Electronic Colloquium on Computational Complexity, Report No. 157 (2013)

Derandomizing Polynomial Identity over Finite Fields Implies Super-Polynomial Circuit Lower Bounds for NEXP

Bin Fu
Department of Computer Science
University of Texas–Pan American
Edinburg, TX 78539, USA
bfu@utpa.edu

November 10, 2013

Abstract

We show that derandomizing polynomial identity testing over an arbitrary finite field implies that NEXP does not have polynomial size boolean circuits. In other words, for any finite field $F(q)$ of size q, $\text{PIT}_q \in \text{NSUBEXP} \Rightarrow \text{NEXP} \not\subseteq \text{P/poly}$, where PIT_q is the polynomial identity testing problem over $F(q)$, and NSUBEXP is the nondeterministic subexpoential time class of languages. Our result is in contract to Kabanets and Impagliazzo's existing theorem that derandomizing the polynomial identity testing in the integer ring Z implies that NEXP does have polynomial size boolean circuits or permanent over Z does not have polynomial size arithmetic circuits.

1. Introduction

The polynomial identity testing problem (PIT) is to test whether a polynomial computed by an arithmetic circuit is identical to zero. PIT problem plays a significant role in the field of computational complexity. It is known that every polynomial computed by a polynomial size circuit can be determined if it is identical to zero by a polynomial time randomized algorithm [20, 26].

The results of Impagliazzo and Widgerson [10] suggested that every randomized polynomial time algorithm can be derandomized into a deterministic polynomial time algorithm. They proved that P = BPP if E contains any problem that requires $2^{\Omega(n)}$ size boolean circuits. It has been a long standing open problem in complexity theory to separate NEXP from BPP. Kabanets, Impagliazzo and Wigderson showed that derandomizing Promise-BPP implies NEXP $\not\subseteq$ P/poly [9]. Building upon the work of Kabanets, Impagliazzo and Wigderson [9], Kabanets and Impagliazzo [11] proved that to derandomize the polynomial identity testing problem in the integer ring, one must prove that NEXP has no polynomial size boolean circuits or permanent has no polynomial size arithmetic circuits. The proof of Kabanets and Impagliazzo's theorem was simplified by Aaronson and Melkebeek [1]. Many papers have been published toward the derandomization of the polynomial identity problems (see for examples, [3, 4, 6, 11, 12, 14–21, 26]).

We have not found any existing result that shows derandomization of PIT over a finite field implies NEXP $\not\subseteq$ P/poly. It is essential to identify the connection between derandomizing PIT over finite fields and complexity classes separation. In this paper, we study the implication of polynomial identity problem over finite fields to separations in computational complexity theory. Our results are derived without using permanent problem, and give a direct implication for complexity classes separation via derandomization of PIT.

We show that for any finite field F, if PIT over F is in NSUBEXP, then NEXP $\not\subseteq$ P/poly. We also show that if PIT over a finite field F is in NP, then $\text{NTIME}(n^{\log n}) \not\subseteq \text{PH} \cap \text{P/poly}$. It

ISSN 1433-8092

implies that if there exists a polynomial time deterministic algorithm for polynomial identity testing problem over any finite field F, then $\mathrm{NTIME}(n^{\log n}) \not\subseteq \mathrm{BPP}$.

Our proof is different from Kabanets and Impagliazzo's [11] work that is for PIT over Z. Their methods involve permanent that is #P-hard [24], and Toda's theorem $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$ [23]. Our method works for the PIT over any finite fields, but it does not imply that derandomizing PIT over Z separates NEXP from P/poly. Therefore, Kabanets and Impagliazzo's [11] work and this paper are complementary to each other to support the importance of derandomizing the polynomial identity testing problem, and its implication to nonuniform lower bounds.

2. Notations

A *boolean circuit* is a circuit with AND ($\bigwedge$), OR ($\bigvee$), and NEGATION ($\bar{x}$) gates with fan-in at most two, and no feedback. An *arithmetic circuit* is a circuit with $+, -$ and $*$ gates over a finite field. The *size* of a circuit $C(.)$ is the number of gates and is denoted by $|C(.)|$.

The *polynomial identity testing problem* is to test if a polynomial computed by an arithmetic circuit is identical to zero. We use PIT_q to represent the polynomial identity testing problem over the field $F(q)$ of size q. Let PIT_Z be the polynomial identity testing problem over the integers Z, which is an integral ring.

The basic knowledge of algebra can be found in standard algebra textbooks such as [8]. Every finite field $F(q)$ of size q has $q = p^k$ for some prime number p and integer k. For an element a in a finite field $F(q)$, its *order* is the least integer $r \geq 1$ with $a^r = 1$.

