle &= \langle u \otimes v, b\rangle \\ \langle u, a\rangle\langle a, v\rangle &= \langle u \otimes v, b\rangle \\ u^\top a a^\top v &= u^\top B v \\ u^\top (aa^\top - B) v &= 0. \end{aligned}$$

In particular if $B = aa^\top$ then the test succeeds. Otherwise, if $B \neq aa^\top$ then we need to show that the test fails with positive probability independent of n (that way we can get the probability large enough with a constant number of repetitions). We start by noting that for any nonzero $n \times n$ matrix M we have

$$\mathbb{P}(u^\top M v = 0) = \mathbb{P}(Mv = 0) + \mathbb{P}(u^\top M v = 0 | Mv \neq 0)\mathbb{P}(Mv \neq 0)$$

where all probabilities are taken over all possible choices of $u, v \in \mathbb{Z}_2^n$. Note that if $Mv \neq 0$ then the map $u \mapsto \langle u, Mv\rangle$ is a linear map with image of dimension 1 and hence Kernel of dimension $n - 1$. So $\mathbb{P}(u^\top M v = 0 | Mv \neq 0) = 2^{n-1}/2^n = 1/2$. Note also since $M \neq 0$ we have $\operatorname{rank}(M) > 0$ and so $\dim \ker(M) = n - \operatorname{rank}(M) < n$. With this in mind we observe

$$\begin{aligned} \mathbb{P}(Mv = 0) &= \mathbb{P}(v \in \ker(M)) \\ &= \frac{|\mathbb{Z}_2|^{\dim \ker(M)}}{|\mathbb{Z}_2|^n} \\ &= \frac{1}{2^{\operatorname{rank}(M)}} \leq \frac{1}{2}. \end{aligned}$$

Hence we can write

$$\begin{aligned} \mathbb{P}(Mv = 0) + \mathbb{P}(u^\top M v = 0 | Mv \neq 0)\mathbb{P}(Mv \neq 0) &\leq \frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{2}\right) \\ &= \frac{3}{4}. \end{aligned}$$

Thus, if the prover is being dishonest and $b \neq a \otimes a$ then the probability that the verifier becomes convinced (falsely) that $b = a \otimes a$ is $\leq (3/4)^k$ which we can make very small by making k, a large constant. The protocol for ensuring that $c = a \otimes a \otimes a$ is similar.

Circuit Complexity

The main topics we cover in this sections are

- Definitions of AC, $\mathrm{P}_{/\mathrm{POLY}}$ and NC.
- Karp-Lipton Theorem
- PARITY is outside AC^0.

Definitions

We write a circuit family as $\{C_n\}_{n\in\mathbb{N}}$ where C_n is a Boolean circuit that takes computes inputs of length n. We write $C_n(x)$ to denote the output bit resulting from the computation of C_n on input x. For a function $T : \mathbb{N} \to \mathbb{N}$ we say that a circuit family $\{C_n\}$ has size $T(n)$ if the number of nodes in C_n (denotes $|C_n|$) satisfies $|C_n| \leq T(n)$ for every n. A language L is in $\mathrm{SIZE}(T(n))$ if there exists a $T(n)$-size circuit family $\{C_n\}$ such that for every $x \in \{0,1\}^n$ we have $x \in L \Leftrightarrow C_n(x) = 1$.

Definition 0.39 (The complexity class $\mathrm{P}_{/\mathrm{POLY}}$)**.**

$$\mathrm{P}_{/\mathrm{POLY}} \stackrel{\text{def}}{=} \bigcup_{c\in\mathbb{N}} \mathrm{SIZE}(n^c)$$

is the class of all languages decidable by polynomial-size circuit families.

While the size of a circuit is the number of nodes, we can likewise consider the *depth* of a circuit which is the length of the longest path from the input gate to the output gate. Hence we say a language has depth complexity $D(n)$ (for $D : \mathbb{N} \to \mathbb{N}$) if there is a circuit family such that $C_n(x) = 1 \Leftrightarrow x \in L$ and the depth of C_n is bounded above by $D(n)$. We say that a language has size-depth complexity $(T(n), D(n))$ if a uniform circuit family[8] exists with size complexity $T(n)$ and depth complexity $D(n)$.

