Non-Deterministic Exponential Time has Two-Prover Interactive Protocols

László Babai*† Lance Fortnow‡ Carsten Lund§

University of Chicago¶

Abstract

We determine the exact power of two-prover interactive proof systems introduced by Ben-Or, Goldwasser, Kilian, and Wigderson (1988). In this system, two all-powerful noncommunicating provers convince a randomizing polynomial time verifier in polynomial time that the input x belongs to the language L. It was previously suspected (and proved in a relativized sense) that $coNP$-complete languages do not admit such proof systems. In sharp contrast, we show that the class of languages having two-prover interactive proof systems is nondeterministic exponential time.

After the recent results that all languages in $PSPACE$ have single prover interactive proofs (Lund, Fortnow, Karloff, Nisan, and Shamir), this represents a further step demonstrating the unexpectedly immense power of randomization and interaction in efficient provability. Indeed, it follows that multiple provers with coins are strictly stronger than without, since $NEXP \neq NP$. In particular, for the first time, provably polynomial time intractable languages turn out to admit "efficient proof systems" since $NEXP \neq P$.

We show that to prove membership in languages in EXP, the honest provers need the power of EXP only. A consequence, linking more standard concepts of structural complexity, states that if EXP has polynomial size circuits then $EXP = \Sigma_2^P = MA$.

The first part of the proof of the main result extends recent techniques of polynomial extrapolation of truth values used in the single prover case. The second part is a verification scheme for multilinearity of an n-variable function held by an oracle and can be viewed as an independent result on *program verification*. Its proof rests on combinatorial techniques including the estimation of the expansion rate of a graph.

*Supported by NSF Grant CCR-8710078.

†University of Chicago and Eötvös University, Budapest, Hungary H-1088.

‡Supported by NSF Grant CCR-9009936.

§Supported by a fellowship from the University of Århus.

¶1100 E. 58th Street, Chicago, IL 60637.

1 Introduction

The concept of NP has been introduced as a model of languages with efficient proof of membership (Cook (1971), Levin (1973)). As an extension of this concept, single prover interactive proofs have been introduced by Babai (1985) and Goldwasser, Micali, Rackoff (1985). The power of this extension has not been recognized until very recently, when combined work of Lund, Fortnow, Karloff, Nisan [21], and Shamir [27] has shown that every language in $PSPACE$ has an interactive proof. This actually means $IP = PSPACE$ because the inclusion $IP \subseteq PSPACE$ has been known for long (see Papadimitriou [24]).

This paper looks at the class MIP of languages that have multiple-prover interactive proof systems. Ben-Or, Goldwasser, Kilian and Wigderson created the model of multiple provers consisting of provers that cannot communicate and no prover can listen to conversations between the verifier and other provers. BGKW showed in this model that all languages in NP have perfect zero-knowledge multi-prover proof systems, a statement not true for one prover unless the polynomial-time hierarchy collapses (Fortnow [13]). They also show that only two provers are necessary for any language in MIP. Recently, building on the work of Lund-Fortnow-Karloff-Nisan and Shamir, Cai [10] has shown that $PSPACE$ has one-round interactive proofs with two provers.

Surprisingly, the proof that $PSPACE$ contains IP does not carry through for multiple-prover proof systems. The best upper bound known, due to Fortnow, Rompel, and Sipser [15], is non-deterministic exponential time: Guess the strategies of the provers and check for all possible coin tosses of the verifier. This paper shows this upper bound is tight.

Theorem 1.1 *$MIP = NEXP$. In other words, the set of languages with two-prover interactive proof systems is exactly the set of languages computable in non-deterministic exponential time.*

BGKW [7] in fact shows that all languages that

have multi-prover proof systems have perfect zero-knowledge multi-prover proof systems with no cryptographic assumptions. Combining this with our result shows that all of $NEXP$ has perfect zero-knowledge multi-prover proof systems.

Note: In this paper, the term *exponential* always means $2^{p(n)}$ for some polynomial $p(n)$. In particular, $EXP = \bigcup_{k \geq 1} TIME(2^{n^k})$. The nondeterministic version $NEXP$ is defined analogously.

Theorem 1.1 is in sharp contrast to what has previously been expected. Indeed, Fortnow, Rompel and Sipser have shown that relative to some oracle, even the class $coNP$ does not have multi-prover interactive proof systems.

We should also point out that it follows from our result that multiple provers with coins are *provably* strictly stronger than without, since $NEXP \neq NP$ (Seiferas, Fischer, Meyer [26]). In particular, for the first time, provably polynomial time intractable languages turn out to admit "efficient proof systems" since $NEXP \neq P$. (No analogous claims can be made about single prover interactive proof systems, as long as the question $P \neq PSPACE$ remains unresolved.)

Unfortunately if we take BPP to be the class of "tractable" languages, we are no longer able to make the intractability claim since it is not known whether or not $BPP = NEXP$. Indeed, there exists an oracle that makes these two classes collapse (Heller [18]), thus eliminating the hopes for an easy separation. (An earlier version of this paper had a fallacious proof showing $NEXP$ properly contains BPP. We are grateful to Russell Impagliazzo for pointing out the error.)

Theorem 1.1 and the result that $IP = PSPACE$ have the same flavor of replacing universal quantification by probabilistic quantification. $PSPACE$ is exactly the class of languages accepted by a game between two players, one who makes existential moves and the other makes universal moves. Peterson and Reif [25] show that $NEXP$ can be described by a game with three players, two existential players unable to communicate and one universal player who communicates with the other two. Simon [28] and Orponen [23] describe a game between an existential oracle and a universal player and show the equivalence to $NEXP$. Remarkably, in all of these cases, the universal player can be replaced by a probabilistic polynomial time player without reducing the strength of the models. For $PSPACE$, this follows from [27]; for $NEXP$, the equivalence is established by our main result.