The permanent maps square matrices to values. Let $A = (a_{i,j})_{n\times n}$ be an $n \times n$ matrix over integers. Define permanent to be the function $\mathrm{perm}(A) = \sum_\sigma \prod_{i=1}^n a_{i,\sigma(i)}$, where σ is over all permutations of $1, 2, \cdots, n$.

Define $N = \{0, 1, 2, \cdots\}$ to be the set of nonnegative integers. Let $t(n) : N \to N$ be a nondecreasing function. Define $\mathrm{NTIME}(t(n))$ to be the class of languages accepted by nondeterministic Turing machines in time $O(t(n))$. For a function $f(n) : N \to N$, it is *time constructible* if given an integer n, $f(n)$ can be computed in $O(f(n))$ steps by a deterministic Turing machine. Theorem 1 is a separation of nondeterministic complexity classes due to Zak[25].

Theorem 1 ([25]). *If $t_1(.)$ and $t_2(.)$ are time-constructible nondecreasing functions from N to N, and $t_1(n+1) = o(t_2(n))$, then $\mathrm{NTIME}(t_2(n))$ is strictly contained in $\mathrm{NTIME}(t_1(n))$.*

Assume that $M(.)$ is an oracle Turing machine. A decision computation $M^A(x)$ returns either 0 or 1 when the input is x and oracle is A. For a class C of languages, we use $\mathrm{NP}^C = \mathrm{NP}_\mathrm{T}(C)$ to represent the class of languages that can be reducible to the languages in C via polynomial time nondeterministic Turing reductions. Define $\mathrm{NEXP} = \cup_{c=1}^\infty \mathrm{NTIME}(2^{n^c})$ and $\mathrm{NP} = \cup_{c=1}^\infty \mathrm{NTIME}(n^c)$.

Let PH be the class of polynomial time hierarchy [22] $\mathrm{PH} = \cup_{i=1}^\infty \sum_i^\mathrm{P}$, where $\sum_1^\mathrm{P} = \mathrm{NP}$, and $\sum_{i+1}^\mathrm{P} = \mathrm{NP}^{\sum_i^\mathrm{P}}$ for all $i \geq 1$. Define the subexponential time nondeterministic class to be $\mathrm{NSUBEXP} = \cap_{\epsilon>0} \mathrm{NTIME}(2^{n^\epsilon})$. Define P/poly to be the class of languages that have nonuniform polynomial size circuits. BPP, which stands for bounded-error probabilistic polynomial time, is the class of decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of at most 1/3 for all instances.

An *instance of* 3SAT is a 3CNF that is a conjunction of clauses of at most three literals. For example, $(x_1 \bigvee \overline{x_2} \bigvee x_3) \bigwedge (\overline{x_1} \bigvee \overline{x_2} \bigvee x_3) \bigwedge (x_1 \bigvee \overline{x_4} \bigvee x_5)$. A formula is said to be satisfiable if it can be made true by assigning appropriate logical values (i.e. TRUE (1), FALSE(0)) to its variables. The 3SAT is, given a 3CNF, to check whether it is satisfiable. It is well known that 3SAT is NP-complete problem [5]. The number of variables of a 3SAT instance and its length is polynomially related.

3. Our Results

Theorem 2 is the main theorem of this paper. It will be proved in Section 5. Theorem 2 is stated in a format so that we have a self-contained proof. Some corollaries that involve some existing results are stronger than the main theorem.

Theorem 2. *Let $t(n)$ and $t'(n)$ be time constructible nondecreasing superpolynomial functions from N to N with $t'(n+1) = o(t(n))$. Let $h(n)$ be an nondecreasing function from N to N such that for every fixed $c > 0$, $h(n^c) + n^c \leq t'(n)$ for all large n. Let $F(q)$ be a finite field of size q. If $\mathrm{PIT}_q \in \mathrm{NTIME}(h(n))$, then $\mathrm{NTIME}(t(n)) \not\subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{P/poly}$.*

Using the main theorem, we have some corollaries. Their proofs need to combine Theorem 2 with some existing well known theorems in the computational complexity theory.