[8] A circuit family $\{C_n\}$ is uniform if there exists a LogSpace-DTM that can construct C_n on input 1^n.

Definition 0.40 (Nick's class)**.** For $i \geq 0$, NC^i is the class of all languages solvable by $(\mathrm{poly}(n), \log^i(n))$ Boolean circuits. Nick's class denoted NC is then defined as

$$\mathrm{NC} = \bigcup_{i\in\mathbb{N}} \mathrm{NC}^i.$$

Nick's class is named after Nick Pippenger who did a lot of the pioneering research on circuits of polynomial size and logarithmic depth.

Note that we require an NC family of circuits to be Boolean which means each gate in the circuit has fan-in at most 2. If we allow arbitrary fan-in gates then we get another hierarchy.

Definition 0.41 (Alternations class)**.** For $i \geq 0$, AC^i is the class of all languages solvable by $(\mathrm{poly}(n), \log^i(n))$ circuits with arbitrary fan-in gates. The class AC is defined as

$$\mathrm{AC} = \bigcup_{i \in \mathbb{N}} \mathrm{AC}^i$$

We observe that

$$\mathrm{NC}^0 \subseteq \mathrm{AC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{AC}^1 \subseteq \cdots \subseteq \mathrm{P}_{/\mathrm{POLY}}$$

since a gate in a polynomial-size circuit with unbounded fan-in gates can be simulated using a Boolean circuit of logarithmic depth.

Facts about $\mathrm{P}_{/\mathrm{POLY}}$

A key insight is that the computation of an arbitrary polynomial-time DTM can be simulated by polynomial-size Boolean circuits. In particular, we have the following result.

Theorem 0.42. $\mathrm{P} \subseteq \mathrm{P}_{/\mathrm{POLY}}$.

Proof sketch. Let M be a polynomial-time DTM that decides a language L and runs in time $O(p(n))$ for a polynomial p. Let x be an input with length n. Note that there are $O(p(n))$ possible TM configurations during the computation of some input x. We organize the gates of C_n in layers corresponding to the $O(p(n))$ possible configurations. We add wires between layers to encode the transition function. Finally we build wiring to check that the final configuration is an accepting one. This way the circuit outputs 1 iff M accepts that particular input. Note then that since there are $O(p(n))$ configurations and each is $O(p(n))$ bits long (since the computation is space bounded by the time bound), the total size is $O(p(n)^2)$ which is still polynomial. □

Note that we have the following characterization of $\mathrm{P}_{/\mathrm{POLY}}$.

Proposition 0.43. *A language L is in $\mathrm{P}_{/\mathrm{POLY}}$ iff there is a language $L' \in \mathrm{P}$ and advice strings $a_0, a_1, \ldots$ with $|a_n| \leq p(n)$ for some polynomial $p(n)$ and $x \in L \Leftrightarrow (x, a_{|x|}) \in L'$.*

Proof. Suppose $L \in \mathrm{P}_{/\mathrm{POLY}}$. Then there is a polynomial-size circuit family which computes L. We can just use the description of the circuit C_n as the advice string a_n. Then there is a polynomial-time DTM that on input (x, C_n) computes the circuit C_n on x.

Conversely if L is decidable by a polynomial-time DTM M which takes as input x and advice string a_n, then by Theorem 0.42 we can construct a family of circuits $\{C_n\}$ which on input (x, a_n) decides whether $(x, a_n) \in L$. Then we simply define the family $\{C'_n\}$ by $C'_n(x) = C_{n+|a_n|}(x, a_n)$ (since $|(x, a_n)| = |x| + |a_n|$). In other words, we simply reuse the circuits C_n with the advice string a_n fixed. □

Karp-Lipton Theorem

The Karp-Lipton theorem provides evidence that NP is probably not contained in $\mathrm{P}_{/\mathrm{POLY}}$.