In the course of the proof of the main theorem, we show how to *test whether a function in n variables over* $\mathbf{Z}$, *given as an oracle, is multilinear over a large interval.* This test has independent interest for *program testing and correction*, in the context of Blum-Kannan [8], Blum-Luby-Rubinfeld [9], and Lipton [20] (see Section 6).

The reduction to the test involves ideas of the $PSPACE = IP$ proof (polynomial extrapolation of truth values). The proof of correctness of the multilinearity test rests on combinatorial techniques including simple eigenvalue calculation to estimate the expansion rate of a graph.

2 Multi-prover protocols and Probabilistic Oracle Machines

In this section we give some basic background on multiprover interactive proof systems. The definitions and results first appeared in Ben-Or–Goldwasser–Kilian–Wigderson [7] and Fortnow–Rompel–Sipser [15]. For completeness, we include an outline of the proofs.

Let $P_1, P_2, \ldots P_k$ be infinitely powerful machines and V be a probabilistic polynomial-time machine, all of which share the same read-only input tape. The verifier V shares communication tapes with each P_i, but different provers P_i and P_j have no tapes they can both access besides the input tape. We allow k to be as large as a polynomial in the size of the input; any larger and V could not access all the provers.

Formally, similarly to the prover of a single prover interactive proof system [17], each P_i is a function that outputs a message determined by the input and the conversation it has seen so far. We put no restrictions on the complexity of this function other than that the lengths of the messages produced by this function must be bounded by a polynomial in the size of the input.

$P_1, \ldots, P_k$ and V form a multi-prover interactive protocol for a language L if:

1. If $x \in L$ then $\Pr(P_1, \ldots, P_k \text{ and } V \text{ on } x \text{ accept}) > 1 - 2^{-n}$.

2. If $x \notin L$ then for all provers $P'_1, \ldots, P'_k$, $\Pr(P'_1, \ldots, P'_k \text{ and } V \text{ on } x \text{ accept}) < 2^{-n}$

MIP is the class of all languages which have multiprover interactive protocols. If $k = 1$ we obtain the class IP of languages accepted by standard interactive proof systems.

Let M be a probabilistic polynomial time Turing machine with access to an oracle O. We define the languages L that can be described by these machines as follows:

We say that L is accepted by a *probabilistic oracle machine* M iff

1. For every $x \in L$ there is an oracle O such that M^O accepts x with probability $> 1 - \frac{1}{p(|x|)}$ for all polynomials p and x large.

2. For every $x \notin L$ and for all oracles O, M^O accepts with probability $< \frac{1}{p(|x|)}$ for all polynomials p and x large.

One way to think about this model is that the oracle convinces M to accept. This differs from the standard interactive protocol model in that the oracle must be set ahead of time while in an interactive protocol the prover may let his future answers depend on previous questions. Fortnow-Rompel-Sipser [15] show that L is accepted by a *probabilistic oracle machine* if and only if $L$is accepted by a *multi-prover interactive protocol.* An immediate consequence is the result of Ben-Or, Goldwasser, Kilian and Wigderson [7] that if a language L is accepted by a *multi-prover* interactive protocol then L is accepted by a *two-prover* interactive protocol.

3 Proof of the Main Theorem

This section as well as the next one are devoted to proving Theorem 1.1. In view of the fact $MIP \subseteq NEXP$ ([15], see the comment before the statement of Theorem 1.1) we have to prove $NEXP \subseteq MIP$.

3.1 Preliminary remarks

Look at the tableau describing the computation of a non-deterministic exponential time Turing machine M on input x. Convert this to a 3-CNF like in the proof of the Cook-Levin theorem (NP-completeness of 3-satisfiability; Aho, Hopcroft and Ullman, p. 385). There will be an exponential number of variables and an exponential number of clauses. However, the clauses are easily definable, if fact there exists a polynomial-time function $f_x(i)$ that describes the variables of clause i. Thus $L(M) = \{x|$ there is an assignment of variables A that for all i, A satisfies clause $f_x(i)\}$.

Suppose we could quantify over all functions. Then we could say M accepts x iff there exists a function A taking variables to "true" or "false" such that for all i, A satisfies $f_x(i)$. Note that it is important that A is completely specified before i is chosen. Also notice that given A as an oracle, we can check whether A satisfies $f_x(i)$ in polynomial time.

In fact, as outlined in Section 2, we can create predetermined, though untrustworthy, functions (oracles) using multiple-prover protocols. So we can use multi-provers to create A.

The next thing to do is to ask if A satisfies $f_x(i)$ for all i. However we cannot immediately do such universal quantification with multi-provers. The obvious "statistical approach", replacing the "for all i" with "for most i" will clearly fail.

We might try handling the universal quantification with the techniques of Lund-Fortnow-Karloff-Nisan [21], Babai-Fortnow [5], and Shamir [27], but these results do not relativize and A may not have the proper algebraic properties necessary for this proof.

We need further reduction of the problem, involving a deeper arithmetization of the fact that $f_x(i)$ is polynomial time computable.

3.2 Arithmetization of $NEXP$

We begin with some simple observations.

Definition 3.1 *A function $g : \mathbf{Z}^n \to \mathbf{Q}$ represents a boolean function $f : \{0,1\}^n \to \{0,1\}$ if for every $\alpha \in \{0,1\}^n$,*

$$g(\alpha) = f(\alpha).$$

Proposition 3.1 *Given a function $f : \{0,1\}^n \to \{0,1\}$ there exists a unique multilinear function $g : \mathbf{Z}^n \to \mathbf{Z}$ representing f.* □

Proposition 3.2 *If $\varphi(x_1, \ldots, x_n)$ is a 3-CNF formula then an arithmetic expression of a polynomial $f(x_1, \ldots, x_n)$ of degree $3m$ representing φ can be computed from φ in polynomial time, where m is the number of clauses of φ.* □

Lemma 3.3 essentially states that any nondeterministic polynomial-time language can be expressed as a low degree polynomial.