Corollary 3. *Let $t(n)$ and $t'(n)$ be time constructible nondecreasing superpolynomial functions from N to N with $t'(n+1) = o(t(n))$. Let $h(n)$ be a nondecreasing function from N to N such that for every fixed $c > 0$, $h(n^c) + n^c \leq t'(n)$ for all large n. Let $F(q)$ be a finite field of size q. If $\mathrm{PIT}_q \in \mathrm{NTIME}(h(n))$, then $\mathrm{NTIME}(t(n)) \not\subseteq \mathrm{PH} \cap \mathrm{P/poly}$.*

Proof: Assume $\mathrm{NTIME}(t(n)) \subseteq \mathrm{PH} \cap \mathrm{P/poly}$. By Karp and Lipton's thereom [13], we have $\mathrm{PH} = \sum_2^P = \mathrm{NP}^{\mathrm{NP}}$. It follows from Corollary 3. ∎

Corollary 4. *If $\mathrm{PIT}_q \in \mathrm{NSUBEXP}$ for a finite field $F(q)$, then $\mathrm{NEXP} \not\subseteq \mathrm{P/poly}$.*

Proof: Assume that $\mathrm{PIT}_q \in \mathrm{NSUBEXP}$ and $\mathrm{NEXP} \subseteq \mathrm{P/poly}$. By Impagliazzo, Kabanets, and Wigderson's theorem [9], $\mathrm{NEXP} = \mathrm{PH}$. We have a contradiction by Theorem 2. ∎

Corollary 5. *If $\mathrm{PIT}_q \in \mathrm{NSUBEXP}$ for a finite field $F(q)$, then $\mathrm{NEXP} \neq \mathrm{BPP}$.*

Proof: Assume that $\mathrm{PIT}_q \in \mathrm{NSUBEXP}$ and $\mathrm{NEXP} = \mathrm{BPP}$. It is well known that Adleman [2] proved $\mathrm{BPP} \subseteq \mathrm{NP}^{\mathrm{NP}}$. We have a contradiction by Theorem 2. ∎

Corollary 6. *If $\mathrm{PIT}_q \in \mathrm{NP}$ for a finite field $F(q)$, then $\mathrm{NTIME}(n^{\log n}) \not\subseteq \mathrm{BPP}$.*

Proof: It is similar to the proof of Corollary 5. ∎

4. Overview of Our Method

In this section, we give a brief review of our method. The main theorem will be proved by contradiction. A special version of our main theorem is formulated as $\mathrm{PIT}_2 \in \mathrm{NP} \Rightarrow \mathrm{NEXP} \not\subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{P/poly}$.

Assume $\mathrm{PIT}_2 \in \mathrm{NP}$ and $\mathrm{NEXP} \subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{P/poly}$. Let K be a complete language of the class NEXP and let 3SAT be computed by a polynomial size circuit $C(.)$. Our main technical contribution is a method that transforms a boolean circuit $C(.)$ into an arithmetic circuit $A^*_C(.)$ over a finite field $F(q)$ such that $C(.)$ decides 3SAT if and only if $A^*_C(.)$ is identical to zero.

As the 3SAT problem is not arithmetically defined as permanent. If each instance of 3SAT is encoded as a binary string that will be easy to decode, then there are some binary strings that do not encode valid instances of 3SAT. In other words, a mapping from instances of 3SAT to binary strings may not be both one-one and onto. We construct a special polynomial size arithmetic function $G(Y)$ that is zero if Y is not an instance of 3SAT, and nonzero otherwise. For an instance $f(x_1, x_2, \cdots, x_n)$ of 3SAT, it is satisfiable if and only if at least one of $f(0, x_2, \cdots, x_n)$ and $f(1, x_2, \cdots, x_n)$ is satisfiable. This recursive relation is also converted into an arithmetic circuit $A^*_C(.)$ as PIT problem to verify whether $C(.)$ decides 3SAT. The arithmetic circuit $A^*_C(.)$ is expressed as $H(f)G(f)$. The arithmetic circuit $H(f)$ is used to verify the recursive relationship of circuit $C(.)$ for deciding 3SAT. As the

input of $A^*_C(.)$ has many cases that do not encode any instance of 3SAT, the function $G(.)$ has a value zero to pass the identity testing among those cases.

We will show that K can be computed by $M^{3\mathrm{SAT}}(.)$ for a polynomial time oracle Turing machine $M(.)$. A polynomial time nondeterministic computation will be derived to compute K. A circuit $C(.)$ will be guessed and is checked via converting to PIT_2, which is verified again in a nondeterministic polynomial time. Thus, we have $K \in \mathrm{NP}$. This contradicts the well know nondeterministic computational complexity hierarchy, which is stated in Theorem 1 and implies $\mathrm{NEXP} \neq \mathrm{NP}$.

This approach lets us obtain lower bound for NEXP under the existence of the derandmoization of PIT over an arbitrary finite field without using permanent that is #P hard. This paper has almost self-contained proof for the main theorem. A reader is able to understand our main theorem just by knowing Theorem 1 and that 3SAT is NP-complete [5], which can be found in a standard textbook of theory of computation.