Theorem 0.44 (Karp-Lipton Theorem)**.** *If* $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{POLY}}$ *then* $\mathrm{PH} = \Sigma_2$

Proof. Our strategy will be to show that $\Pi_2 \subseteq \Sigma_2$ which will inductively imply the collapse of the hierarchy to the second level. Consider the following Π_2-complete problem: given a Boolean formula φ over variables $V = \{x_1, \ldots, x_{2n}\}$ decide if

This proof is adapted from both [AB09] and [Aar16]

$$\forall x \in \{0,1\}^n \; \exists y \in \{0,1\}^n \; \varphi(x, y) = 1$$

holds where φ contains no quantifiers. Note that the following problem is in NP: given φ and x does there exist a $y \in \{0,1\}^n$ such that $\varphi(x, y) = 1$. Thus by assumption since this problem is in NP it is also in $\mathrm{P}_{/\mathrm{POLY}}$ which shows that there is a polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that $C_n((\varphi, x)) = 1$ iff there exists a $y \in \{0,1\}^n$ such that $\varphi(x, y) = 1$. Just as we can easily modify a polynomial-time algorithm that decides SAT into one that computes a satisfying assignment in polynomial-time we can likewise modify the polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ into a polynomial-size circuit family $\{C'_n\}_{n\in\mathbb{N}}$ which outputs the value y which satisfies $\varphi(x, y)$. Hence we can turn the above Π_2-complete problem into the following Σ_2 problem: does there exist a circuit C such that for all $x \in \{0,1\}^n$ we have $\varphi(x, C(x, \varphi)) = 1$. Note it is important here that the if the statement "$\forall x \in \{0,1\}^n \; \exists y \in \{0,1\}^n \; \varphi(x, y) = 1$" holds then there is a *polynomial*-size circuit C (polynomial in $|(\varphi, x)|$) satisfying the above since otherwise the problem we turned the original one into may not lie in Σ_2. □

Håstad's Switching Lemma

A key result in circuit complexity is Håstad's switching lemma which allows us to "switch" between k-DNF formulas and s-CNF formulas with high probability. This lemma is used to prove a number of lower bounds in circuit complexity.

Note that a k-DNF formula is of the form $\vee_{i=1}^{m} C_i$ where each C_i is a conjunction of at most k literals. Also, s-CNF formulas are defined likewise.

The statement of the theorem requires a notion of random restrictions. Let f be a function on n variables. A random restriction ρ is a partial assignment to a subset of the variables of f. The restriction of f under ρ denoted $f|_\rho$ takes an assignment to the variables left unrestricted by ρ and computes f on those values and the values set by ρ.

Lemma 0.45 (Håstad's Switching Lemma)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be expressible as a k-DNF and let ρ denote a random restriction that assigns values to t randomly selected input bits. Then for every $s \geq 2$ we have*

$$\mathbb{P}_\rho[f|_\rho \text{ is not expressible as a } s\text{-CNF}] \leq \left(\frac{(n-t)k^{10}}{n}\right)^{s/2}.$$

Proof sketch [ABo9]. Let R_t denote the set of all restrictions to t variables so that $|R_t| = \binom{n}{t}2^t$. Let B be the set of *bad restrictions* which satisfy $\rho \in B \Rightarrow f_\rho$ is not expressible as an s-CNF. The proof proceed by exhibiting a one to one correspondence between the set B and $R_{t+s} \times \{0,1\}^\ell$ for $\ell = O(s \log k)$ and R_{t+s}, the est of random restrictions on $(t+s)$ variables. This product has size $\binom{n}{t+s}2^{O((t+s)(s\log k))} = \binom{n}{t+s}2^t k^{O(s)}$. So $|B|/|R_t|$ can be bounded by

$$\frac{\binom{n}{t+s}2^t k^{O(s)}}{\binom{n}{t}2^t} = \frac{\binom{n}{t+s}k^{O(s)}}{\binom{n}{t}}$$

which is the probability that a random restriction is bad. With a little bit of work we can bound this quantity above by the desired expression.

□

Parity is not in AC^0

A parity function is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) \equiv \sum_{i=1}^n x_i \pmod 2$ or equivalently it is a Boolean function such that flipping any single input bit will flip the output bit or also equivalently it is the language defined by the set

$$\{x \in \{0,1\}^n : x \text{ has an odd number of } 1s\}.$$

The main theorem of this section is the following

Theorem 0.46 (Parity is not in AC^0)**.** *Let f be a parity function. Then f is not computable by an AC^0 circuit.*

There are two main proofs for Theorem 0.46 - perhaps the most well-known being that which utilizes Håstad's switching lemma (Lemma 0.45). There is another proof sometimes referred to as the "polynomial proof" which relies on low-degree polynomial constructions. We start with the polynomial proof. In particular we prove the

following theorem which is actually a stronger result than Theorem 0.46.

