Comput Complexity 1 (1991), 3–40 1016-3328/91/010003-38 $1.50+0.20

NON-DETERMINISTIC EXPONENTIAL TIME HAS TWO-PROVER INTERACTIVE PROTOCOLS

László Babai, Lance Fortnow
and Carsten Lund

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. We show that the class of languages having two-prover interactive proof systems is nondeterministic exponential time.

We also show that to prove membership in languages in EXP, the honest provers need the power of EXP only.

The first part of the proof of the main result extends recent techniques of polynomial extrapolation used in the single prover case by Lund, Fortnow, Karloff, Nisan, and Shamir.

The second part is a *verification scheme for multilinearity* of a function in several variables held by an oracle and can be viewed as an independent result on *program verification.* Its proof rests on combinatorial techniques employing a simple isoperimetric inequality for certain graphs:

Subject classifications. 68Q15, 68Q60

1. Introduction

The concept of NP was introduced in the early 70's as a model of languages with efficient proof of membership (Cook [16], Levin [27]). As an extension of this concept, two variants of single prover interactive proofs were introduced in 1985 by Babai [4] and Goldwasser, Micali, Rackoff [23]. The power of this extension has not been recognized until very recently, when combined work of Lund, Fortnow, Karloff, Nisan [29], and Shamir [35] 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 [31]).

This paper looks at the class MIP of languages that have multiple-prover interactive proof systems. Ben-Or, Goldwasser, Kilian and Wigderson [11] 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. Ben-Or et al. [11] 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 [19]). 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 [15] 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 [21], 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:* $MIP = NEXP$. After the $IP = PSPACE$ result, this represents a further step demonstrating the unexpectedly immense power of randomization and interaction in efficient provability.

THEOREM 1.1. (MAIN THEOREM) $MIP = NEXP$. *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.*

Ben-Or et al. [11] in fact show 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.

REMARK 1.2. 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.

We also show that to prove membership in languages in EXP, the honest provers need the power of EXP only (Corollary 4.10). A consequence, linking more standard concepts of structural complexity, states that if EXP has polynomial size circuits then $EXP = MA$ (Theorem 6.8), strengthening a result of A. Meyer that $EXP = \Sigma_2^P$ under the same condition. – Instance–checking and self-testing/self-correcting properties (in the sense of Blum, Kannan [12] and Blum, Luby, Rubinfeld [13]) of EXP-complete languages also follow (see the discussion in Section 6).

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 [34]). 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 [25]), thus eliminating the hopes for an easy separation.

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 [32] 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 [36] and Orponen [30] 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 [35]; 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 several 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 [12], Blum, Luby, Rubinfeld [13], and Lipton [28] (see Section 6).

The reduction to the test involves ideas of the $PSPACE = IP$ proof (arithmetic extrapolation of truth values). The proof of correctness of the multilinearity test rests on combinatorial techniques. A more efficient multilinearity test, with important consequences, has been found recently by Mario Szegedy [38].

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 [11] and Fortnow, Rompel, Sipser [21]. 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 [23], 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.

With the exception of Section 5, $n = |x|$ will denote the *length of the input* throughout the paper.

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

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

2. If $x \notin L$ then for all provers $P_1', \ldots, P_k'$, $\Pr(P_1', \ldots, P_k'$ make V accept $x) < 2^{-n}$

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

For easier reading, we introduce some terminology. The functions $P_1, \ldots, P_k$ will be called the *honest provers*; any other collection of provers is *dishonest*. Although not required by the formal definition, we may assume that the honest provers don't attempt to get x accepted when in fact $x \notin L$. (They print a special symbol upon which V automatically rejects.) We shall also use the term "the provers *win*" to indicate that V accepts. We allow dishonest provers to have a good chance of winning but only if $x \in L$. On the other hand, dishonest provers may lose with large probability even if $x \in L$.

REMARK 2.1. It is implicit in this definition that the Verifier's coins are *private*; the Provers receive only information computed by the Verifier based on previous messages and the coin flips.

REMARK 2.2. It would seem natural to suggest that the *timing* of the messages could convey information. This is expressly excluded in the definition (the provers are *functions* of all the information printed on their tapes), but this restriction can be overcome by requiring the protocol to be *oblivious* in the sense that for every i, the sender, the recipient, and the time of the i^{th} message is determined in advance by the length of the input and regardless of the outcome of the communication. This modification would not change the class MIP.

Let M be a probabilistic polynomial time Turing machine with access to an oracle $\mathcal{O}$. 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 $\mathcal{O}$ such that $M^{\mathcal{O}}$ accepts x with probability $> 1 - \frac{1}{p(|x|)}$ for all polynomials p and x sufficiently large.

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

One way to think about this model is that the *oracle convinces* M (by way of overwhelming statistical evidence) to accept. This differs from the standard interactive protocol model in that the oracle must be set ahead of time (it is a fixed function) while in an interactive protocol the prover can be adaptive (he can make his future answers depend on previous questions). This seemingly slight difference accounts for the apparently huge increase in power, from $IP = PSPACE$ to $MIP = NEXP$. The oracle can be thought of as a very long proof of a theorem, which the Verifier can rapidly check. This aspect of this concept will be developed and refined in [7].

THEOREM 2.3. *(Fortnow, Rompel, Sipser [21]) L is accepted by a probabilistic oracle machine if and only if L is accepted by a multi-prover interactive protocol.*

A further important result states that *two provers always suffice.*

THEOREM 2.4. *(Ben-Or, Goldwasser, Kilian, Wigderson [11]) If a language L is accepted by a multi-prover interactive protocol then L is accepted by a two-prover interactive protocol.*

We combine the proofs of these two theorems.

PROOF.

1. First we show how to simulate a probabilistic oracle machine by two provers.

   Let M be a probabilistic oracle machine that runs in time n^c. Have the verifier of the two-prover protocol ask all the oracle questions of Prover 1, then pick one of the questions asked at random and verify the answer with Prover 2. The probability that cheating provers are not caught this way is at most $(1 - n^{-c})$. Repeat this process n^{c+1} times to reduce error probability below $(1 - n^{-c})^{n^{c+1}} < e^{-n}$. (We should stress that we are not assuming that the Provers' responses in a later round would not depend on the messages exchanged in previous rounds. In fact, the result remains valid even if between the rounds, the Provers are allowed to communicate with each other. It is the conditional probability of their success in any particular round, conditioned on arbitrary history of previous communication, that is less than $(1 - n^{-c})$.)