5. Proof of Main Theorem

In this section, we derive our main theorem. Some lemmas are provided to convert boolean circuits into arithmetic circuits. It is divided into several subsections to prove the main theorem.

5.1. From Boolean Circuits to Arithmetic Circuits

In this section, we show how to transform boolean circuits into an arithmetic circuits.

For a finite field $F(q)$, we have the following property that is often called "Fermat Little Theorem" for the case that q is a prime number. Its proof can be found in a standard algebra textbook. For completeness, its proof is included here.

Lemma 7. *Let $F(q)$ be a finite field. For any $a \in F(q) - \{0\}$, $a^{q-1} = 1$.*

Proof: Assume that F is a finite field. Let $[a] = \{a, a^2, a^3, \cdots, \}$ be the set of elements in $F(q)$ generated by a. $([a], .)$ forms a subgroup of $F(q)^* = F(q) - \{0\}$, where "." is the multiplication operation over field $F(q)$. Therefore, the order r of a is the size of $[a]$. Therefore, r is a divisor of $q-1$. So, $a^r = a^{q-1} = 1$.

∎

We give Lemma 9 to convert an instance of 3SAT into a binary string. We give Definition 8 to normalize the input of an instance of 3SAT.

Definition 8. Assume that an instance $C_1 \bigwedge C_2 \bigwedge \cdots \bigwedge C_m$ of 3SAT of n variables and satisfies the conditions below:

1. each C_i has at most three literals,
2. no variable appears in two literals of the same clause,
3. all clauses have a different set of literals, and
4. its n variables are $x_1, x_2, \cdots, x_n$ that are indexed from 1 to n

We have the following definitions:

- Define $E_l(x_i) = (i, 1)$ and $E_l(\overline{x_i}) = (i, 0)$.
- Define $E_c((y_i \bigvee y_j \bigvee y_k))) = (E_l(y_i), E_l(y_j), E_l(y_k))$ for each clause $(y_i \bigvee y_j \bigvee y_k)$.
- Define the normalized representation of $C_1 \bigwedge C_2 \bigwedge \cdots \bigwedge C_m$ of 3SAT to be $(E_c(C_1), E_c(C_2), \cdots, E_c(C_m))$.
- The logical value TRUE (1) is treated as special instance of 3SAT, and we define its normalized representation to be (1).

- The logical value FALSE (0) is treated as special instance of 3SAT, and we define its normalized representation to be (0).

Lemma 9. *Assume an instance f of* 3SAT *is a normalized representation as Definition 8.*

i. There is a polynomial time encoding method $E(.)$ such that given an instance f of at most n variables of 3SAT*, $E(n, f)$ is a 0,1-string of length $8n^4$.*

ii. There is a polynomial time decoding method $D(.)$ such that given a 0,1-string $s = E(n, f)$ for some instance f with at most n variables of 3SAT*, $D(s) = f$.*

iii. There is a polynomial time algorithm $H(.)$ such that $H(1^n)$ generates a polynomial size boolean circuit $V_n(.)$ such that given a 0,1-string s of length $8n^4$, $V_n(s) = 1$ if $s = E(n, f)$ for some instance of at most n variables of 3SAT*, and 0 otherwise.*

Proof: We prove the three statements below:

Statement i: Given a normalized representation an instance of 3SAT, just replace each symbol with ASCI table to transfer it into a binary string. Append 10^k for some k so that the total length is exactly equal to $8n^4$. Each 3CNF instance has at most $24\binom{n}{3} < 4n^3$ different clauses. $8n^4$ binary bits are enough to encode any 3CNF instance of at most n variables.

Statement ii: It is straight forward to decode the binary string into an instance of 3SAT by using the ASCI table.

Statement iii: With a polynomial time, we can check if a binary string is a binary string to encode an valid instance of a 3SAT. It can be converted into a polynomial size boolean circuit. ∎

Definition 10. Let $C(x_1, x_2, \cdots, x_n) : \{0,1\}^n \to \{0,1\}$ be a boolean circuit, and $A(y_1, y_2, \cdots, y_n) : F(q)^n \to F(q)$ be an arithmetic circuit over a finite field $F(q)$. We say $C(.)$ and $A(.)$ are *equivalent* if for any $a_1, a_2, \cdots, a_n \in \{0,1\}$, $C(a_1, a_2, \cdots, a_n) = 0 \Leftrightarrow A(a_1, a_2, \cdots, a_n) = 0$ in the field $F(q)$; and $C(a_1, a_2, \cdots, a_n) = 1 \Leftrightarrow A(a_1, a_2, \cdots, a_n) = 1$ in the field $F(q)$.

