CS640 Course Project

Rishabh Batra Devansh Shringi
Submitted to: Raghunath Tewari

November 8, 2021

Contents

1 Introduction

We present a summarized interpretation of the paper Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds [KI03] by Valentine Kabanets and Russell Impagliazzo, based on our reading of the paper.

The main focus of this project will be proof ideas of the following theorem, which presents how derandomizing PIT gives lower bounds for either arithmetic circuits or boolean circuits.

Theorem 1.1.

$$PIT \in P \implies per \notin Arth - P/poly \text{ or } NEXP \nsubseteq P/poly$$

We will first present brief description of the preliminaries required for understanding the proof of theorem. These are :

- Arithmetic Circuits
- The problem of Polynomial Identity Testing (PIT)
- PseudoRandom Generators (PRGs)

Then, we will look at the proof ideas key lemma's that give us the theorem. These lemmas are

Lemma 1.2. $PIT \in P$ and $per \in Arth - P/poly \implies P^{per} \subseteq NP$

Lemma 1.3. $EXP \subseteq P/poly \implies EXP = MA$

Lemma 1.4. $NEXP \in P/poly \implies NEXP = EXP$

After that we will see how these lemmas combine to give the main theorem. AT the end we will look at the implications and open questions related.

2 Preliminaries

We use $\mathbb{F}$ to represent fields. We will mainly be working with function fields, and the variables will be denoted mainly by $x_1, \ldots, x_n$ with n denoting the number of indeterminants. $\mathbb{F}[x_1, \ldots, x_n]$ will denote the polynomial ring over $\mathbb{F}$.

2.1 Arithmetic Circuits

Arithmetic circuits are the most natural and standard model for computing polynomials and we will be using these to represent polynomials. We use the following definition of Arithmetic circuits

Definition 2.1.1. (Arithmetic circuits) An arithmetic circuit C over the field $\mathbb{F}$ and the set of variables $x_1, \ldots, x_n$ is a directed acyclic graph as follows. The vertices of C are called gates. Every gate of C of in-degree 0 is labelled by either a variable or a field element. Every other gate is labelled by either $+$ or $\times$. An edge is labelled with field constants, which is 1 by default.

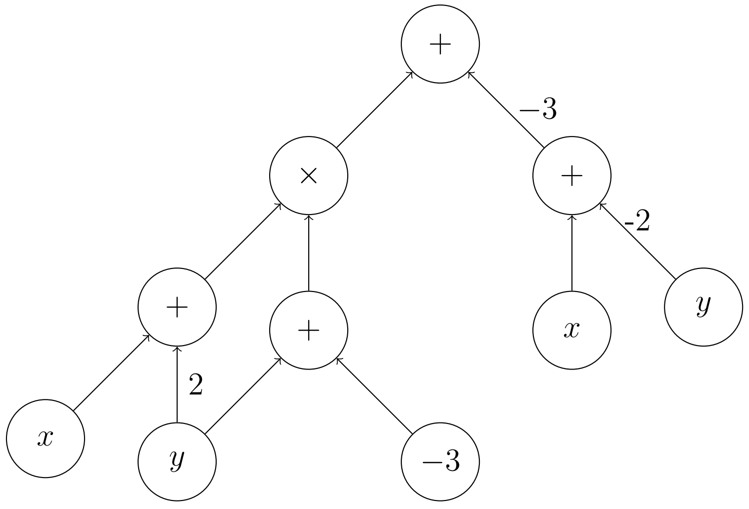

Figure 1: Circuit computing $xy + 2y^2$

An arithmetic circuit computes a polynomial in a natural way : An input gate labeled by $\alpha \in \mathbb{F} \cup \{x_1, \ldots, x_n\}$ computes the polynomial α. A product gate (gate with label $\times$) computes the product of the polynomials computed by its children. Similarly a sum gate (gate with label $+$) computes the sum of the polynomials computed by its children. An example of an arithmetic circuit is given below.

We define the *size* of an arithmetic circuit to be the number of edges in the graph. We define the *depth* of a gate to be the length of the longest directed path to it. The depth of a circuit is the maximal depth of a gate in it.

As we saw in the model of arithmetic circuits, the two main resources are size and depth. Based on size, we define class *VP* as the family of circuits $\{C_n\}$ computing polynomials such that n is number of variables , degree and size of the circuit is bounded by $poly(n)$. The class is the arithmetic analog of P. For more details in the algebraic complexity area, refer to the survey [SY10].