2. Suppose now that L is accepted by a multi-prover interactive protocol. Then define M as follows: Have M simulate V with M remembering all messages. When V sends the jth message to the ith prover, M asks the oracle the question $(i, j, \ell, \beta_{i1}, \ldots, \beta_{ij})$ properly encoded and uses the response as the ℓth bit of the jth response from prover i where $\beta_{i1}, \ldots, \beta_{ij}$ is everything prover i has seen at that point.

   (a) If $x \in L$ then the oracle $\mathcal{O}$ could convince M to accept by just encoding each prover's answer to each question.

   (b) If an oracle $\mathcal{O}$ could convince M to accept a string x then the provers could convince the verifier to accept by just using that $\mathcal{O}$ to create their responses.

   (The full details of this proof can be found in [20].) □

3. Arithmetization: a Variant of the LFKN Protocol

The purpose of the first half of this section is largely didactical. We describe a variant of the LFKN protocol using ideas from Babai, Fortnow [6] (cf. also Shamir [35]). The reader needs to thoroughly understand this protocol before moving on to the proof of the Main Theorem. At the end of this section we derive a lemma which will be used directly in the proof of the Main Theorem.

3.1. Arithmetization. We describe an interactive protocol for *coNP*. We have to show how the Prover convinces the Verifier that a given Boolean formula is *not* satisfiable.

A *Boolean function* in m variables is a function $\{0,1\}^m \to \{0,1\}$.

We say that a polynomial $f(x_1, \ldots, x_m)$ (over some field) is an *arithmetization* of the Boolean function $B(u_1, \ldots, u_m)$ if on all $(0,1)$-substitutions, the (Boolean) value of B and the (arithmetic) value of f agree.

A *Boolean formula* is a well-formed expression built from the constants $0, 1$ and variable symbols using the operations $\wedge, \vee, \neg$. A Boolean formula represents a Boolean function in the obvious sense.

An *arithmetic formula* is a well-formed expression built from the constants $0, 1$ and variable symbols using the operations $+, -, \times$. An arithmetic formula represents a polynomial function in the obvious sense over any commutative ring with identity.

PROPOSITION 3.1. *Given a Boolean formula B, one can construct in linear time an arithmetic formula f which will represent an arithmetization of B over any field (and indeed over any ring with identity).*

PROOF. We eliminate the $\vee$'s in B, replacing them by $\neg$'s and $\wedge$'s. We then replace Boolean variable symbols by arithmetic variable symbols, $\wedge$'s by $\times$'s and subexpressions of the form $\neg g$ by $(1 - g)$. □

REMARK 3.2. We note that the arithmetization we obtained of a Boolean formula of length d is a polynomial of degree less than d.

Let now B be the Boolean formula which the prover claims is not satisfiable. Let f be the arithmetization of B constructed above. We view f as a polynomial over $\mathbf{Z}$ (an integral domain). The prover has to convince the verifier that f vanishes on all $(0,1)$-substitutions. The fact that each member of this *exponentially large* collection of quantities vanishes, can be expressed concisely as follows:

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_m=0}^{1} f(x_1, \ldots, x_m)^2 = 0 \qquad (3.1)$$

Recall that the polynomial f is given to the Verifier in the form of an explicit arithmetic expression.

3.2. The LFKN variant. We describe how the Prover convinces the Verifier of the validity of the slightly more general equality

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_m=0}^{1} h(x_1, \ldots, x_m) = a \tag{3.2}$$

where a is a given number and h is a polynomial of degree $\leq d$ in each variable, given as an explicit arithmetic expression. The protocol works over an arbitrary field (or integral domain with identity) of order $\geq 2dm$.

PROPOSITION 3.3. *The set of pairs (h, a), where h is an arithmetic formula and a is a number such that equation (3.2) holds, belongs to IP.*

Let $\mathbb{I}$ be a sufficiently large subset of the field: $|\mathbb{I}| \geq 2dm$. (We extend the field of definition if necessary.) For $i = 0, 1, \ldots, m$ we shall consider the partial sums

$$h_i(x_1, \ldots, x_i) := \sum_{x_{i+1}=0}^{1} \cdots \sum_{x_m=0}^{1} h(x_1, \ldots, x_m). \tag{3.3}$$

Clearly, $h_m = h$, and

$$h_{i-1} = h_i(x_i = 0) + h_i(x_i = 1) \tag{3.4}$$

(using self-explanatory notation for substitution).

The protocol to verify (3.2) proceeds in rounds. There are m rounds.

At the end of round i, the Verifier picks a random number $r_i \in \mathbb{I}$; and computes a *"stated value"* b_i. We set $b_0 = a$. The Prover will maintain that for each i, including $i = 0$,

$$b_i = h_i(r_1, \ldots, r_i). \tag{3.5}$$

So by the beginning of round $i \geq 1$, the numbers $r_1, \ldots, r_{i-1}$ have been picked and the "stated values" $b_0 = a$, $b_1, \ldots, b_{i-1}$ have been computed.

Now the Prover is requested to state the coefficients of the univariate polynomial

$$g_i(x) = h_i(r_1, \ldots, r_{i-1}, x). \tag{3.6}$$

Let $\tilde{g}_i$ denote the polynomial stated by the Prover. The Verifier performs a **Consistency Test**; with equation (3.4) in mind, he checks the condition

$$b_{i-1} = \tilde{g}_i(0) + \tilde{g}_i(1). \tag{3.7}$$

If this test fails, the Verifier rejects; else he generates the next random number $r_i \in \mathbb{I}$ and declares $b_i := \tilde{g}_i(r_i)$ to be the next "stated value". After the m^{th}

round we have the stated value b_m and the random numbers $r_1, \ldots, r_m$; and the verifier performs the **Final Test**

$$b_m = h(r_1, \ldots, r_m). \tag{3.8}$$

The Verifier accepts if all the m Consistency Tests as well as the Final Test have been passed.

The proof of correctness exploits the basic idea that if the Prover wants to cheat, he is likely to be forced to cheat on polynomials with fewer and fewer variables; eventually reaching a constant, the correctness of which the Verifier can check by a *single* substitution into the explicit polynomial behind the summations.

PROOF OF CORRECTNESS OF THE PROTOCOL. Assume first that (3.2) holds. Then the *honest* Prover will always answer correctly ($\tilde{g}_i = g_i$) and win.