Lemma 3.3 *If $B \subseteq \{0,1\}^*$ is a nondeterministic polynomial time computable language then there exists a constant c such that for every n there exists a polynomial P_n of degree $\leq n^c$ in $n + n^c$ variables such that for every $\alpha \in \{0,1\}^n$*

$$\alpha \in B \Longleftrightarrow (\exists \beta \in \{0,1\}^{n^c})(P_n(\alpha, \beta) \neq 0).$$

On input 1^n, an arithmetic expression of P_n is computable in polynomial time.

proof : Let φ_{B_n} be the 3-CNF from the Cook-Levin Theorem. We thus have

$$\alpha \in B \Longleftrightarrow (\exists \beta \in \{0,1\}^{n^c})(\varphi_{B_n}(\alpha, \beta) \neq 0).$$

Now by the preceding proposition there exists P_n that represents φ_{B_n}. □

The following result is essentially due to J. Simon [28]; similar proofs appear in Peterson–Reif [25] and Orponen [23].

Lemma 3.4 (NEXP version of Cook-Levin) *Let $L \in NEXP$. Then for every $x \in \{0,1\}^n$ there exists a 3-CNF formula with an exponential number of clauses and variables and with the properties listed below:*

$$\Phi_x(X(0),\ldots,X(2^{n^c}-1)) = \bigwedge_{i=0}^{2^{n^c}-1} C_i,$$

where

$$C_i = t_0^i X(b_1^i) \vee t_1^i X(b_2^i) \vee t_2^i X(b_3^i).$$

Here $t_j^i \in \{0,1\}$; where "0X" means "$\neg X$" and "1X" means "X". The following properties hold.

(1) $b_j^i \in \{0,1\}^{n^c}$.

(2) The values of t_j^i and b_j^i are computable in polynomial time from i, x, j.

(3) For every $x \in \{0,1\}^n$,

$$x \in L \Longleftrightarrow \exists X \in \{0,1\}^{n^c} : \Phi_x(X) \quad \square$$

The next lemma describes a low degree polynomial with free access to an assignment of A from which we can determine if A satisfies the 3-CNF.

In order to verify A we have to extend its domain from $\{0,1\}^{n^c}$ to $\mathbf{I}^{n^c}$ where $\mathbf{I} = \{0,\ldots,N-1\}$ for some suitable large integer N. So we say that $A : \mathbf{I}^{n^c} \to \mathbf{Q}$ satisfies Φ_x if $A|_{\{0,1\}^{n^c}}$ satisfies Φ_x. In particular A must take $\{0,1\}$ values on $\{0,1\}^{n^c}$.

Lemma 3.7 *Let $\Phi_x(X(0),\ldots,X(2^{n^c}-1))$ be the 3-CNF formula associated above with input x ($n = |x|$) for a language $L \in NEXP$. Then there exists a polynomial Q_x in $n^{O(1)}$ variables such that*

(1) An arithmetic expression for Q_x is computable in $n^{O(1)}$ time from x. In particular the degree of Q_x is $n^{O(1)}$.

(2) A function $A : \mathbf{I}^{n^c} \to \mathbf{Q}$ satisfies Φ_x if and only if

$$\sum Q_x(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0 \quad (1)$$

where the summation extends over all (0,1)-substitutions of the variables. ($z \in \{0,1\}^{n^{O(1)}}, b_j \in \{0,1\}^{n^c}$). $\square$

3.3 The protocol

We let the provers provide A. We shall clarify in Section 5 how this is done.

If A is multilinear then we shall show how to verify that the sum (1) is zero. We shall discuss the verification of (approximate) multilinearity after this protocol.

Since $A(b)$ is now a multilinear function of its n^c arguments $b \in \mathbf{I}^{n^c}$, the function $R_x^A(z, b_1, b_2, b_3) = Q_x(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3))$ is a polynomial of polynomial degree of the $n^{O(1)}$ components of (z, b_1, b_2, b_3). Now we can run the protocol of Lund-Fortnow-Karloff-Nisan, Babai-Fortnow and Shamir with a single prover to verify this value is zero.

Roughly the protocol works as follows: Let $m = n^C$ be an upper bound on the time required to calculate the arithmetic expression for Q_x. Let $\mathbf{I} = \{0,\ldots,N-1\}$ where $N > m^3$.

V: Let value=0. Repeat the following process for each variable v (for all the i's, t's, b's and z's)

V: Take v out of the sum and look at the expression as a function of v.

$P \to V$: A polynomial $q(v)$ over $\mathbf{Z}$ of degree $\leq mn^c$, with integer coefficients $\leq (2N)^{mn^c}$.

V: Check $q(0) + q(1)$ =value. Pick a random b in $\mathbf{I}$ and let value= $q(b)$. Replace v with b in the formula.

V: Repeat the above for each variable and then check that the remaining formula computes to value.

This protocol works essentially because if two polynomials of low degree differ then they differ in many locations; clearly the degree of Q_x is less than m and therefore, if A is indeed multilinear in its n^c variables, the degree of R_x^A is less than $3mn^c$. The upper bound on the coefficients easily follows if A indeed represents a Boolean function.

So the only remaining problem is if the provers can try to gain an advantage by having the function A not multilinear. Indeed, A could be some horribly complex functions intended to foul up the above protocol that uses heavily the fact that A is a low-degree polynomial.

We can guard against this problem with the following result. For typographic convenience, we use $\exp_2(u)$ to denote 2^u.