2.2 Polynomial Identity Testing

Polynomial Identity Testing (PIT). It is the problem in which we are given a polynomial as an arithmetic circuit $C(\mathbf{x})$, over a ring R, to efficiently test whether the C is identically zero. In this report we will focus on the case of R being a field. By efficient we mean the algorithm should run in $poly(size(C))$ many $\mathbb{F}$ operations. The problem is trivial if the polynomial is given as a vector of coefficients, for which we only need to check if any of the coefficient is non-zero. It also has an easy solution for univariate polynomials, which requires it to be evaluated at $degree + 1$ many points, and it is an identity iff all the evaluations are zero. This method doesn't work for multivariate polynomials, as there can be infinite solution for a simple bivariate polynomial (eg. $xy = 0$ over $\mathbb{R}$).

Definition 2.1.1. (Polynomial Identity Testing) Let $\mathcal{C}$ be a class of circuits having $size \leq s$, which computes polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ of degree $< d$. The PIT problem for this class $\mathcal{C}$ asks for a deterministic algorithm to test whether a polynomial f_C, computed by a circuit $\mathbb{C} \in \mathcal{C}$, is identically zero or not. The algorithm is considered efficient if it uses only $poly(s, n, d)\mathbb{F}$ operations.

2.2.1 The Randomized solution of PIT

It is notable that the problem of PIT has a very simple and elegant randomized solution thanks to the PIT lemma.

Lemma 2.2.1.1. (PIT Lemma) *(Schwartz-Zippel[Sch80])* Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial of total degree $d \geq 0$. Let S be any finite subset of $\mathbb{F}$, and let $\alpha_1, \ldots, \alpha_n$ be elements selected independently, uniformly and randomly from S. Then,

$$Pr_{\alpha_1,\ldots,\alpha_n \in S}[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}$$

The above lemma can be easily proved inductively, with the base case being the univariate case. The randomized algorithm is to simply pick any point from a large enough set, evaluate the input polynomial on it, and output 0 if the evaluation is zero, and non-zero otherwise. The probability that the random point is a root of the polynomial is small due to the PIT Lemma. Since, we cannot make a mistake on the non-zero part, we have $PIT \in coRP$. The problem of derandomizing PIT, so as to put it in P is still open. One trivial derandomization is to check $(d + 1)^n$ many points, but is inefficient. For more details on PIT, check [Sax09].

2.3 PseduoRandom Generators(PRGs)

A PseudoRandom distribution is a distribution (say over n bits) that is close to uniform distribution. By close, we mean that small (polynomial sized) circuits cannot distinguish these two. PRG's are used to obtain pseudorandom distributions by decreasing the number of random bits required.

For a given stretch $S : \mathbb{N} \to \mathbb{N}$, a $2^{O(n)}$-computable function $G : \{0, 1\}^* \to \{0, 1\}^*$ is an $S - prg$, if $\forall l$, $G : \{0, 1\}^l \to \{0, 1\}^{S(l)}$, and $\forall$ circuits C of size $\leq S(l)^3$

$$|Pr_{x \in U_l}[C(G(x)) = 1] - Pr_{x \in U_{S(l)}}[C(x) = 1]| < 0.1$$

PRGs can be used to derandomise classes:
If a S-prg exists then $\forall$ functions l

$$BP - TIME(S(l(n))) \subseteq DTIME(2^{l(n)}S(l(n)))$$

From the above expression, we see that if an exponential stretch PRG exists, then we can derandomise BPP. So $2^{\epsilon l}$-prg $\implies$ BPP=P

2.4 Hardness

There are 2 kinds of hardness for boolean functions:

- **Worst-case Hardness** For $f : \{0,1\}^* \to \{0,1\}$, $H_{wrs}(f)$ is the largest $S(n)$ st. $\forall$ circuit $C_n \in size(S(n))$,
$$Pr_{x \in U_n}[C_n(x) = f(x)] < 1$$
- **Average-case Hardness** $H_{avg}(f)$ is the largest $S(n)$ st. $\forall$ circuit $C_n \in size(S(n))$,
$$Pr_{x \in U_n}[C_n(x) = f(x)] < \frac{1}{2} + \frac{1}{S(n)}$$
- It can an be shown that a worst-case hard function gives also an average-case hard function.