Assume now that at some point, the Prover *cheats:* $\tilde{g}_{i-1} \neq g_{i-1}$. Here we allow $i = 1$; we define the constant polynomial $\tilde{g}_0 := b_0 = a$. Then with probability $\geq 1 - m/|\mathsf{I}|$, $b_{i-1} = \tilde{g}_{i-1}(r_{i-1}) \neq g_{i-1}(r_{i-1})$ since two different univariate polynomials of degree $\leq d$ cannot agree at more than d places. Assuming now that the Prover passes the next Consistency Test (3.7), it follows that he must cheat in the next round: $\tilde{g}_i \neq g_i$.

If now (3.2) does not hold, then the constant $a = b_0 = \tilde{g}_0$ differs from $g_0 = h_0$, hence the Prover automatically cheats in round 0. It follows that with probability $\geq 1 - dm/|\mathsf{I}|$, he will be forced to cheat in each round. But cheating in the last round is discovered by the Final Test. □

3.3. Oracle polynomials and oracle protocols. We now wish to drop the condition in Proposition 3.3 that the polynomial h is given by an explicit arithmetic formula. Instead we want the Verifier to access the values of h from an oracle.

To formalize this, we have to introduce the concept of an *interactive oracle-protocol.* This is the same as a two-prover interactive protocol except the second prover is restricted to be a *function*, *i.e.*, its responses must be nonadaptive. In this case we call the second prover the *oracle* and we view the protocol a single prover protocol, where the Verifier has random access to the Oracle, and the Prover wishes to convince the Verifier in polynomial time that the Oracle has a certain property.

Since slight changes in the oracle will not be noticed by the Verifier, global properties of h cannot be verified this way, but under certain conditions well-behaved approximations of h can be ascertained to have certain global properties. "Well-behaved" will mean low degree polynomials; the key idea being that such polynomials form an error-correcting code (cf. Remarks 3.6, 3.7). The notion of approximation is defined as follows.

DEFINITION 3.4. Let f, g be functions over a finite set X. For $\delta \in [0, 1]$ we say that f *δ-approximates* g if the number of places $x \in X$ such that $f(x) \neq g(x)$ is less than $\delta|X|$.

LEMMA 3.5. *Let $d, m, k \geq 0$, a field F and a subset $\mathsf{I} \subseteq \mathsf{F}$, $|\mathsf{I}| = kdm$ be the input. Suppose the Oracle accepts queries of the form $(r_1, \ldots, r_m)$ $(r_i \in \mathsf{I})$, and responds with an element $h(r_1, \ldots, r_m) \in \mathsf{F}$ of polynomial length. There exists an interactive oracle-protocol such that*

(i) if h is a polynomial of degree $\leq d$ in each variable and (3.2) holds then there exists a Prover which the Verifier surely accepts;

(ii) if h is δ-approximated on I^m by some polynomial f which has degree $\leq d$ in each variable ($\delta \leq 1/4$) and if there exists a Prover which has greater than $\delta + 1/k$ chance of being accepted then (3.2) holds with f in place of h.

REMARK 3.6. The Lemma says that the validity of (3.2) can be verified in polynomial time with large confidence assuming h has low degree; and even if h itself is "forged" from some low degree polynomial f by changing the values of f at a fixed positive fraction of the inputs, it can be verified that (3.2) holds for the correct f. This *error-correcting property* of the protocol is related to the next remark.

REMARK 3.7. A well-known lemma of Jacob Schwartz asserts that *a nonzero polynomial of total degree $\leq d$ vanishes at no more than a $d/|\mathsf{I}|$ fraction of* I^m (m is the number of variables) [37] (cf. [9, Lemma 2.35]). (The proof is a simple induction on m.) It follows that given h, its low degree correction (if exists) is unique. Indeed, assume both f_1 and f_2 are δ-approximations of h, and let $f = f_1 - f_2$. Then f is a low degree polynomial (degree $\leq d$ in *each* variable) which vanishes on all but a 2δ fraction of the inputs from I^m. By Schwartz's Lemma, f is identically zero unless $2\delta \geq 1 - dm/|\mathsf{I}|$. So if $k \geq 2$ and $\delta \leq 1/4$ then the correction of h is unique. (This is a multi-variable version of the principle of the Reed–Solomon codes.)

PROOF OF LEMMA 3.5. We perform the LFKN-type protocol as described in Section 3.2. The only difference is that in the *Final Test*, the Verifier makes a query to the Oracle rather than evaluating $h(r_1, \ldots, r_m)$ himself.

We prove that the protocol has the properties stated in the Lemma. Property (i) is straight forward: just as in the proof of Proposition 3.3, the honest Prover will always win.

In order to prove Property (ii), let f be the unique polynomial of degree $\leq d$ in each variable which δ-approximates h. Assume that (3.2) does not hold for f in place of h. Let us pretend for a moment that the Oracle holds f rather than h. Then, according to the analysis in the proof of Proposition 3.3, the *Consistency Tests* will force the Prover to cheat in each round, including the last one, with probability $\geq 1-dm/|\mathrm{I}| = 1-1/k$. At this point the only possible rescue for the Prover is that f and h may differ on the random substitution $(r_1, \ldots, r_m)$. But the probability of this is at most δ, hence with probability $\geq 1-(\delta+1/k)$, the Prover will be caught. □

An apparent weakness of this result is that we have to assume that h is a low degree polynomial, or at least a good approximation of such a polynomial. One of the main technical results of this paper is that this circumstance can be checked by the randomizing Verifier (see Section 5). Apart from this problem, the main task in proving the Main Theorem (Theorem 1.1) is to reduce it to the simultaneous vanishing of a low degree polynomial, held by an oracle, over all (0,1)-substitutions.

3.4. Implementing oracle-protocols with two provers. Oracle-protocols, as defined in Section 3.3, represent a compromise between the extremes of two-prover protocols and probabilistic oracle machines, shown to be equivalent in Section 2. Not surprisingly, oracle protocols are equal to both in power. This is clear from the aforegoing. Indeed, on the one hand, an oracle protocol can simulate a probabilistic oracle machine simply by adding a dummy prover. On the other hand, the Oracle part of an oracle protocol can be simulated by two provers as dicussed in the proof of Theorem 2.4 resulting in 3 provers which can then be reduced to two by the results stated in Section 2.

In fact, the latter simulation can further be simplified as follows.