The following Lemma 11 shows how a boolean circuit is converted into an equivalent arithmetic circuit with a similar size.

Lemma 11. *For any boolean circuit $C(.)$, then there is an equivalent arithmetic circuit $A_C(.)$ over a field $F(q)$ such that $|A_C(.)| = O(|C(.)|)$. Furthermore, $A_C(.)$ can be constructed from $C(.)$ in a polynomial time of $|C(.)|$.*

Proof: We just show how to simulate the three AND, OR, and NOT gates in a boolean circuit with arithmetic operations. The arithmetic circuit is constructed by simulating the boolean circuit $C(.)$ gate by gate. For an AND operation $a \bigwedge b$, it can be converted into product $a \cdot b$ over $F(q)$. For an OR operation $a \bigvee b$, it can be converted into $1 - (1 - a)(1 - b)$. For an NOT operation $\neg a$, it is converted into $1 - a$. Since each gate in $C(.)$ is transformed into $O(1)$ gates in $A_C(.)$, we have $|A_C(.)| = O(|C(.)|)$. It is easy to see that the total time to construct $A_C(.)$ is a polynomial time of $|C(.)|$. ∎

Definition 12. Let f be a normalized representation of an instance of 3SAT. Define $E(f)$ to be the *normalized binary encoding* of f, where $E(.)$ is as defined in Lemma 9. Define $one_n = E(n, (1))$ and $zero_n = E(n, (0))$ for the normalized binary representation of true and false respectively, where $E(.)$ is given in Lemma 9.

Lemma 13. *Let $F(q)$ be a fixed finite field. Then there is a polynomial time algorithm that given an unary integer 1^n, it generates an arithmetic circuit $G_n(x_1, x_2, \cdots, x_m)$ with $m = 8n^4$ such that*

i. $G_n(x_1, x_2, \cdots, x_m) = 0$ *if at least one of* $x_1, x_2, \cdots, x_m$ *is not in* $\{0, 1\}$*;*

ii. $G_n(x_1, x_2, \cdots, x_m) = 0$ *if* $x_1 x_2 \cdots x_m$ *is not a normalized binary encoding of an instance of* 3SAT *with at most* n *variables; and*

iii. $G_n(x_1, x_2, \cdots, x_m) \neq 0$ *if* $x_1 x_2 \cdots x_m$ *is a normalized binary encoding of an instance of* 3SAT *with at most* n *variables.*

Proof: By Lemma 9, we let $V_n(f)$ be a boolean circuit such that $V_n(f) \neq 0$ if and only if f is a normalized binary encoding of an instance of 3SAT. Let $A_V(.)$ be the arithmetic circuit defined by Lemma 11.

Define $R(x) = 1 - (x(x-1))^{q-1}$. It is easy to see that $R(x) \neq 0$ if and only if $x \in \{0, 1\}$ by Lemma 7.

Finally, we define $G_n(x_1, x_2, \cdots, x_m) = R(x_1)R(x_2)\cdots R(x_m)A_{V_n}(x_1, x_2, \cdots, x_m)$. It is easy to see that $G_n(.)$ satisfies expected properties. ∎

Lemma 14. *Assume that each input instance of* 3SAT *is a normalized binary encoding (see Definition 12. Then there is a polynomial time algorithm such that given* 1^n*, it generates* $n^{O(1)}$ *size arithmetic circuits* $S_{n,0}(.)$*, and* $S_{n,1}(.)$ *such that the following are satisfied:*

i. $S_{n,0}(f)$ *generates a normalized binary encoding for* $g(0, x_2, \cdots, x_k)$ *if* f *is a normalized binary encoding of a* 3SAT *instance* $g(x_1, x_2, \cdots, x_k)$ *with* $0 \leq k \leq n$*;*

ii. $S_{n,1}(f)$ *generates a normalized binary encoding for* $g(1, x_2, \cdots, x_k)$ *if* f *is a normalized binary encoding of a* 3SAT *instance* $g(x_1, x_2, \cdots, x_k)$ *with* $0 \leq k \leq n$*; and*

iii. $S_{n,i}(f) = f$ *for* $i \in \{0, 1\}$ *and* $f \in \{zero_k, one_k\}$ *with* $0 \leq k \leq n$*.*

Proof: It is easy to see that there is a polynomial time algorithm to generate the formulas $g(0, x_2, \cdots, x_k)$, $g(1, x_2, \cdots, x_k)$ with $0 \leq k \leq n$. Thus, we can get a boolean circuits to generate them. By Lemma 11, we can get the equivalent arithmetic circuits to generate them respectively. ∎

5.2. From Arithmetic Circuits to PIT

In this section, we show how to convert the arithmetic expressions developed in the last section and a circuit for 3SAT into a PIT problem.