2.5 NW-PRG Design

Nissan and Wigderson showd that we can construct a PRG if we are given a hard function

Theorem 2.1. *If $\exists f \in E$ with $H_{avg} \geq S(n)$, then $\exists S'(l)$-prg, where $S'(l) = S(n)^{0.01}$ for $\frac{100n^2}{logS(N)} \leq l \leq \frac{100(n+1)^2}{logS(n+1)}$*

So, $Hardness \implies PRGs \implies Derandomization$

3 Lemma 1.2

Lemma 1.2. $PIT \in P$ and $per \in Arth - P/poly \implies P^{per} \subseteq NP$.

Proof Idea

The main idea of the proof is to "guess" the small circuit for permanent using the self-reducibility of the permanent and then verify it using $PIT \in P$. We know that we have this expression:
$per_n(A) = \sum_{i \in [n]} A_{1i}.per(A'_{1i})$ where A'_{1i} is the corresponding minor.

Let C_n be arithmetic circuit corresponding to the per_n. We then have the following guess and verification protocol for obtaining the circuit for the permanent.

1. Given C_{n-1}, we guess the circuit for C_n as follows:
$$C_n(A) = \sum_{i \in [n]} A_{1i}.C_{n-1}(A'_{1i}) \ldots\ldots (1)$$
2. Use PIT for verifying whether the above expression is correct or not.
3. Repeat it for circuits C_{n-1} which we used for minors and so on.

Using this recursive guess and verify procedure, we can get a circuit $C_n(A) = per_n(A)$ by induction on n.
Now we show $P^{per} \subseteq NP$ Let $L \in P^{per}$.
We can guess C_n for per_n using the recursive procedure and then use this circuit C_n for per_n instead of the oracle. Now, $PIT \in P$, implies the entire verification is in P. Also, $per \in Arth - P/poly$, implies the guess that our machine need to do is poly-sized. So we get $per \in NP$ using the recursive procedure.
This gives $L \in NP \implies P^{per} \subseteq NP$

4 Lemma 1.3

Now we will have a look at the proof sketch of lemma 1.2, which we state it again for ease of read

Lemma 1.3. $EXP \subseteq P/poly \implies EXP = MA$

Proof Idea First we show $EXP \subseteq P/poly \implies EXP = \Sigma_2^p$. Consider $L \in EXP$, with an exp-time Turing Machine N. The idea will be to encode steps of N using a circuit and $\exists\forall$ quantifiers.

As N is an exp-time machine, on any input x,for any i, j, the j-th of $N(x)$'s i-th configuration can be computed in exponential time. As we have $EXP \subseteq P/poly$ from assumption, we have that there must be a poly-size circuit $C(x, i, j)$ that computes that computes this bit. Using this, we can easily capture the computation of N as follows:

$$x \in L \iff \exists C, \forall (i, j)[C(x, i, j) \to C(x, i+1, j) \text{ is a valid step of } N(x)]$$

Thus, we have $EXP \subseteq \Sigma_2^p$. We already know from class, $\Sigma_2^p \subseteq PSPACE = IP \subseteq EXP$, and we showed under our assumption $EXP \subseteq \Sigma_2^p$. This means all these classes are equal, and by our assumption we have

$$\Sigma_2 = PSPACE = IP = EXP \subseteq P/poly$$

As $EXP = IP$, we must have an IP protocol for L. TO show L in MA, we will convert this IP protocol into a single round protocol. We know from class that the Prover for an IP protocol can be simulated by a PSPACE machine. But, we also have in our assumption that $PSPACE \subseteq P/poly$. This means prover can also be simulated by a poly-size circuit family $\{C_n\}_{n \geq 1}$.

This gives us the 1-round protocol for checking $x \in L$ as:
Prover: Sends his circuit C_n, for $n = |x|$.
Verifier: Simulate the IP protocol using C_n as P.

Thus, the IP protocol for L becomes a MA protocol. Hence, $L \in MA$. Therefore, $EXP \subseteq MA$. While we already know that $MA \subseteq PSPACE \subseteq EXP$. This gives us the desired result as $EXP = MA$. □

5 Lemma 1.4

Now we will have a look at the proof sketch of lemma 1.3, which we state it again for ease of read

Lemma 1.4. $NEXP \subseteq P/poly \implies EXP = NEXP$

Proof Idea We will prove this by contradiction. Assume $NEXP \subseteq P/poly$ but $EXP \neq NEXP$. Therefore there must exist a Language $L \in NEXP \setminus EXP$, i.e. it is in NEXP but not in EXP. BY definition of NEXP we have a verifier R that runs in $exp(|x|^{10c})$-time such that