Suppose the language L is accepted by an oracle-protocol and x is an input. To simulate the oracle protocol with two provers, we execute the entire protocol with Prover 1 including the queries to the Oracle. We then choose a random question we have asked Prover 1 about the Oracle and ask this question to Prover 2. If the answers differ then we reject.

As in the proof of Theorem 2.4, we observe that this protocol guarantees at least an $n^{-O(1)}$ chance for cheating provers (*i.e.*, $x \notin L$) to be caught. Repeating the process a polynomial number of times time results in an exponentially small probability that cheating provers could get away.

4. 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 $MIP \subseteq NEXP$ ([21], see the comment before the statement of Theorem 1.1) we have to prove $NEXP \subseteq MIP$.

4.1. 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; [16], [27], cf. [1, p. 385]) There will be an exponential number of variables and an exponential number of clauses. However, the clauses are easily definable, in fact there exists a polynomial-time computable function $f_x(i)$ that describes the variables of clause i. Thus $L(M) = \{x|$ there is an assignment of variables A such 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. (An easy implementation of this in our context, along the lines of the proof of Theorem 2.3, will be given in Section 3.4.)

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 [29], Babai, Fortnow [6], and Shamir [35], but these results do not relativize and A may not have the proper algebraic properties necessary for this proof.

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

4.2. A $NEXP$-complete language. For the purposes of Section 4, we adopt some notational conventions. We shall use lower case letters x, b, f, t, z, w (possibly subscripted) for strings of variables of *polynomially bounded length*; the corresponding individual variables will be denoted $\xi_i, \beta_i, \varphi_i, \tau_i, \zeta_i, \omega_i$. A typical variable in the string b_j will be β_{ji}. These variables will be either Boolean or belong to the field $\mathbf{Q}$. We view the Boolean domain $\{0,1\}$ as a subset of $\mathbf{Q}$.

DEFINITION 4.1. Let r, s be nonnegative integers; let z and b_i $(i = 1,2,3)$ be strings of variables, where $|z| = r$ and $|b_i| = s$. For brevity, let $b = b_1b_2b_3$ and $w = zb$ (juxtaposition indicates concatenation). Let $t = \tau_1\tau_2\tau_3$ be a string of 3 variables. Let $B(w,t)$ be a Boolean formula in $r + 3s + 3$ variables. We say that a Boolean function A in s variables is a *3-satisfying oracle* for B if $B(w, A(b_1), A(b_2), A(b_3))$ is true for each of the 2^{r+3s} Boolean substitutions into the string w of $r + 3s$ variables. We say that B is *oracle-3-satisfiable*, if such a function A exists. The *oracle-3-satisfiability* problem takes a Boolean formula B as input together with the integers r, s and accepts it if it is oracle-3-satisfiable.

PROPOSITION 4.2. *Oracle-3-satisfiability is NEXP-complete.*

PROOF. Clearly, this language belongs to $NEXP$. Let now $L \in NEXP$ and x an input of length n for the membership problem in L. We construct in polynomial time an instance (B, r, s) of oracle-3-satisfiability which is accepted if and only if $x \in L$.

The first part of this construction is essentially due to J. Simon [36]; similar proofs appear in Peterson, Reif [32] and Orponen [30], describing $NEXP$ analogues of the Cook–Levin theorem (NP-completeness of 3-SAT).

Let M be the $NEXP$ Turing machine accepting L. Look at the tableau describing the computation of M on input x. Convert this to a 3-CNF Φ_x like in the proof of the Cook–Levin theorem. There will be an exponential number N_v of variables and an exponential number N_c of clauses. For sake of simplicity assume without loss of generality that $N_v = 2^s$ where $s = n^c$ for some constant c. We label the variables by binary strings of length 2^s: $X(b), b \in \{0,1\}^s$. There are 2^{3s+3} possible clauses with 3 signed variables each (a *signed variable* is a variable or its negation). A typical clause has this form:

$$C(b,f,X) = (\varphi_1 \oplus X(b_1)) \vee (\varphi_2 \oplus X(b_2)) \vee (\varphi_3 \oplus X(b_3)), \qquad (4.9)$$

where the 3-bit string $f = \varphi_1\varphi_2\varphi_3$ encodes the signs of the variables, and the $3s$-bit string $b = b_1b_2b_3$ encodes the variables themselves. ($\varphi = 1$ stands for negation and $\varphi = 0$ for the absence of it; $\oplus$ denotes addition mod 2.)

The clauses themselves are polynomial time recognizable, *i.e.*, there is a polynomial time computable predicate p such that $C(b,f,X)$ is a clause of Φ_x if and only if $p(x,b,f)$ holds. We infer that $x \in L$ if and only if there exists a Boolean function $A : \{0,1\}^s \rightarrow \{0,1\}$ (a satisfying instance) such that for every $b \in \{0,1\}^{3s}$ and $t \in \{0,1\}^3$,

$$D(b,f,A) =: C(b,f,A) \vee \neg p(x,b,f) \qquad (4.10)$$

holds. (Those clauses which belong to Φ_x according to p must be satisfied when the variables $X(b_i)$ are replaced by the Boolean values $A(b_i)$.)

Observe that $C(b, f, A)$ is obtained from the explicit Boolean formula

$$B_1(f,t) := (\varphi_1 \oplus \tau_1) \vee (\varphi_2 \oplus \tau_2) \vee (\varphi_3 \oplus \tau_3) \tag{4.11}$$

by substituting $A(b_i)$ for τ_i.

We now wish to replace $\neg p(x, b, f)$ by a Boolean formula. To this end we regard $p(x, b, f)$ as computable in NP, and apply Cook–Levin to obtain an equivalent 3-SAT instance B_2. The 3-CNF formula B_2, computable in polynomial time from x, involves the variable strings b, f and u, the latter being the "witness" (of polynomial length). Having fixed the values of b, f, we observe that B_2 is satisfiable if and only if $p(x, b, f) = 1$.

Let finally

$$B(u,b,f,t) := B_1(f,t) \vee \neg B_2(u,b,f). \tag{4.12}$$

This is the Boolean function we have sought. (To consolidate with the notation of Definition 4.1 let $z = uf$ and let r denote the length of z.) When does a function $A : \{0,1\}^s \to \{0,1\}$ 3-satisfy B? For every $b = b_1b_2b_3$, the substitutions $\tau_i = A(b_i)$ must satisfy B for all possible values of z. But this is precisely what we have shown to be equivalent to $x \in L$. □

4.3. Arithmetization of *NEXP*. We use the same variable symbols as introduced at the beginning of Section 4.2. For the definition of *arithmetization*, we refer to Section 3.1.