Theorem 3.8 *Let $d \geq 1$ and $k \geq 1$ be fixed constants. Let N be an integer, $n^{d+3} < N \leq 2^{n^k}$ for some n. Let $\mathbf{I}$ denote the set of integers $\{0,\ldots,N-1\}$. Let $A(x_1,\ldots,x_n)$ be an arbitrary function from $\mathbf{I}^n$ to $\mathbf{Q}$. We say A is α-approximately multilinear if there exists a multilinear function g such that $g = A$ on at least a $1-\alpha$ fraction of $\mathbf{I}^n$. Then for any constant k' there exists a probabilistic polynomial-time Turing machine M such that given access to A as an oracle:*

1. *If A is multilinear, integral valued, and does not take values greater than $\exp_2(n^{k'})$ then M^A always accepts.*

2. *If A is not n^{-d}-approximately multilinear then with high probability M^A rejects.*

We prove this result in the next section. We remark that the machine to be constructed will carry the following additional guarantee, which we don't need for the proof of the main theorem.

(3) If A is n^{-d}-approximated by a multilinear function g and for some $\alpha \in \mathbf{I}^n$, $|g(\alpha)| > \exp(n^{k+k'+2})$, or $n!g$ is not integral valued, then with high probability M^A rejects.

Remark 3.9 Theorem 3.8 has applications to program testing and correcting. We shall elaborate on this in Section 6.3.

Remark 3.10 The same test works if we replace multilinearity by the condition that the polynomial be of low degree. Let $k_1, \ldots, k_n$ be positive integers $< n^c$. Assume we wish to test if the function $A : \mathbf{I}^n \to \mathbf{Z}$ is a polynomial having degree $\leq k_i$ in variable x_i for every i. Theorem 3.8 extends to this situation, with only trivial modifications in the proof.

Before the verifier runs the protocol on formula (1), it checks A using the algorithm in Theorem 3.8 for $d = 2, k' = k + 1$. For honest provers, A is multilinear and takes small values only ($< \exp(n^{k+1})$), so this test always accepts. If the test accepts, the verifier can have high confidence that A is n^{-2}-approximately multilinear.

In the end of the protocol, all of the b's are replaced by random elements from $\mathbf{I}$. With probability greater than $1 - \frac{1}{n}$, $g = A$ on the three queries that the verifier asks of A. Thus if the provers used g instead of A then the probability that the verifier would accept could change by no more than $\frac{1}{n}$. However, we argued above that if the provers used a multilinear function, they would not convince the verifier to accept a bad input with more than some small probability.

To summarize the conclusion:

1. If A represents an honest extension of a satisfying instance of Φ_x then the verifier will accept.

2. Conversely, if the verifier accepts, then he can infer with high confidence that g represents a satisfying instance.

This completes the proof of Theorem 1.1 modulo the multilinearity assumption to be clarified in the next section as stated in Theorem 3.8. An easy two-prover implementation of the protocol which refers to A as an oracle (predetermined function) will be discussed in Section 5.

3.4 The Power of the Provers

We state a byproduct of the above proof regarding the required power of the provers. Let $\mathcal{C}$ be either class of languages or a class of functions. We say that a language L has (single or multiple prover) interactive proof systems with provers of complexity $\mathcal{C}$ if

- For any $x \in L$, honest provers P_i restricted to answering questions of membership in some language $L_i \in \mathcal{C}$ are able to convince the verifier about membership of x in L;

- Even all-powerful provers don't have a chance of convincing the verifier about membership of x in L if in fact $x \notin L$.

If $\mathcal{C}$ is a class of functions, we define provers of complexity $\mathcal{C}$ analogously: the honest provers are restricted to evaluating some function $f \in \mathcal{C}$. It is clear that a prover of power $\mathcal{C}$ is equivalent to one of power $P^{\mathcal{C}}$. In particular, provers of power PP are equivalent to provers of power $\#P$ since $P^{PP} = P^{\#P}$.

The result of Feldman [12] combined with Shamir's [27] implies that $PSPACE$ has single prover interactive proof systems with a prover of complexity $PSPACE$. The result of Lund et al. [21] implies that $P^{\#P}$ has single prover interactive proof systems with a prover of complexity $\#P$. A similar property of EXP follows from our proof.

Corollary 3.11 *For any $L \in EXP$, there is a multiple-prover interactive proof system with provers of complexity EXP.* □

4 Verification of Multilinear Functions.

First we need some definitions and notation.

As before, we use $\mathbf{I}$ to denote the interval $\{0, 1, \ldots, N-1\}$ for some suitable large integer N.

We shall consider the n'th cartesian power of a finite set X (usually $X = \mathbf{I}$). A subset $U \subseteq X^n$ will be called a k-dimensional *subspace* of X^n if there exist $n - k$ different coordinates $i_1, \ldots, i_{n-k}$ and $n - k$ values of these coordinates $\alpha_{i_1}, \ldots, \alpha_{i_{n-k}} \in X$ such that

$$U = \{(\alpha_1, \ldots, \alpha_n) : x_{i_j} = \alpha_{i_j} \text{ for } j = 1, 2, ..., n-k\}$$

A *line* is a 1-dimensional subspace. The points of a line in the k^{th} direction have all but the k^{th} coordinate in

common. We shall denote by L the set of lines and by L_i the set of lines in the i'th direction. A *hyperplane* is a subspace of dimension $n-1$. We shall use these terms for the case $X = \mathbf{I}$. Note that what we call *subspaces* correspond to the subspaces *aligned* with the coordinate system in the affine space. (E.g., in this terminology, $\mathbf{I}^n$ has nN hyperplanes.)

Definition 4.1 *Let f, g be functions over a finite set X. For $\delta \in [0,1]$ we say that f* δ-approximates *g if $\mu(\{x \in X | f(x) \neq g(x)\}) < \delta$.*

Definition 4.2 *Let $f : \mathbf{I}^n \to \mathbf{Q}$ be a function. We call f* multilinear *if its restriction to any line (in the above sense) of $\mathbf{I}^n$ is linear.*

Definition 4.3 *Let $f : \mathbf{I}^n \to \mathbf{Q}$. For $\delta \in [0,1]$ we say that f is* δ-approximately multilinear *if there exists a multilinear g such that g δ-approximates f. If n=1, we obtain the concept of* δ-approximately linear *functions.*

Definition 4.4 *Given a function $A : \mathbf{I}^n \to \mathbf{Q}$, we call a line ℓ in $\mathbf{I}^n$* correct, *if the restriction $A|_\ell$ is a linear function. We say that ℓ is* δ-wrong *or just* wrong *if $A|_\ell$ is not δ-approximately linear.*

4.1 The Test

If we ever catch a point where the value of A is not integral or too large ($> \exp_2 n^k$), we reject and halt. Henceforth we assume this never occurs. From this one can infer with high confidence that

(*) for most $x \in \mathbf{I}^n$, $A(x)$ is integral and not greater than $K = \exp_2(n^{k+1})$.