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

The idea of the problem is to use the fact that the certificate for most x will be hard for EXP. The hard function will be the boolean function $f_y : \{0,1\}^{n^c} \to \{0,1\}$ such that it's truth table is represented by y, i.e. if we arrange the $exp(n^c)$ outputs according to the lexicographical ordering of inputs, it forms the string y. To formally prove that f_y has circuit hardness we consider for all $D > 0$, the machine M_D as follows:

- construct tt of all circuits of size n^{100D}, with n^c input.
- if $\exists C, R(x, tt) = 1$ ACCEPT, else REJECT

The running time for each circuit guess will be $exp(n^c) + exp(n^{10c})$ where the first part is for constructing the truth table and later for running R. The number of circuits will be $exp(n^{100D})$. Thus the total running time will be $exp(n^{100D} + n^{10c})$, which is exponential. Thus, M_D is an exponential time machine.

As M_D is exp-time machine, it must not be able to solve L. Therefore, there must be infinitely many x on which M_D fails.This means that the certificates for these x will not be computable by circuits of size n^{100D}.Thus, $H_{wrs}(f_y) \geq n^{100D}$. Now using f_y in the NW PRG design, we get a l^D PRG.

Now if we have n as the input length of some string which is "hard" for the tt circuits, we can replace Arthur by a non-deterministic algorithm in poly$(n^d)2^{n^c}$ time that does not toss any random coins by using the prg obtained before (the 2^{n^c} factor is for calculating the n random bits deterministically)
This gives $L \in$ NTIME $(2^{n^{c'}})$ "infinitely often" with n-bit advice. Thus $EXP \subseteq$ NTIME $(2^{n^{c'}})$ "infinitely often" with n-bit advice
But $NEXP \subseteq P/poly$. Thus we have NTIME $(2^{n^{c'}}) \subseteq SIZE(n^{c'})$ for a constant c'. So $EXP \subseteq SIZE(n^{c'})$ infinitely often (the n bit advice can be hardcoded in the circuit).

So, $\exists$ c' such that every language in EXP can be decided on infinitely many inputs by a circuit family of size $n + n^{c'}$. Yet this can be ruled out using standard diagonalization. This gives a contradiction. So $NEXP \subseteq P/poly \implies EXP = NEXP$

6 Proof of the Main theorem

Now we prove the theorem statement:

$$PIT \in P \implies per \notin Arth - P/poly \text{ or } NEXP \nsubseteq P/poly$$

We combine the previously proved lemmas for this.

- Suppose $PIT \in P$, $per \in Arth - P/poly$ and $NEXP \subseteq P/poly$.
- Then from lemmas 2 and 3,$NEXP = EXP = MA \subseteq PH$.
- Also $PH \subseteq P^{per}$ (Toda's theorem)
- So $NEXP \subseteq P^{per}$
- Now as we have $PIT \in P$ and $per \in Arth - P/poly$, using lemma 1, we get $P^{per} \subseteq NP$
- Combining these two, we get $NEXP \subseteq NP$ which contradicts the non-deterministic time hierarchy theorem. Thus atleast of the assumptions is false which gives:

$$PIT \in P \implies per \notin Arth - P/poly \text{ or } NEXP \nsubseteq P/poly$$

7 Conclusion

- Derandomizing RP or BPP would give us circuit lower bounds for NEXP or for permanent.
- The results in the present paper do not rule out that ZPP = P can be proved without having to prove any circuit lower bounds first. This leaves some hope that unconditional derandomization of ZPP could be achieved.
- A thing to note would be that the same circuit lower bounds would be present if PIT is not in P but in some larger class (say SUBEXP) as the time hierarchy theorem would still get violated.
- We also don't need PIT for general circuits to be in P, as we saw in Lemma 1, a poly-time algorithm to test the symbolic determinant identity should be enough.

8 Open Problems

- $BPP = P$, $PIT \in P$, $per \notin Arth - P/poly$ and $NEXP \nsubseteq P/poly$ individually.(we believe all of these to be true)
- Does BPP=P imply circuit lower bounds for EXP (instead of NEXP) ?

References

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests meansproving circuit lower bounds. *ACM symposium on Theory of computing*, 2003.

[Sax09] Nitin Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS, 99:49–79*, 2009.

[Sch80] Jacob T Schwart. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 1980.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science: Vol. 5*, 2010.