LEMMA 4.3. *Given an instance (B, r, s) of oracle-3-satisfiability (where $B = B(w,t)$ is a Boolean formula in $r + 3s + 3$ variables), one can compute in polynomial time an arithmetic expression for a polynomial g with integer coefficients over the same set of $r + 3s + 3$ variable symbols such that a function $A : \{0,1\}^s \to \mathbb{Q}$ constitutes a 3-satisfying oracle for B if and only if*

$$\sum_{w \in \{0,1\}^{r+3s}} g(w, A(b_1), A(b_2), A(b_3)) = 0. \tag{4.13}$$

PROOF. First we use Proposition 3.1 to obtain an arithmetic expression for a polynomial f representing an arithmetization of B. Next, set

$$g(w,t) := (f(w,t))^2 + (\tau_1(\tau_1 - 1))^2. \tag{4.14}$$

Since now the left hand side of (4.13) is a sum of squares, that sum will vanish if and only if all terms vanish.

The vanishing of the last term of the right hand side of (4.14), *i.e.,* the relation $A(b_1)(A(b_1)-1)=0$, corresponds to the guarantee that all values of A are from $\{0,1\}$. Assuming now that this is the case, the vanishing of the first term on the right hand side of (4.14) is by definition equivalent to the relation $B(w,A(b_1),A(b_2),A(b_3))=0$. □

4.4. Multilinearity. Let $A:\{0,1\}^s \to \mathbf{Q}$ be a function stored by the Oracle, and let $h(w):=g(w,A(b_1),A(b_2),A(b_3))$. Our task is to verify (4.13). In order to be able to use an LFKN-type protocol as described in Section 3, we have to turn A into a polynomial of low degree. There is a very simple way to do so. A polynomial is *multilinear* if it is linear in every variable.

PROPOSITION 4.4. *Let $A:\{0,1\}^s \to \mathbf{Q}$ be a function. Then A has a unique multilinear extension $\tilde{A}:\mathbf{Q}^s \to \mathbf{Q}$. If A takes integer values then so does $\tilde{A}$ over $\mathbf{Z}^s$.*

PROOF. Define $\tilde{A}$ by

$$\tilde{A}(x) =: \sum_{b\in\{0,1\}^s} \prod_{i=1}^{s} A(b)\ell_{\beta_i}(\xi_i), \tag{4.15}$$

where $x=(\xi_1,\dots,\xi_s)$; b is the bit-string $\beta_1\dots\beta_s$; and $\ell_0(\xi)=1-\xi$, $\ell_1(\xi)=\xi$. Clearly, $\tilde{A}$ possesses the required properties.

To prove the uniqueness, assume $f:\mathbf{Q}^s \to \mathbf{Q}$ is multilinear and its restriction to $\{0,1\}^s$ is zero. For $x\in\mathbf{Q}^s$, let $k(x)$ denote the number of coordinates different from 0,1. We prove by induction on $k(x)$ that $f(x)=0$. Indeed this is true by assumption for $k(x)=0$. Now for some $k(x)>0$ suppose e.g. that $\xi_1\notin\{0,1\}$. Replacing ξ_1 by either 0 or 1 we obtain places where f vanishes by the induction hypothesis; but then, by the linearity in ξ_1, it vanishes at x as well. □

REMARK 4.5. The proof works over any integral domain with identity.

Now we are ready to apply the oracle version of the LFKN protocol from Lemma 3.5. What we need is that the Oracle store not just A, the oracle 3-satisfying B, but the multilinear extension of A to a suitable domain $\mathbf{I}^s$. The set $\mathbf{I}\subset\mathbf{Z}$ has to be large enough for the LFKN protocol to work. Multilinearity of A will guarantee that the function $h(w)=g(w,\tilde{A}(b_1),\tilde{A}(b_2),\tilde{A}(b_3))$ is a polynomial of low degree and the LFKN protocol verifies (4.13).

The difficulty with this approach is that a dishonest Oracle may cheat by storing a function that is not multilinear.

This question will be addressed by a separate protocol in Section 5. That protocol will ask simple randomized questions to the Oracle. If the function $A : \mathbb{I}^s \rightarrow \mathbb{Q}$ stored by the Oracle is multilinear, the protocol will always accept. On the other hand, if the protocol has $\geq 1/2$ chance of accepting, then the function stored by the Oracle is at least an ϵ-approximation of a multilinear function $\mathbb{I}^s \rightarrow \mathbb{Q}$ (cf. definition 3.4).

We state the result here. We say that a function $\mathbb{I}^s \rightarrow \mathbb{Q}$ is *ϵ-approximately multilinear* if it is an ϵ-approximation of a multilinear function. For typographic convenience, we use $\exp_2(u)$ to denote 2^u.

THEOREM 4.6. *Let $d \geq 1$ and $k \geq 1$ be fixed constants. Let N be an integer, $s^{d+3} < N \leq 2^{s^k}$ for some s. Let $\mathbb{I}$ denote the set of integers $\{0, \ldots, N-1\}$. Let $A(\xi_1, \ldots, \xi_s)$ be an arbitrary function from $\mathbb{I}^s$ to $\mathbb{Q}$. 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 $\exp_2(s^{k'})$ then M^A always accepts.*
2. *If A is not s^{-d}-approximately multilinear then with high probability M^A rejects.*

The proof of this result will be the subject of Section 5. The following observation justifies the upper bound posed on the values of A in statement #1 in the Theorem.

PROPOSITION 4.7. *If $\mathbb{I} = \{0, \ldots, N-1\}$ and $A : \mathbb{I}^s \rightarrow \mathbb{Q}$ is the multilinear extension of a Boolean function then for any $x \in \mathbb{I}^s$, the absolute value of $A(x)$ is bounded by*

$$|A(x)| < (2N)^s. \tag{4.16}$$

PROOF. Immediate from equation (4.15). □

REMARK 4.8. Theorem 4.6 has applications to program testing and correcting. We shall elaborate on this in Sections 6.3 and 6.4.

REMARK 4.9. 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 $A : \mathbb{I}^n \rightarrow \mathbb{Z}$ is a polynomial having degree $\leq k_i$ in variable x_i for every i. Theorem 4.6 extends to this situation, with only trivial modifications in the proof.