Although we don't need this later on, we mention that it follows from (*) that, if $A(x)$ is multilinear, then it never gets too large (greater than $n^n K$) on $\mathbf{I}^n$. Furthermore, $n!A(x)$ is integral. – These conclusions hold even if we replace "most" by "a positive fraction of" in (*).

First we describe a subtest that tests if a line is wrong:

Test$_0$(line ℓ) Select $m_1 + 2$ random points of ℓ. If A restricted to these points agrees with a linear function then *accept* else *reject.*

Proposition 4.1 *(a) If ℓ is correct, then Test$_0$ surely accepts ℓ.*

(b) If ℓ is wrong, Test$_0$ will reject it with probability $> 1 - \exp(-\delta N)$.

proof : Take two of the points; interpolate their A-values to a linear function $h(x)$ on ℓ. If for more than a δ fraction of $x \in l$ we have that $A(x) \neq h(x)$ then the probability that the test detects no such point is at most

$$(1-\delta)^{m_1} < \exp(-\delta m_1) \quad \square$$

Now the whole test is the following:

Test Select m_2 random lines from L_i for each i. If Test$_0$ accepts each of these lines then *accept* else *reject.*

Proposition 4.2 *(a) If A is multilinear, then Test accepts.*

(b) If for some i more than an ϵ fraction of the lines in L_i is wrong, then the probability that **Test** *rejects is greater than*

$$1 - (\exp(-\epsilon m_2) + \exp(-\delta m_1)).$$

proof : The probability that no wrong line is selected is $(1-\epsilon)^{m_2} < \exp(-\epsilon m_2)$. The probability that a wrong line, when selected, remains undetected, is $< \exp(-\delta m_1)$. It is easy to see that the sum of these two quantities is an upper bound on the failure probability of the Test. $\square$

We paraphrase statement (b) above.

Proposition 4.3 *Given a function $A : \mathbf{I}^n \to \mathbf{Q}$, assume that A passes the Test with $m_2 = t/\epsilon, m_1 = t/\delta$. Then we infer with confidence $\geq 1 - 2e^{-t}$ that*

$(\forall i)$ the proportion of wrong lines among L_i $< \epsilon$. (**)

The rest of this section is devoted to proving that the above conclusion (**) implies that A is ϵ'-approximately multilinear for some small ϵ'. See Theorem 4.9 (end of this section) for the formal statement of this result.

4.2 The Self-Improvement Lemma

We need a combinatorial isoperimetric inequality.

Definition 4.5 *Let X be a finite set. For $S \subseteq X^n$ define the closure of S as*

$$\bar{S} = \cup_{i=1}^n \pi_i^{-1}(\pi_i(S))$$

where $\pi_i : X^n \to X^{n-1}$ is the projection in the i'th dimension.

Lemma 4.4 (Expansion Lemma) *Let $S \subseteq X^n$. If $|S| \leq |X|^n/2$ then $|\bar{S}| \geq |S|(1 + 1/2n)$.*

This lemma was proved by D. Aldous [2, Lemma 3.1]. It is also implicit in work by Babai and Erdős [4, Lemma].

The key step in the induction argument that will yield Theorem 4.9 is the verification that if a function passes the Test and it is multilinear on a fair portion of the space then it is actually multilinear almost everywhere. Here is the formal statement:

Lemma 4.6 (Self-improvement lemma) *Given a function* $A : \mathbf{I}^n \to \mathbf{Q}$, *assume that*

$(\forall i)$ *the proportion of wrong lines among* L_i *is* $< \epsilon$

and

$\exists g : g$ *is multilinear and* g Δ*-approximates* A,

where $\Delta \leq 1/2$. *Then*

g ϵ'*-approximates* A, *where* $\epsilon' = 3n^2(\epsilon + \delta + 1/N)$.

proof : based on the Expansion Lemma. □

4.3 The Pasting Lemma

The multilinear function which closely approximates A will be constructed for certain subspaces by induction on their dimension. What we show below is that if A is approximately linear on most lines and approximately multilinear on a fair portion of the hyperplanes, then it is approximately multilinear on the entire space. The "self-improvement lemma" prevents the devaluation, through repeated application in the induction argument, of the term "approximately" in this result.

Lemma 4.7 (Pasting Lemma) *Given a function* $A : \mathbf{I}^n \to \mathbf{Q}$, *assume that* $\delta, \epsilon > 0$, $\epsilon + \delta \leq \frac{1}{200n}$, $N \geq 40n$,

$(\forall i)$ *the proportion of wrong lines among* L_i *is* $< \epsilon$

and $\exists g : g(x,y)$ *is multilinear in* $y \in \mathbf{I}^{n-1}$ *such that the set*

$$\Phi := \{\xi | g(\xi, y) \ \beta\text{-approximates } A(\xi, y)\},$$

has fair density:

$$\mu(\Phi) \geq \varphi$$

where $\varphi = \frac{1}{10n}$ *and* $\beta = \frac{1}{10}$.