We follow the exposition of the polynomial proof found in Jonathan Katz's lecture notes [Kat08] which can also be found in [LMN+90] and was originally published in [Raz87] and [Smo87].

Theorem 0.47. *For n large enough, any constant depth circuit that computes the parity function $f : \{0,1\}^n \to \{0,1\}$ must have at least*

$$\frac{1}{50}\left(\sqrt{2^{n^{1/(2d)}}}\right)$$

gates.

Let $\varepsilon = 1/(2d)$ then the above states that the size of any circuit family computing parity is $O(2^{n^\varepsilon})$ which is superpolynomial. The proof is broken into two parts (lemmas). First, we show that any AC^0 circuit can be approximated by a low-degree polynomial and then we show that a parity function cannot be approximated by a low-degree polynomial. We start with some preliminary remarks.

Let $\mathbb{F}_3 = \{-1, 0, 1\}$ be a field of size 3 so that e.g. $1 + 1 = -1$. A polynomial $p \in \mathbb{F}_3[x_1, \ldots, x_n]$ is called *proper* if $p(x_1, \ldots, x_n) \in \{0,1\}$ for any $x_1, \ldots, x_n \in \{0,1\}$. We now state and prove the first lemma.

Lemma 0.48 (Any AC^0 circuit can be approximated by a low-degree polynomial)**.** *Let C be a depth d circuit with polynomial size and n inputs. For every $t > 0$, there is a proper polynomial p such that* $\deg(p) = (2t)^d$ *and p differs from C on at most* $\mathrm{SIZE}(C)(2^{n-t})$ *inputs.*

Proof. Let C be a circuit of depth d with n inputs. Assume without loss of generality that C has only $\vee$ gates and $\neg$ gates (this is without loss of generality since we can use De Morgan's laws to convert any $\wedge$ gates into two $\neg$ gates and one $\vee$ gate). Associate a proper polynomial with each wire of the circuit in the following way. Associate x_i with each input wire. Now proceed inductively through the layers of the circuit assuming that the polynomials of the parent level have already been set (here we are viewing the circuit as a tree with the input level at the top). For a gate g we break into cases depending on whether g is a $\neg$ gate or an $\vee$ gate:

$\neg$ gate $-$ associate with g the polynomial $1 - p$ where p is the polynomial associated with the input wire

$\vee$ gate $-$ associate with g the polynomial

$$p = 1 - \prod_{i=1}^{t}(1 - q_i)$$

where

$$q_i = \left(\sum_{j \in S_i} p_j\right)^2$$

for $i = 1, \ldots, t$ and where $p_1, \ldots, k$ are the polynomials associated with the input wires and $S_1, \ldots, S_t \subseteq [k] = \{1, \ldots, k\}$ are chosen to minimize error (see below).

We observe that this process preserves the polynomials being proper since $q_i \geq 0$ and $1 - p$ is proper if p is. Also the degree of $1 - p$ is the degree of p and

$$\begin{aligned} deg\left(1 - \prod_{i=1}^{t}(1 - q_i)\right) &= \sum_{i=1}^{t} \deg(1 - q_i) \\ &= \sum_{i=1}^{t} \deg\left(\sum_{j \in S_i} p_j\right)^2 \\ &\leq \sum_{i=1}^{t} 2 \max_j \deg(p_j) \\ &= 2t \max_j \deg(p_j). \end{aligned}$$

Let p' be the polynomial associated with the output wire of the circuit. We bound the error[9] inductively. No error is introduced at with the input wires or any $\neg$ gates. On an $\vee$ gate with associated input polynomials $p_1, \ldots, p_k$ we describe how to select the sets $S_1, \ldots, S_t \subseteq [k]$ to make the error small. If all the p_i evaluate to 0 on an input then p will be 0 on that input as well since $1 - q_i$ will be 1. Otherwise suppose that one of the p_i evaluates to 1. Call this polynomial p_ℓ. We look at one particular S_i and show that

$$\mathop{\mathbb{P}}_{S_i \subseteq [k]}\left(\sum_{j \in S_i} p_j(x) = 0\right) \leq \frac{1}{2}$$