4.5. The Protocol. Let $L \in NEXP$. We have to design a MIP protocol to verify $x \in L$. As described in Section 3.3, what we actually construct is an *interactive oracle-protocol.* Section 3.4 describes a simple implementation of such a protocol with two provers.

We shall thus have a single Prover and an Oracle. According to Proposition 4.2 the Verifier constructs an instance (B, r, s) of oracle 3-satisfiability which is accepted if and only if $x \in L$. Next, following Lemma 4.3 the Verifier constructs an arithmetic expression for a polynomial g in $r + 3s + 3$ variables with integer coefficients such that $x \in L$ if and only if there exists a Boolean function $A : \{0,1\}^s \rightarrow \{0,1\}$ such that equation (4.13) holds.

We select reliability parameters $\delta \leq 1/4$ and $k \geq 4$ such that $1/\delta + k \leq n^{O(1)}$ $(n = |x|)$. We set $\mathsf{I} = \{0, \ldots, N-1\}$, where $N = kdm$ where d is the degree of g and $m = r + 3s$. (Clearly, all these parameters are bounded by $n^{O(1)}$.)

We ask the Oracle to store a function $A : \mathsf{I}^s \rightarrow \mathbb{Z}$ which is supposed to be a multilinear extension of a 3-satisfying oracle for (B, r, s).

Phase One of the protocol is the *multilinearity test.* This phase does not require the Prover; the Verifier will ask a polynomial number of randomized questions to the Oracle according to Theorem 4.6. If this phase ends in rejection, the Verifier rejects the claim $x \in L$ and the protocol terminates.

Phase Two is invoked if the multilinearity test ends with acceptance. This phase is intended to verify (4.13). This is accomplished via the LFKN-type interactive oracle-protocol stated in Lemma 3.5, applied to the function $h(w) = g(w, A(b_1), A(b_2), A(b_3))$. The *Final Test, i.e.,* the last step of that protocol requires the evaluation of $h(w)$ at a single random $w \in \mathsf{I}^m$ where $m = r + 3s$. The Verifier accomplishes this by making three queries to the Oracle: $A(b_1), A(b_2), A(b_3)$. (In Phase Two, these are the only queries to the Oracle.) Note that these three places have been chosen at random from I^s by the Verifier. According to the outcome of Phase Two, the Verifier accepts or rejects the claim $x \in L$.

PROOF OF CORRECTNESS. If $x \in L$, then the honest Prover–Oracle pair will clearly always win.

Suppose now that a Prover–Oracle pair has greater than $\delta + 1/k$ chance of winning. We claim that then $x \in L$.

First of all, the Oracle A has to be at least $\delta/3$-approximately multilinear; otherwise it would be rejected in Phase One with high probability. Let A' be a multilinear function which $\delta/3$-approximates A.

Let $h(w) = g(w, A(b_1), A(b_2), A(b_3))$ and $h'(w) = g(w, A'(b_1), A'(b_2), A'(b_3))$. Clearly, h is δ-approximated by h' over I^m, and h' has degree $\leq d$ in each of its m variables. Therefore, according to Lemma 3.5, if the Prover has greater than

$\delta + 1/k$ chance of winning, then (4.13) holds with A' in the place of A. This means that A' is a 3-satisfying oracle for (B, r, s), thus proving that $x \in L$. □

This concludes the proof of the Main Theorem modulo the multilinearity test which follows in Section 5.

4.6. The Power of the Provers. We state a by-product of the above proof regarding the required power of the provers. Let $\mathcal{C}$ be either a 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 $x \in L$, the honest provers P_i are restricted to answering questions of membership in some language $L_i \in \mathcal{C}$, and are able to convince the verifier about membership of x in L;
- Even all-powerful provers do not have a chance of convincing the verifier of 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 [18] combined with Shamir's [35] implies that each language in $PSPACE$ has a single prover interactive proof system with a prover of complexity $PSPACE$. The result of Lund et al. [29] 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 4.10. *For any $L \in EXP$, there is a multiple-prover interactive proof system with provers of complexity EXP.*

PROOF. Notice that the tableau of the computation performed by a deterministic exponential-time machine M on a specific input x is unique and any bit of that tableau can be computed in deterministic exponential time by simulating the computation of $M(x)$. In the proof of Proposition 4.2 we reduce $x \in L$ to an instance of oracle-3-satisfiability where a satisfying instance can be computed in deterministic exponential time. From this the result is immediate. □

REMARK 4.11. Although we know that all languages provable in a multi-prover proof system must lie in $NEXP$ we do not know whether $NEXP$ provers are sufficient to prove any $NEXP$ language to a verifier. It would be important that all provers have access to the same tableau of an accepting computation; but there could be several since the $NEXP$ machine may have many accepting paths. The best upper bound we know on the power of the provers for $NEXP$ is EXP^{NP}, pointed out by Gábor Tardos. Indeed, it is easy to see that an EXP^{NP}-machine is capable of computing a lexicographically first tableau for any $NEXP$ language. (The computation proceeds by sequentially asking every bit of the lexicographically first accepting tableau in the form of an exponentially long (padded) question to the NP orcale.)

Also of interest is the power of provers needed to prove a $coNP$-complete language like DNF tautology. Lund et al. [29] show that $\#P$ provers are sufficient. We know of no better bound.

5. Verification of Multilinear Functions and Polynomials of Low Degree

First we need some definitions and notation.

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

We shall consider the nth 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 $i_1, \dots, i_{n-k}$ and $n-k$ values of these coordinates $\alpha_{i_1}, \dots, \alpha_{i_{n-k}} \in X$ such that

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

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 ith 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 5.1. *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 5.2. *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 5.3. *Given a function $A : \mathbb{I}^n \to \mathbb{Q}$, we call a line ℓ in $\mathbb{I}^n$* 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.*

5.1. 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 $x \in \mathbb{I}^n$, $A(x)$ 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 $\mathbb{I}^n$. 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 5.4. *(a) If ℓ is correct, then Test$_0$ surely accepts ℓ.*

(b) If ℓ is δ-wrong, Test$_0$ will reject it with probability greater than $1 - \exp(-\delta m_1)$.

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

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

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 5.5. *(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 δ-wrong, then the probability that **Test** *rejects is greater than*

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

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

We paraphrase statement (b) above.