Then A *is* Δ*-approximately multilinear, where* $\Delta = \epsilon + \delta + 4\beta \leq 1/2$. *And by the self-improvement lemma* A *is* ϵ'*-approximately multilinear, where* $\epsilon' = 3n^2(\epsilon + \delta + 1/N)$.

proof : First observe that the first part of the assumption implies that there exist functions $f_1(y)$ and $f_2(y) : \mathbf{I}^{n-1} \to \mathbf{Q}$ such that $xf_1(y) + f_2(y)$ $(\epsilon + \delta)$-approximates $A(x,y)$. Define $\Psi = \{\xi | \xi f_1(y) + f_2(y)$ β-approximates $A(\xi, y)$ as a function of $y\}$. Then $\mu(\Psi) \leq \frac{\epsilon+\delta}{\beta} = 10(\epsilon+\delta)$. So $|\Phi \setminus \Psi| \geq N\left(\frac{1}{10n} - 10(\epsilon+\delta)\right) \geq 2$. Let $\xi_1, \xi_2 \in \Phi \setminus \Psi$, $\xi_1 \neq \xi_2$. Then for $i = 1, 2$ there exist a multilinear function $g_i(y)$ that 2β-approximate $\xi_i f_1(y) + f_2(y)$. Hence on a set of measure $1 - 4\beta$ we have that

$$f_1(y) = \frac{g_1(y) - g_2(y)}{\xi_1 - \xi_2}$$

and

$$f_2(y) = \frac{\xi_2 g_1(y) - \xi_1 g_2(y)}{\xi_2 - \xi_1}$$

Now denote the multilinear functions on the right hand side by $\tilde{f}_1(y), \tilde{f}_2(y)$. Then the multilinear function $x\tilde{f}_1(y) + \tilde{f}_2(y)$ Δ-approximates $A(x,y)$. □

4.4 The Tree Coloring Lemma.

The next lemma provides the overall structure of the induction. It demonstrates, as we shall see in the next subsection, that the Pasting lemma is strong enough to carry approximate multilinearity all way from most lines to the entire space.

Let T be a depth n levelwise uniform tree (vertices on the same level have the same number of children). We will color the tree by two colors red/white. The input is a coloring of the leaves. Then we color the tree bottom up according to the following rule.

Fix the parameters ϵ_0 and φ. $0 \leq \epsilon_0, \varphi \leq 1$. Color a vertex red if each of the following two conditions are met:

- "Almost all leaves" in the subtree rooted at T_v are red: only a $< \epsilon_0$ fraction is white.
- A "fair number" of children of v are red: the proportion of red children is $\geq \varphi$.

Lemma 4.8 (Tree coloring lemma) *Let* $\epsilon_k = (1-\varphi)^k \epsilon$. *Let* v *be a vertex on level* k. *(The leaves are on level 0.) Assume that all but an* ϵ_k *fraction of the leaves in* T_v *are red. Then* v *is red.*

proof : By induction on k.

$k = 1$ By assumption the proportion of red children of v is $1 - \epsilon_1 = 1 - (1-\varphi)\epsilon_0 \geq \varphi$ and the proportion of white leaves of T_v is less than $(1-\varphi)\epsilon_0 \leq \epsilon_0$ so v is red.

$k \geq 2$ Take a random child u of v. Now $E_u(\mu($ white leaves in $T_u)) < \epsilon_k)$. Hence $Pr_u(\mu($ white leaves in $T_u) < \epsilon_{k-1}) < \frac{\epsilon_k}{\epsilon_{k-1}} = 1 - \varphi$. But this probability is by the inductive hypothesis greater than $Pr_u[u$ is white$]$. So

$$Pr_u[u \text{ is red }] \geq \varphi$$

Hence the proportion of red children of v is greater than φ and also the proportion of red leaves in T_v is greater than $1 - \epsilon_k \geq 1 - \epsilon_0$. Hence v is red. □

4.5 Conclusion.

Theorem 4.9 *Given $A : \mathbf{F}^n \to \mathbf{Q}$, assume that*

($\forall i$) the proportion of wrong lines among L_i is $< \epsilon$.

Then

A is ϵ'-approximately multilinear,

where $\epsilon' = 3n^2(\epsilon + \delta + 1/N)$, assuming the parameters have been so chosen that $N \geq 40n^2$, $\delta \leq \frac{1}{400n^2}$ and $\epsilon \leq \frac{1}{800n^3}$.

proof : We first construct a tree T of depth $n-1$. The nodes of the tree will correspond to subspaces of $\mathbf{F}^n$. (We consider aligned affine subspaces; see the conventions stated at the beginning of Section 3.) The root corresponds to $\mathbf{F}^n$; and the children of a node correspond to its hyperplanes. Observe that the leaves correspond to lines. We color all the leaves corresponding to a wrong line white, all the others red. Now we color the rest of T accordingly to the coloring rule with $\epsilon_0 = 2\epsilon$ and $\varphi = \frac{1}{10n}$. Note that $\epsilon < \epsilon_n = (1-\varphi)^n \epsilon_0$. From the coloring lemma we obtain that the root is red. Now we only have to make the following observation. We shall say that a subspace U is β-approximately multilinear if the restriction $A|_U$ has this property.

Lemma 4.10 *If $v \in T$ is red then the subspace U_v corresponding to v is β-approximately multilinear, where $\beta = 1/10$.*

proof : By induction on the level k of v.

$k = 1$ is okay since $\delta < \beta$.

$k \geq 1$ We know that since v is red, it has a fraction of $\geq \varphi$ red children. By the inductive hypothesis the subspaces corresponding to them are β-approximately multilinear. There must thus be a direction such that a fraction of $\geq \varphi$ of hyperplanes of U_v in that direction is β-approximately multilinear.