where the probability is taken over all possible choices of S_i. Let $P = \{T \subseteq [k] : \sum_{j \in T} p_j(x) = 0\}$. Hence the probability is

$$\mathop{\mathbb{P}}_{S_i \subseteq [k]}\left(\sum_{j \in S_i} p_j = 0\right) = \frac{|P|}{2^k}$$

since there are 2^k subsets of $[k]$. This observation shows it is sufficient to demonstrate that $|P| \leq 2^{k-1}$. Let S' be an arbitrary subset of $[k] \setminus \{\ell\}$. Then it cannot be the case that both $\sum_{j \in S'} p_j = 0$ AND $p_\ell + \sum_{j \in S'} p_j = 0$ as $p_\ell = 1$. It follows that either $S' \notin P$ or $S' \cup \{\ell\} \notin P$. Since there are 2^{k-1} such S' we must have $|P| \leq 2^k - 2^{k-1} = 2^{k-1}$. It follows then immediately that

$$\mathop{\mathbb{P}}_{S_i \subseteq [k]}[q_i \text{ evaluates to } 1] \geq \frac{1}{2}$$

since if $\sum_{j \in S_i} p_j(x) \neq 0$ then the sum is either -1 or 1 in which case the square of it, q_i, is 1. We also observe that if any of the q_i evaluate to 1 then $p = 1 - \prod_{i=1}^{t}(1 - q_i)$ evaluates to 1. Hence the probability that p does not evaluate to 1 is $\leq 1/2^t$. But this implies that there is some specific choice of subsets S_i such that the probability that p

[9] We think of error as being the fraction of inputs on which p' differs from C.

does not evaluate to 1 is $\leq 1/2^t$ because otherwise, we could average this probability value over all possible choices of S_i and would get something $> 1/2^t$ which would contradict our above computation.[10]

Since each level increases the degree of the polynomial associated with each wire by at most $2t$ times the max degree of the parent polynomials and the degree of the start polynomials are 1, the degree of p' is at most $(2t)^d$ (as there are d levels). Also suppose we fix an input x and let E_g denote the event that the polynomial associated with g introduced an error. Then the probability that some error occurs is bounded above by the probability of the event $\cup_{g \in C} E_g$ (note this is an upper bound since potentially two errors could be introduced with cancel each other out, but we are only interested in an upper bound on the error probability).[11] We can thus compute

$$\mathbb{P}\left(\bigcup_{g \in C} E_g\right) \leq \sum_{g \in C} \mathbb{P}(E_g) \leq \sum_{g \in C} \frac{1}{2^t} = \frac{\text{SIZE}(C)}{2^t}.$$

Hence since there are 2^n possible inputs w have that the number of inputs causing an error are at most

$$(2^n)\frac{\text{SIZE}(C)}{2^t} = \text{SIZE}(C)(2^{n-t}).$$

□

Lemma 0.49. *Let $p \in \mathbb{F}_3[x_1, \ldots, x_n]$ be a proper polynomial with $\deg(p) \leq \sqrt{n}$. Then p differs from the parity function on at least $2^n/50$ inputs for n large enough.*

Proof. Here is a roadmap of the proof.

1. Consider the set S of all inputs on which p and the parity function agree.
2. Consider the set of all functions from S to $\mathbb{F}_3$.
3. Show that all such functions can be written as polynomials.
4. Rewrite such polynomials to reduce their degree.
5. Use the bound on their degree to give an upper bound on the total number of polynomials from S to $\mathbb{F}_3$.
6. Use this upper bound to get an upper bound on the set S which is the number of points on which p and the parity function agree.

In our analysis it will help to consider functions from $\{-1,1\}^n \to \{-1,1\}$ instead of the traditional $\{0,1\}^n \to \{0,1\}$. At the outset this change is mainly a cosmetic detail but it will make the construction possible. Let $\oplus : \{0,1\}^n \to \{0,1\}$ be the original parity function. Then the new parity function under our change is the function $\oplus' : \{-1,1\}^n \to \{-1,1\}$ given by $\oplus'(x_1, \ldots, x_n) = \prod_{i=1}^n x_i$ since flipping

[10] This is the same type of averaging argument used to show that allowing the prover in an interactive proof to access random bits won't change the class of problems that can be solved.