PROPOSITION 5.6. *Given a function $A : \mathsf{I}^n \to \mathbb{Q}$, 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*

$$(\forall i) \text{ the proportion of } \delta\text{-wrong lines among } L_i \text{ is } < \epsilon. \qquad (**)$$

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 5.13 (end of this section) for the formal statement of this result.

5.2. The Self-Improvement Lemma. We need a combinatorial isoperimetric inequality.

DEFINITION 5.7. *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 ith dimension.

LEMMA 5.8. (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 [5, Lemma].

The key step in the induction argument that will yield Theorem 5.13 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 5.10. (SELF-IMPROVEMENT LEMMA) *Given a function $A : \mathsf{I}^n \to \mathbb{Q}$, assume that*

$$(\forall i) \text{ the proportion of } \delta\text{-wrong lines among } L_i \text{ is } < \epsilon$$

and

some multilinear function g Δ-approximates A,

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

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

PROOF. We will partition the points of $\mathbb{I}^n$ into four sets. For $S \subseteq \mathbb{I}^n$, we set $\mu(S) = |S|/N^n$.

B: $B = \{x \in \mathbb{I}^n | A(x) \neq g(x)\}$. Call them bad points.

W: Union of δ-wrong lines. Call the points in W wrong. Observe that the assumption gives $\mu(W) < n\epsilon$.

M: Points $x \notin W$ which belong to lines ℓ where $A|_\ell$ is δ-approximated by some linear function h, but $h(x) \neq A(x)$. Call these points misplaced. Since for each line only a δ-fraction is misplaced and since each point lies on n lines we obtain that $\mu(M) < n\delta$.

I: Points x on lines ℓ such that $A|_\ell$ is δ-approximated by a linear function h, where $h \neq g|_\ell$, but $A(x) = g(x) = h(x)$. Since at most one such point belongs to each line, $\mu(I) \leq n/N$.

Now define $S := B \setminus (W \cup M)$. We claim that $\bar{S} \subseteq B \cup M \cup I$. To see this take $\alpha \in S$ and let β be a point on a line ℓ through α. Assume that $\beta \notin B \cup M$. First observe that since α is not a wrong point there is a linear function h which δ-approximates A restricted to ℓ. Since neither α nor β were misplaced and α was bad and β is not bad, we have that $A(\beta)$ belongs to two different linear function g restricted to ℓ and h. Hence $\beta \in I$.

So if $\mu(S) \leq 1/2$ then from the Expansion Lemma we obtain that

$$(1 + \frac{1}{2n})\mu(S) \leq \mu(\bar{S}) < \mu(S) + n(\epsilon + \delta) + n/N,$$

hence $\mu(S) < 2n^2(\epsilon + \delta + 1/N)$. This concludes the proof since

$$\mu(B) < \mu(S) + n(\epsilon + \delta) < 3n^2(\epsilon + \delta + 1/N). \quad \square$$

5.3. 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 5.11. (PASTING LEMMA) *Given a function $A : \mathbb{I}^n \to \mathbb{Q}$, assume that $\delta, \epsilon > 0$, $\epsilon + \delta \leq \frac{1}{200n}$, $N \geq 40n$,*

$$(\forall i) \text{ the proportion of } \delta\text{-wrong lines among } L_i \text{ is } < \epsilon$$

and there exists $g : \mathbb{I}^n \times \mathbb{I}^n \to \mathbb{Q}$ such that $g(x, y)$ is multilinear in $y \in \mathbb{I}^{n-1}$ for every $x \in \mathbb{I}$ and 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 $f_2(y) : \mathbb{I}^{n-1} \to \mathbb{Q}$ such that $xf_1(y) + f_2(y)$ $(\epsilon + \delta)$-approximates $A(x, y)$. Define $\Psi = \{\xi | \xi f_1(y) + f_2(y)$ does not β-approximate $A(\xi, y)$ 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 multilinear functions $g_i(y)$ 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 $A(x, y)$. □

5.4. 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 and 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 with $0 \leq \epsilon_0, \varphi < 1$. Color a vertex red if each of the following two conditions are met (and otherwise color it arbitrarily):

- "Almost all leaves" in the subtree T_v rooted at v are red: only a fraction not greater than ϵ_0 is white.
- A "fair number" of children of v are red: the proportion of red children is at least φ.

LEMMA 5.12. (TREE COLORING LEMMA) *Let $\epsilon_k = (1-\varphi)^k\epsilon_0$. 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.

$k = 0$ This case says "if a leaf is red then it is red."

$k \geq 1$ Take a random child u of v. Recall that we use μ to denote the uniform probability measure over nonempty finite sets. Now $E_u(\mu($ white leaves in $T_u)) \leq \epsilon_k$. Hence $Pr_u(\mu($ white leaves in $T_u) \geq \epsilon_{k-1}) \leq \frac{\epsilon_k}{\epsilon_{k-1}} = 1 - \varphi$. But this probability is by the inductive hypothesis at least $Pr_u[u$ is white$]$. So

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

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

THEOREM 5.13. *Given $A : \mathbf{I}^n \to \mathbf{Q}$, assume that*

$$(\forall i) \text{ the proportion of } \delta\text{-wrong lines among } L_i \text{ is } < \epsilon.$$

Then

$$A \text{ is } \epsilon'\text{-approximately multilinear, where } \epsilon' = 3n^2(\epsilon + \delta + 1/N),$$

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 $\mathbf{I}^n$. (We consider aligned affine subspaces; see the conventions stated at the beginning of this section.) The root corresponds to $\mathbf{I}^n$; 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 δ-wrong line white, all the others red. Now we color the rest of T according 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 5.14. *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 δ-wrong lines in U_v is $< \epsilon_0$. This implies that the proportion of δ-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 5.13. □

Now the proof of Theorem 4.6 is immediate. Let our probabilistic Turing machine perform the Test, setting the parameters so that $\epsilon' \leq n^{-c}$. If A is multilinear, integral valued, and takes no too large values, then the machine will clearly accept. On the other hand, Proposition 5.6 guarantees that in case the machine accepts, condition (**) (Proposition 5.6) can be inferred with high confidence. By Theorem 5.13, this implies that A is n^{-c}-approximately multilinear. □

REMARK 5.15. As we mentioned in Remark 4.9, this test for multilinearity can be extended to polynomials having small degree in each variable, where "small" means bounded by a polynomial of the length of the input. 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 5.13 extends to this situation, with the following modification of the parameters. We should set $\epsilon' = 3n^2(\epsilon + \delta + k/N)$ where $k = \max_i k_i$; furthermore $\beta = \frac{1}{5(k+1)}$, $N \geq 20(k+1)^2 n^2$, $\delta \leq \frac{1}{200n^2(k+1)}$ and $\epsilon \leq \frac{1}{400n^3(k+1)}$. With the obvious modifications, the same proof applies.