Since v is red we also know that the proportion of wrong lines in U_v is $< \epsilon_0$. This implies that the proposition of wrong lines in U_v in any direction is $< n\epsilon_0$. Therefore, by the Pasting lemma U_v is ϵ^*-approximately multilinear, where $\epsilon^* = 3n^2(n\epsilon_0 + \delta + 1/N)$. This concludes the proof of the lemma since the choice of parameters implies that $\epsilon^* \leq \beta$. □

So now A is ϵ^*-approximately multilinear. By the Self-improvement lemma it follows that A is ϵ'-approximately multilinear, completing the proof of Theorem 4.9. □

5 Implementing the Protocol with Two Provers

We now show how to use two provers to execute the above protocol based on work of Fortnow–Rompel–Sipser and Ben-Or–Goldwasser–Kilian–Wigderson described in Section 2. We will ask prover 2 exactly one question about A. We in fact define A as the function of prover 2's responses to the single question he is asked.

We execute the entire protocol with prover 1 including questions about A. We then choose a random question we have asked prover 1 about A and ask this question to prover 2. If the answers differ then we reject.

If the provers have been honest then all questions about A will have the same answer with prover 1 and prover 2. If prover 1 gives an incorrect answer about A then the verifier will catch him with probability at least $1/n^k$ where n^k is greater then the number of questions about A that the verifier has asked of prover 1.

We repeat the above protocol n^{k+1} times. Then if prover 1 has to lie about some value of A on each run of the protocol then the verifier will catch the prover with probability $> 1 - e^{-n}$.

6 Program Testing, Verification and Self-Reducibility

The results of this paper have many connections to program testing, verification and self-correcting code. We make the connections precise in this section.

6.1 Robustness

In this section we will describe a useful property, $PSPACE$-robustness, of languages. We show that every $PSPACE$-robust language is Turing-equivalent to a family of multilinear functions (one n-variable function for every n).

Definition 6.1 *A language L is $PSPACE$-robust if $P^L = PSPACE^L$.*

Examples of $PSPACE$-robust languages include the $PSPACE$-complete and EXP-complete languages.

Lemma 6.1 *Every $PSPACE$-robust language has a Turing-equivalent family of multilinear functions over the integers.*

proof : Let L be a $PSPACE$-robust language. Let $g_n(x_1, \ldots, x_n)$ be the multilinear extension of the characteristic function of $L_n = L \cap \{0,1\}^n$ (see Proposition 3.1). Clearly $L \in P^g$, where $g = \{g_n : n \geq 0\}$.

We will describe an alternating polynomial-time Turing machine with access to L computing g. First guess the value $z = g_n(x_1, \ldots, x_n)$. Then existentially guess the linear function $h_1(y) = g(y, x_2, \ldots, x_n)$ and verify that $h_1(x_1) = z$. Then universally choose $t_1 \in \{0,1\}$ and existentially guess the linear function $h_2(y) = g(t_1, y, x_3, \ldots, x_n)$. Keep repeating this process until we have specified $t_1, \ldots, t_n$ and then verify that $t_1 \ldots t_n \in L$. Since a $PSPACE$ machine can simulate an alternating polynomial-time Turing machine, if L is $PSPACE$-robust then g is Turing-reducible to L. □

In particular, we have multilinear $PSPACE$-complete functions, EXP-complete functions, etc. This lemma, inspired by Beaver-Feigenbaum [6] and spelled out simultaneously by the authors of this paper and of [6], has significant consequences, as we shall see below.

There are natural classes of languages satisfying the conclusion of Lemma 6.1 which are not known to be $PSPACE$-robust; $P^{\#P}$-complete languages being the prime example, since they are equivalent to the permanent, a multilinear function (Valiant [30]).

6.2 Instance Checking

In Blum-Kannan [8], "function-restricted IP" is defined as follows:

The set of all decision problems π for which there is an interactive proof system for YES-instances of π satisfying the conditions that the honest prover must compute function π and any dishonest prover must be a function from the set of instances to {YES, NO}.

By a theorem due to Fortnow-Rompel-Sipser [15] we see that function-restricted IP is equivalent to multi-prover interactive proof systems where the honest provers can only answer questions about the language they are being asked to prove.

Blum-Kannan also define a program checker $C_L^{\mathcal{P}}$ for a language L and an instance $x \in \{0,1\}^*$ as a probabilistic polynomial-time oracle Turing Machine that given a program $\mathcal{P}$ claiming to compute L, and an input x:

1. If $\mathcal{P}$ correctly computes L for all inputs then with high probability $C_L^{\mathcal{P}}$ will output "correct".
2. If $\mathcal{P}(x) \neq L(x)$, with high probability $C_L^{\mathcal{P}}(x)$ will output "$\mathcal{P}$ does not compute L".

Blum-Kannan show that a language has a program checker if and only if the language and its complement each have a function-restricted interactive proof system.

The recent results by Lund-Fortnow-Karloff-Nisan [21] and Shamir [27] show all $P^{\#P}$-complete and $PSPACE$-complete languages have function restricted interactive proofs and (since both classes are closed under complements) program checkers. This implies a program checker for any $\#P$-complete function such as the permanent of a matrix.

The following follows from Corollary 3.11:

Corollary 6.2 *Every EXP-complete language has a function-restricted interactive proof system and thus a program checker.*

Thus not every language in function-restricted IP has a single prover interactive proof unless $PSPACE$=EXP. This essentially gives a negative answer to the open question of Blum-Kannan [8] as to whether IP contains function-restricted IP.

Still open is the question as to whether NP-complete languages have program checkers. This is directly related to the question of whether $co - NP$ languages have protocols with NP provers (see Section 3.4)

6.3 Self-Testing and Self-Correcting Programs

Our test of multilinear functions (Section 4) also has applications to program testing as described by Blum-Luby-Rubinfeld [9] and Lipton [20]. We will use the following definition from Blum-Luby-Rubinfeld:

A pair of probabilistic programs (T, S) is a *self-testing/correcting pair* for f if given a program $\mathcal{P}$ as an oracle

1. If $\mathcal{P}$ computes f correctly on every input then $T^{\mathcal{P}}$ always accepts.
2. If $\mathcal{P}$ differs from f on at least $1/n^2$ of the inputs then $T^{\mathcal{P}}$ will reject with probability at least $1 - \frac{1}{2^n}$.
3. If $\mathcal{P}$ differs from f on at most $1/n^2$ of the inputs then $S^{\mathcal{P}}$ will correctly compute f *on every input* with probability $1 - \frac{1}{2^n}$.