[11] We abuse notation here and write $g \in C$ to mean that g is a gate in C.

any one input bit (from -1 to 1 or vice versa) will flip the output bit. Note we can relate the two functions over $\mathbb{F}_3$ by

$$\oplus'(x_1, \ldots, x_n) = \oplus(x_1 - 1, \ldots, x_n - 1) + 1$$

since if $x_i \in \{-1, 1\}$ then $x_i - 1 \in \{0, 1\}$ and likewise $\oplus(x) + 1 \in \{-1, 1\}$ since $\oplus(x) \in \{0, 1\}$. Thus for a polynomial p that agrees with $\oplus$ on m inputs then there is a polynomial $p' : \{-1, 1\} \to \{-1, 1\}$ which agrees with $\oplus'$ on m inputs (simply set $p' = p(x_1 - 1, \ldots, x_n - 1)$ and the statement follows from the above equality). Note also that p' will have degree at most $\sqrt{n}$ as well.

Let $S \subseteq \{-1, 1\}^n$ be the set of inputs on which $\oplus'$ and p' agree as in the first step of the roadmap. Let F be the set of all functions $f : S \to \mathbb{F}_3$. We have $|F| = |\mathbb{F}_3|^{|S|} = 3^{|S|}$. Let $f \in F$. We show how to write f has a polynomial $p_f \in \mathbb{F}_3[x_1, \ldots, x_n]$ (so that $f(x) = p_f(x)$ for all $x \in S$). Specifically we define

$$p_f(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum_{y \in S} f(y) \prod_{i=1}^{n} (-y_i x_i - 1).$$

To see why for any $z \in S$ we have $p_f(z) = f(z)$ note that for any $y \neq z$ there is some i such that $y_i \neq z_i$ and, as $y_i, z_i \in \{-1, 1\}$, we must have $z_i = 1$ and $y_i = -1$ or vice versa. Either way we have $y_i z_i = -1$ so that $-y_i x_i - 1 = 0$. Thus the product $\prod_{i=1}^{n} (-y_i z_i - 1)$ evaluates to 1 only if $z_i = y_i$ for all i and otherwise the product is 0. It follows that $p_f(z) = f(z)$ since all other terms in the sum from the definition will be 0.

Now we reduce the degree of p_f (since $\deg(p_f)$ could be as large as n). A monomial in p_f will look like $c \prod_{i \in T} x_i$ for $T \subseteq [n]$ and $c \in \{-1, 1\}$. The set T is the degree of this monomial. Suppose that $|T| > n/2 + \sqrt{n}$ then we rewrite the monomial as

$$\begin{aligned} c \prod_{i \in T} x_i &= c \prod_{i \in [n]} x_i \cdot \prod_{i \notin T} x_i \\ &= c \oplus'(x) \prod_{i \notin T} x_i \\ &= c p'(x) \prod_{i \notin T} x_i \end{aligned}$$

The first equivalence holds since if $\prod_{i \in T} x_i = 1$ then the number of -1s is even and hence the parity of the number of -1s in $\prod_{i \in [n]} x_i$ is the same as $\prod_{i \notin T} x_i$. Likewise if the number of -1s in $\prod_{i \in T} x_i = 1$ is odd then exactly one of the other products has an odd number of -1s.

Note that $\oplus'(x) = p'(x)$ since we are only consider functions on the set S which is the set on which the two functions agree. The above has degree

$$\deg(p') + n - |T| \leq \sqrt{n} + n - (n/2 + \sqrt{n}) = n/2.$$

Thus applying this transformation to each monomial with degree larger than $n/2 + \sqrt{n}$ we can guarantee that p_f has degree $n/2 + \sqrt{n}$.