The following Lemma 15 shows how to use the PIT problem over a finite field to check if a boolean circuit decides 3SAT. It transform a boolean circuit into an arithmetic circuit in a polynomial number of steps.

Lemma 15. *Let* $F(q)$ *be a fixed finite field. Then there is a polynomail time algorithm such that given a circuit* $C_n(.)$*, it generates another arithmetic circuit* $A^*_{C_n}(.)$ *over a finite field* $F(q)$ *such that* $C_n(.)$ *decides instances for* 3SAT *with at most* n *variables if and only if* $A^*_{C_n}(.)$ *is identical to zero.*

Proof: We assume that all instances of 3SAT with at most n variables have normalized binary encoding of length $8n^4$ as input for $C_n(.)$. Let $S_{n,0}(.)$ and $S_{n,1}(.)$ be defined as in Lemma 14. Let $A_{C_n}(.)$ be the arithmetic circuit that is equivalent to $C_n(f)$ by Lemma 11. Let one_n be the normalized binary encoding of logical constant TRUE (1), and let $zero_n$ be the normalized binary encoding of logical constant FALSE (0) (see Definition 12). Let y_0, y_1, y_2 be new variables that do not appear in $A_{C_n(.)}$. We have the arithmetic circuit

$$\begin{aligned} H(f, y_0, y_1, y_2) = \quad & y_0(A_{C_n}(one_n) - 1) + y_1 A_{C_n}(zero_n) + \\ & y_2(A_{C_n}(f) - (1 - (1 - A_{C_n}(S_{n,0}(f)))(1 - A_{C_n}(S_{n,1}(f))))). \end{aligned}$$

Let $G_n(.)$ be the arithmetic circuit defined by Lemma 13. Define $A^*_{C_n}(f, y_0, y_1, y_2) = H(f, y_0, y_1, y_2)G_n(f)$.

Assume that circuit $C_n(.)$ decides 3SAT for all instance of at most n variables, and takes the normalized binary encoding of length $8n^4$ as input. For each normalized binary encoding f of an instance of 3SAT, we have $H(f, y_0, y_1, y_2) = 0$. This is because recursive relation for each decider of 3SAT. If f is not a normalized binary encoding of a valid instance of 3SAT, we have $G_n(f) = 0$. Therefore, $A^*_{C_n}$ is identical to zero.

Assume that $A^*_{C_n}$ is identical to zero. We need to verify that $C_n(.)$ is a circuit for 3SAT. For each valid instance f with at most n variables of 3SAT, we have $G_n(f) \neq 0$ by Lemma 13. Thus, $H(f, y_0, y_1, y_2) = 0$. It confirms the $C_n(.)$ satisfies the recursive relation for a 3SAT decider. For each instance $g(x_1, \cdots, x_n)$ with n variables for 3SAT, let $a_1, \cdots, a_k \in \{0, 1\}$ be an assignment for its first k variables with $0 \leq k \leq n$. We can still find a normalized binary encoding f_k for $g(a_1, \cdots, a_k, x_{k+1}, \cdots, x_n)$ and f_k has length $8n^4$. Since $G_n(f_k) \neq 0$ and $H(f_k, y_0, y_1, y_2) = 0$, $C_n(.)$ satisfies the recursive relation for a 3SAT decider at the cases $g(a_1, \cdots, a_k, x_{k+1}, \cdots, x_n)$ for all $0 \leq k \leq n$..

We have that $C_n(.)$ is a circuit for 3SAT if and only if $A^*_{C_n}(f, y_0, y_1, y_2)$ is zero since it verifies if the circuit $C_n(.)$ satisfies the recursion for 3SAT instance satisfiability. ∎

5.3. From Derandomization to Separations

In this section, we show that derandomizing PIT over a finite field implies separation of computational complexity classes. The proof of main theorem is given here.

Proof: [Theorem 2] Assume NTIME($t(n)$) $\subseteq$ NP$^{\text{NP}} \cap$ P/poly. Let K be an arbitrary language of NTIME($t(n)$) under polynomial time many-one reduction. Let $M^{\text{3SAT}}(.)$ be a polynomial time nondeterministic Turing machine to accept K, and runs in a polynomial time $p(n)$ that is nondecreasing function from N to N. Let $N(.)$ be an $O(h(n))$ time nondeterministic Turing machine that decides PIT$_q$. We have the following nondeterministic algorithm for K.