This definition is more restrictive than the definition given by Blum-Luby-Rubinfeld [9] which is more general in many ways including weaker error bounds.

We can use verification of multilinear functions to convert a weak self-testing program to a strong self-testing/correcting pair for a multilinear function.

Corollary 6.3 *Let f be a multilinear function. Suppose we had a polynomial-time program T' that self-tests f where T' works correctly only under the assumption that $\mathcal{P}$ is multilinear. Then there exists a self-testing/correcting pair of polynomial-time computable functions for f.*

proof : For the self-testing program T we will use the multilinearity test described in Section 4 followed by T'. □

We can also do program verification in the spirit of Blum-Luby-Rubinfeld without an assumption of a tester T' as follows: Suppose a program $\mathcal{P}$ claims to compute a multilinear function. We can test that there is some multilinear function f such that $\mathcal{P} = f$ on most inputs and then if $\mathcal{P} = f$ on most inputs we can create a correcting function S such that $S = f$ on all inputs with high probability. The proof is virtually identical to the proof above.

We can use Corollary 6.3 to exhibit some functions that have self-testing/correcting pairs.

Theorem 6.4 *If L is $PSPACE$-robust and has a function-restricted interactive proof system then there exists a family g of multilinear functions Turing-equivalent to L that has a self-testing/correcting pair.* □

Corollary 6.5 *There exist $PSPACE$-complete and EXP-complete families of functions with self-testing/correcting pairs.* □

References

[1] Aho, Hopcroft, Ullman: *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] Aldous, D.: On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing, *Probability in the Engineering and Informational Sciences* 1 (1987), pp. 33-46.

[3] Babai, L.: Trading group theory for randomness, *Proc. 17th STOC* (1985), pp. 421-429.

[4] Babai, L., Erdős, P.: Representation of group elements as short products, *Annals of Discrete Mathematics* 12 (1982), pp. 27-30.

[5] Babai, L., Fortnow, L.: A characterization of $\#P$ by straight line programs of polynomials. *Proc. 31th FOCS* (1990).

[6] Beaver, D., Feigenbaum J.: Hiding Instances in Multioracle Queries, *Proc. 7th Symp. on Theoretical Aspects of Comp. Sci.* , *LNCS* 415 (1990), pp. 37-48.

[7] Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A.: Multi-prover interactive proofs: How to remove the intractability assumptions, *Proc. 20th STOC* (1988), pp. 113-131.

[8] Blum, M., Kannan, S.: Designing Programs that Check Their Work, *Proc. 21st STOC* (1989), pp. 86-97.

[9] Blum, M., Luby, M., Rubinfeld, R.: Self-testing and Self-correcting programs, with Applications to Numerical Programs, *Proc. 22nd STOC* (1990), pp. 73-83.

[10] Cai, J.: PSPACE is provable by two provers in one round, manuscript (1990).

[11] Cook, S. A.: The complexity of theorem proving procedures, *Proc. 3rd STOC* (1971), pp. 151-158.

[12] Feldman, P.: The Optimum Prover lives in $PSPACE$, manuscript, M.I.T.(1986).

[13] Fortnow L.: The Complexity of Perfect Zero-Knowledge, In S. Micali, ed., *Randomness and Computation*, Advances in Computing Research Vol. 5 (1989), JAI Press, pp. 327-343.

[14] Fortnow L.: Complexity-Theoretic Aspects of Interactive Proof Systems, Ph.D. Thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, Tech Report MIT/LCS/TR-447 (1989).

[15] Fortnow, L., Rompel, J., Sipser, M.: On the power of multi-prover interactive protocols, *Proc. 3rd Structure in Complexity Theory Conf.* (1988), pp. 156-161.

[16] Fortnow, L., Sipser, M.: Are there interactive protocols for co-NP languages?, *Inf. Proc. Letters* 28 (1988), pp. 249-251.

[17] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems, *Proc. 17th STOC* (1985), pp. 291-304.

[18] Heller, H.: On Relativized Exponential and Probabilistic Complexity Classes, *Information and Computation* 71 (1986), 231-243.

[19] Levin, L.: Universal'nyĭe perebornyĭe zadachi (Universal search problems : in Russian), *Problemy Peredachi Informatsii* 9: 3, pp. 265-266.

[20] Lipton, R. J.: New directions in testing, manuscript (1989).

[21] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic Methods for Interactive Proof Systems, *Proc. 31th FOCS* (1990).

[22] Nisan, N.: co-SAT has multi-prover interactive proofs, *e-mail announcement*, Nov. 27 1989.

[23] Orponen, P.: Complexity Classes of Alternating Machines with Oracles, *Proc. 10th ICALP* (1983), *LNCS* 154, pp. 573-584.

[24] Papadimitriou, C.: Games against Nature, *Proc. 24th FOCS* (1983), pp. 446-450.

[25] Peterson, G., Reif, J.: Multiple-person alternation, *Proc. 20th FOCS* (1979), pp. 348-363.

[26] Seiferas, J., Fischer, M., Meyer, A.: Separating Nondeterministic Time Complexity Classes, *JACM* **25** 1 (1978), pp. 146-167.

[27] Shamir, A.: IP=PSPACE, *Proc. 31th FOCS* (1990).

[28] Simon, J.: On Some Central Problems in Computational Complexity, Ph.D. Thesis, Cornell University, Computer Science, Tech Report TR 75-224, (1975).

[29] Toda, S.: On the computational power of PP and $\oplus P$, in: *Proc. 30th FOCS*, 1989, pp. 514-519.

[30] Valiant, L. G.: The complexity of computing the permanent, *Theoretical Computer Science* **8** (1979), 189-201.