Suppose that a monomial has degree i. Then the number of such monomials (ignoring coefficients) is given by $\binom{n}{i}$ since this quantity is how many ways we can choose i variables from the set $\{x_1, \ldots, x_n\}$ and we don't care about ordering since multiplication is commutative. Thus the total number of monomials whose degree is at most $n/2 + \sqrt{n}$ is given by

$$\sum_{i=0}^{n/2+\sqrt{n}} \binom{n}{i}.$$

To bound this quantity we note that $\sum_{i=0}^{n} \binom{n}{i} = 2^n$. Also as $\binom{n}{n-1} = n$ we observe that for n large enough we have

$$\begin{aligned}\sum_{i=0}^{n/2+\sqrt{n}} \binom{n}{i} &< \sum_{i=0}^{n-1} \binom{n}{i} \\ &= 2^n - n - 1 < 2^n - n \\ &< 2^n - \frac{2^n}{50} \\ &= \frac{49}{50}(2^n).\end{aligned}$$

Thus the total number of polynomials who degree is bounded above by $n/2 + \sqrt{n}$ is at most $3^{\frac{49}{50}(2^n)}$ since for each monomial we have a choice of 3 coefficients and there are at most $\frac{49}{50}(2^n)$ monomials. Hence we can write

$$|F| = 3^{|S|} \leq 3^{\frac{49}{50}(2^n)} \Rightarrow |S| \leq \frac{49}{50}(2^n)$$

which completes the proof since $|S|$ is the number of points on which p' and $\oplus'$ agree (and hence by construction also the number of points on which p and $\oplus$ agree). □

With these two lemmas are in a position to prove Theorem 0.47 which we restate for completeness.

Theorem 0.47. For n large enough, any constant depth circuit that computes the parity function $f : \{0,1\}^n \to \{0,1\}$ must have at least

$$\frac{1}{50}\left(\sqrt{2^{n^{1/(2d)}}}\right)$$

gates.

Proof. Let C be a constant depth circuit that computes parity. Let $t = \frac{1}{2}(n^{1/(2d)})$ in Lemma 0.48. Then we have a polynomial of degree at most $(n^{1/(2d)})^d = \sqrt{n}$ that differs from the parity function on at most

$$\text{SIZE}(C) \cdot 2^{n-t} = \text{SIZE}(C) \cdot 2^{n-n^{1/(2d)}/2}$$

inputs. Then by Lemma 0.49, the polynomial p must differ from the parity function by at least $2^n/50$. This shows that

$$\begin{aligned}\text{SIZE}(C) \cdot 2^{n-n^{1/(2d)}/2} &\geq 2^n/50 \\ \text{SIZE}(C) &\geq \frac{1}{50}\left(\frac{1}{2^{n^{1/(2d)}/2}}\right).\end{aligned}$$

□

As an immediate corollary we have

Corollary 0.50. *Parity is not in* AC^0.

Proof. By Theorem 0.46 any constant depth circuit computing the parity function has superpolynomial size and thus no AC^0 circuit family can compute parity. □

An alternative proof that parity is outside AC^0 relies on the switching lemma.

Second proof of Theorem 0.46 using Håsted's switching lemma. The main idea of the proof is that using the switching lemma we can reduce any AC^0 circuit to two levels. Basically we consider the DNF subcircuits at second to last layer (before the output bit) and apply a random restriction which will allow us with high probablity to swap out the DNF formulas with CNF formulas and absorb the conjunctions into the conjunctions at level 1 (since we are allowed unbounded fan-in). This process reduced the depth by one level. Continuing this way we eventually get a circuit of depth 2. But this is just a k-DNF or k-CNF which means we can fix k variables to make the circuit always evaluate to 1 (e.g. if it is a DNF formula we can make one clause evaluate to true). But such a circuit cannot compute parity since regardless of how many variables we restrict, we can always flip the output bit by flipping on of the unrestricted input bits.

For the detailed proof, let C be an AC^0 circuit. Without loss of generality, we assume that all gates in C have fan-out 1 and C is separated into levels in which $\wedge$ gates and $\vee$ gates alternate between levels and all $\neg$ gates are located at the top with the input wires.

Let n^b be an upper bound on the size of C. We iteratively apply random restrictions to more and more variables until the circuit has depth 2. Let n_i be the number of unrestricted variables at step i. Then we restrict $n_i - \sqrt{n_i}$ variables to get to step $i+1$. Thus $n_{i+1} = n_i - (n_i - \sqrt{n_i}) = \sqrt{n_i}$. We start with $n_0 = n$ so inductively we get that $n_i = n^{1/2^i}$. Let $k_i = 10b2^i$. The goal is to show that, with high probability, at the ith iteration the depth of the circuit is $d - i$ and the fan-in of the bottom gate is at most k_i. We proceed by induction on i. Suppose that the bottom level is an $\wedge$ gate. By assumption the level above it contains $\vee$ gates. Each such $\vee$ gate computes a k_i-DNF. We apply Håsted's switching lemma with $k = k_i$, $s = k_{i+1}$ and $t = n_i - n_{i+1}$ (since the number of variables we restrict at the ith iteration is given by $n_i - n_{i+1}$, the difference between the number of unrestricted variables at level i and the number of unrestricted variables at level $i+1$). Also note that the n in the lemma is n_i since we have already restricted some variables. Then we have the probability that the function associated with the $\vee$ gate is not expressible as a