Nondeterministic Algorithm for K
Input x of length n,

1. Guess a circuit $C_{p(n)}(.)$ for deciding the instances of variables at most $p(n)$ for 3SAT;
2. Generate an arithmetic circuit $A^*_{C_{p(n)}}(.)$ PIT$_q$ problem to verify $C_{p(n)}(.)$ by Lemma 15;
3. Run $N(A^*_{C_{p(n)}}(.))$ nondeterministically to decide if $A^*_{C_{p(n)}}(.)$ is identical to zero;
4. Nondeterministically select a path P^* in $M^{\text{3SAT}}(x)$;
5. If step 3 is successful, use $A^*_{C_{p(n)}}(.)$ to answer all the queries to 3SAT in path P^*;
6. Output yes, if P^* accepts;

End of Algorithm

Since $M(.)$ runs in polynomial time $p(n)$, the instance of queries made by $M(.)$ has at most $p(n)$ variables. Since NTIME($t(n)$) $\in$ P/poly implies 3SAT$\in$ P/poly, there is a polynomial size boolean circuit $C_{p(n)}(.)$ to decide 3SAT for all instances with at most $p(n)$ variables. Let $q_1(n)$ be a polynomial with $|C_{p(n)}(.)| \leq q_1(n)$. We have an arithmetic circuits $A^*_{C_{p(n)}}(.)$ that is equivalent with $C_{p(n)}(.)$ by Lemma 11. We also have $|A^*_{C_{p(n)}}(.)| \leq q_2(n)$ for some polynomial $q_2(n)$. Step 3 in the algorithm takes $O(h(q_2(n)))$ nondeterministic steps. For some constant $c > 0$, the entire computation is in $O(h(n^c) + n^c) = O(t'(n))$ nondeterministic steps.

The nondeterministic algorithm above shows that K is in NTIME($t'(n)$). Since K is an arbitrary language in NTIME($t(n)$), we have NTIME($t(n)$) $\subseteq$ NTIME($t'(n)$). This contradicts the well known hierarchy theorem (see Theorem 1) for nondeterministic computation classes. ∎

6. Generalization to Bounded Depth Circuits

In this section, we consider the problem for the PIT with bounded depth arithmetic circuits, and its connection to the super-polynomial lower bounds of bounded depth boolean circuits. It is an open problem to prove NEXP $\not\subseteq$ NC1/poly, where NC1/poly is the class of languages that have polynomial size $O(\log n)$-bounded depth boolean circuits.

Definition 16. Let $d(n)$ be a function from N to N. Let $F(q)$ be a finite field of size q. Define $\text{PIT}_q(d(n))$ to be the polynomial identity testing problem that decides if a polynomial computed by an arithmetic circuit of depth at most $d(n)$ is identical to zero over field $F(q)$.

Definition 17. Let $d(n)$ be a function from N to N. A $d(n)$-*bounded depth* boolean circuits is the class of boolean circuits that consists of AND, OR, and NOT gates with unbounded fan-in for AND and OR gates. Define *Depth*($d(n)$)-*PC* to be the class of languages that have polynomial size $d(n)$-bounded depth boolean circuits.

Theorem 18. *Let $t(n)$ and $t'(n)$ be time constructible nondecreasing superpolynomial functions from N to N with $t'(n+1) = o(t(n))$. Let $h(n)$ be a nondecreasing function from N to N such that for every fixed $c > 0$, $h(n^c) + n^c \leq t'(n)$ for all large n. Let $F(q)$ be a finite field of size q. Let $d(n)$ be a function from N to N with $d(n) \geq \log n$. If* $\text{PIT}_q(O(d(n)) \in \text{NTIME}(h(n))$, *then* $\text{NTIME}(t(n)) \not\subseteq \text{NP}^{\text{NP}} \cap$ *Depth*($d(n)$)-*PC.*

Proof: [Sketch] The proof is similar to that of Theorem 2. We need to have a similar lemma like Lemma 11 to show that a bounded depth k boolean circuit has an equivalent bounded depth $O(k)$ arithmetic circuit. With $d(n) \geq \log n$, we also have a lemma similar to Lemma 15 such that a bounded $d(n)$ depth boolean circuit for 3SAT can be converted into a $\text{PIT}_q(O(d(n)))$ problem to verify it. This is because $S_{n,0}(.)$ and $S_{n,1}(.)$ have polynomial size $O(\log n)$-depth boolean circuits. ∎

Corollary 19. *Let $t(n)$ and $t'(n)$ be time constructible nondecreasing superpolynomial functions from N to N with $t'(n+1) = o(t(n))$. Let $h(n)$ be an nondecreasing function from N to N such that for every fixed $c > 0$, $h(n^c) + n^c \leq t'(n)$ for all large n. Let $d(n)$ be a function from N to N with $d(n) \geq \log n$. Let $F(q)$ be a finite field of size q. If* $\text{PIT}_q(O(d(n))) \in \text{NTIME}(h(n))$, *then* $\text{NTIME}(t(n)) \not\subseteq \text{PH} \cap$ *Depth*($O(d(n))$)-*PC.*