k_{i+1}-CNF is given by

$$\left(\frac{(n_i - (n_i - n_{i+1}))k_i^{10}}{n_i}\right)^{k_{i+1}/2} = \left(\frac{k_i^{10}}{n^{1/2^{i+1}}}\right)^{k_{i+1}/2}.$$

We compute the RHS further as

$$\begin{aligned}\left(\frac{k_i^{10}}{n^{1/2^{i+1}}}\right)^{k_{i+1}/2} &= \left(\frac{(10b2^i)^{10}}{n^{1/2^{i+1}}}\right)^{10b2^{i-1}} \\ &= \frac{(10b2^i)^{100b2^{i-1}}}{n^{2.5b}}.\end{aligned}$$

Note that for constants c_1, c_2 we can always find n large enough so that $c_1/n^{2.5} \leq 1/c_2 n$. Likewise we can find n large enough so that

$$\frac{(10b2^i)^{100b2^{i-1}}}{n^{2.5b}} \leq \frac{1}{10n^b}.$$

Thus we have that the probability the k_i-DNF formula can be expressible as a k_{i+1}-CNF is at least

$$1 - \frac{1}{10n^b}.$$

Thus we can replace the DNF formula with a CNF formula and merge it with the $\wedge$ gate above it (the level 1 gate), thereby reducing the depth of the circuit by 1. Analogous reasoning also applies to the case when the level 1 gate is an $\vee$ gate in which case we can swap the k_i-CNF with a DNF formula and merge. The probability of failure[12] is at most $1/10n^b$ and we apply the lemma at most once to all the gates in the circuit (which is n^b). Thus the probability of a failure at any point in the process is at most

$$\sum_{j=1}^{n^b} \frac{1}{10n^b} = \frac{1}{10}.$$

Hence the probability of no failures is at least $9/10$. Since this probability is nonzero there must be some sequence of partial valuations that realize it. Then, we get a circuit of depth 2 which cannot compute parity for reasons mentioned in the beginning of the proof. □

[12] A failure in this context means the DNF/CNF formula was not expressible as a CNF/DNF formula so we could not apply the switching lemma.

References

[Aar16] Scott Aaronson. P $\stackrel{?}{=}$ NP. In *Open problems in mathematics*, pages 1–122. Springer, 2016.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[BRS95] Richard Beigel, Nick Reingold, and Daniel Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.

[For97] Lance Fortnow. Counting complexity. *Complexity theory retrospective II*, pages 81–107, 1997.

[For09] Lance Fortnow. A simple proof of Toda's theorem. *Theory of Computing*, 5(1):135–140, 2009.

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

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, pages 174–187. IEEE, 1986.

[HK16] Christoph Haase and Stefan Kiefer. The complexity of the kth largest subset problem and related problems. *Information Processing Letters*, 116(2):111–115, 2016.

[Kat08] Jonathan Katz. Lecture on parity. `http://www.cs.umd.edu/~jkatz/complexity/f05/parity.pdf`, March 2008. [Online; accessed 4-June-2019].

[Koz06] Dexter C Kozen. *Theory of computation*. Springer Science & Business Media, 2006.

[LMN+90] Jan V Leeuwen, AR Meyer, M Nival, et al. *Handbook of theoretical computer science: Algorithms and complexity*. MIT Press, 1990.

[Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

[Sha90] Adi Shamir. IP= PSPACE (interactive proof=polynomial space). In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 11–15. IEEE, 1990.

[She92] Alexander Shen. IP=SPACE: simplified proof. *Journal of the ACM (JACM)*, 39(4):878–880, 1992.

[Smo87] Roman 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*, pages 77–82. ACM, 1987.