Proof: Assume $\text{NTIME}(t(n)) \subseteq \text{PH} \cap \text{Depth}(O(d(n)))$-PC. By Karp and Lipton's thereom [13], we have $\text{PH} = \sum_2^P = \text{NP}^{\text{NP}}$. It follows from Theorem 18. ∎

Corollary 20. *If* $\text{PIT}_q(O(\log n)) \in \text{NSUBEXP}$ *for a finite field $F(q)$, then* $\text{NEXP} \not\subseteq \text{NC1/poly}$.

Proof: Assume that $\text{PIT}_q \in \text{NSUBEXP}$ and $\text{NEXP} \subseteq \text{NC1/poly} = \text{Depth}(O(\log n))$-PC. By Impagliazzo, Kabanets, and Wigderson's theorem [9], NEXP = PH. We have a contradiction by Theorem 18. ∎

7. Conclusions

The result developed in this shows that derandomizing the PIT in any finite field implies NEXP does not have nonuniform polynomial size circuits. It gives right motivation to study the derandomization of PIT in finite fields that the computational complexity community has spent much efforts. We hope that the results in this paper brings a tool to achieve the separation of NEXP from BPP via derandomizing PIT_p for a prime number p such as 2. Since there exists an oracle to collapse NEXP to BPP by Heller [7], separating NEXP from BPP requires a new way to go through the barrier of relativization.

Another interesting open problem is if derandomizing PIT over Z implies NEXP $\not\subseteq$ P/poly (In other words, $\mathrm{PIT}_Z \in$ NSUBEXP $\Rightarrow$ NEXP $\not\subseteq$ P/poly?). Our technology fails on integers Z. We cannot obtain a similar result as Lemma 13 over the ring Z of integers.

Acknowledegments: The author is grateful to Bohan Fan, Cynthia Fu, and Feng Li for their proofreading and suggestions for an earlier version of this paper. This research is supported in part by NSF Early Career Award CCF-0845376.

References

[1] S. Aaronson and D. van Melkebeek. A note on circuit lower bounds from derandomization, electronic colloquium on computational complexity. Technical Report TR10-105, 2010.

[2] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.

[3] M. Agrawal and S. Biswas. Primality and identity testing via chinese remaindering. *J. ACM*, 50:429–433, 2003.

[4] Z.-Z. Chen and M.-Y. Kao. Reducing randomness via irrational numbers. *SIAM Journal on Computing*, 29:1568–1576, 2000.

[5] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[6] Z. Dvir and A. Shpilka. Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. *SIAM J. Comput.*, 3:14041434, 2007.

[7] H. Heller. On relativized exponential and probabilistic complexity classes. *Inf. & Comp.*, 71:231–243, 1986.

[8] T. Hungerford. *Algebra*. Springer-Verlag, 1974.

[9] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *JCSS*, 55(1):672–694, 2002.

[10] R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th STOC*, pages 220–229, 1997.

[11] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[12] Z. S. Karnin and A. Shpilka:. Black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. *Electronic Colloquium on Computational Complexity, TR07-042*, 2007.

[13] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 302–309, 1980.

[14] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. computational complexity. *Computational Complexity*, 16:115138, 2007.

[15] A. Klivans and D. Spielman. Randomness efficient identity testing. In *Proceedings of the 33rd Symposium on Theory of Computing*, pages 216–223, 2001.

[16] D. Lewin and S. P. Vadhan. Checking polynomial identities over any field: Towards a derandomization? In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 438–447, 1998.

[17] R. J. Lipton and N. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 756–760, 2003.

[18] R. Raz and A. Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14:119, 2005.

[19] N. Saxena. Diagonal circuit identity testing and lower bound. In *Proceedings of the International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 5125*, page 6071, 2008.

[20] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27:701–717, 1980.

[21] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 507–516, 2008.

[22] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[23] S. Toda. PP is as hard as the polynomial-time hierarchy. *SICOMP*, 20(5):865–877, 1991.

[24] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[25] S. Zak. A turing machine hierarchy. *Theoretical Computer Science*, 26:327333, 1983.

[26] R. Zippel. Probabilistic algorithms for sparse polynomials. In *ISSAC'79: Proc. Int'l Symposium on symbolic and algebraic computation, Lecture notes in computer science*, pages 216 – 226, 1979.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de