REMARK 5.16. Recently, Mario Szegedy [38] succeeded in devising a more efficient protocol for multilinearity (and low degree) testing; at the same time the proof of correctness of his protocol is also simpler.

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 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 6.2. *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 4.4). 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 [10] and spelled out simultaneously by the authors of this paper and of [10], has significant consequences, as we shall see below.

There are natural classes of languages satisfying the conclusion of Lemma 6.2 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 [41]).

6.2. Instance Checking. In Blum, Kannan [12], "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 the function π and any prover (whether honest or not) must be a function from the set of instances to {YES, NO}.

By Theorem 2.3 due to Fortnow, Rompel, Sipser 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 and 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 and 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 [29] and Shamir [35] 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 4.10:

Corollary 6.3. *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 [12] 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 $coNP$ languages have protocols with NP provers (see Section 4.6).

6.3. Self-Testing and Self-Correcting Programs. Our test of multilinear functions (Section 5) also has applications to program testing as described by Blum, Luby, Rubinfeld [13] and Lipton [28].

We will use the following definition of self-testing/correcting programs which is slightly different from but in the spirit of the Blum–Luby–Rubinfeld definition. We make the connection between the two models clear in Section 6.4.

An input set I is a sequence of subsets $I_1, I_2, \dots$ of $\{0,1\}^*$ such that for some k and for all n, if $x \in I_n$ then $n^{1/k} \leq |x| \leq n^k$. We let I represent the set $\bigcup_{n\geq 1} I_n$.

We say a pair of probabilistic polynomial time programs (T, C) is a *self-testing/correcting pair* for a function f over an input set I if given a program $\mathcal{P}$ that purports to compute f the following hold for every n:

1. The tester $T(\mathcal{P}, 1^n)$ will output either "Pass" or "Fail".

2. If $\mathcal{P}$ correctly computes f on all inputs of I then $T(\mathcal{P}, 1^n)$ will say "Pass" with probability at least 2/3.

3. For every $x \in I_n$, if $\Pr(T(\mathcal{P}, 1^n)$ says "Pass"$) > 1/3$ then $\Pr(C(\mathcal{P}, x) = f(x)) > 2/3$.

The errors can be made exponentially small by repeated trials and majority vote. A language has a self-testing/correcting pair if its characteristic function does.

An alternative definition would require the tester to always say "Pass" for a correct program. In every case that we know, the tester has this property. However, we allow the more general definition for a better comparison with the Blum–Luby–Rubinfeld model (see Section 6.4).

THEOREM 6.4. *Every PSPACE-complete and EXP-complete language has a self-testing/correcting pair over* $I = \{I_n | I_n = \{0,1\}^n\}$.

We will prove Theorem 6.4 for EXP-complete languages. The proof for PSPACE-complete languages is analogous. In fact this proof holds for any language L that is PSPACE-robust and has a multiple prover interactive proof system where the provers only answer questions about membership in L.

LEMMA 6.5. *Let g_n be the multilinear extension of an EXP-complete language L over the field of p_n elements where p_n is the least prime greater than n. The function $g = \{g_n : n \geq 0\}$ has a self-testing/correcting pair over the set $I = \{I_n | I_n = \mathsf{F}_{p_n}^n\}$.*

PROOF. Since g is EXP-hard and each bit of $g_n(x_1, \ldots, x_n)$ is computable in EXP (Lemma 6.2), by Corollary 4.10 there exists a multiple prover interactive proof system for verifying a specific bit of $g_n(x_1, \ldots, x_n)$ where the provers need only answer questions about g.

Let $\mathcal{P}$ be a program that purports to compute g. The tester program $T(\mathcal{P}, 1^n)$ will choose n^3 randomly chosen $(y_1, \ldots, y_n) \in \mathsf{F}_{p_n}^n$. T will then verify that each bit of the $\mathcal{P}(y_1, \ldots, y_n)$ is the same as $g(y_1, \ldots, y_n)$ with a multiprover interactive proof system using $\mathcal{P}$ as the provers. The tester T will output "Pass" if every bit checks correctly. If T outputs "Pass" with probability at least $1/3$ then with extremely high confidence $\mathcal{P}(y_1, \ldots, y_n) = g_n(y_1, \ldots, y_n)$ on all but $1/n^2$ of the possible $(y_1, \ldots, y_n) \in \mathsf{F}_{p_n}^n$.

We now use ideas of Beaver, Feigenbaum [10] and Lipton [28] to create the correcting function C. Suppose we wish to compute $g_n(x_1, \ldots, x_n)$. Choose elements $r_1, \ldots, r_n \in \mathsf{F}_{p_n}^n$ at random and let $r^i = (x_1 + ir_1, \ldots, x_n + ir_n)$ for $1 \leq i \leq n+1$. Let $g'(y) = g_n(x_1 + yr_1, \ldots, x_n + yr_n)$ for all y. With probability greater than $1 - \frac{p_n}{n^2}$ (By "Bertrand's Postulate", $p_n < 2n$), $\mathcal{P}(r^i) = g_n(r^i) = g'(i)$ since each r^i is uniformly random. However, $g'(y)$ is a polynomial of degree at most n and we have $n+1$ points of this polynomial, $g'(1), \ldots, g'(n+1)$. Interpolate this polynomial and compute $g'(0) = g_n(x_1, \ldots, x_n)$. If we repeat this process n times then with extremely high probability a majority of the answers from this process will be the proper value of g_n. □

PROOF OF THEOREM 6.4. Suppose we had a program $\mathcal{Q}$ that purports to compute L. By Lemma 6.2 there exists a polynomial time function $f(y_1, \ldots, y_n, i)$ such that the ith bit of $g_n(y_1, \ldots, y_n)$ is one if and only if $f(y_1, \ldots, y_n, i) \in L$. We create a new program $\mathcal{P}$ that simulates this process asking questions to $\mathcal{Q}$ instead of L. If $\mathcal{Q}$ properly computes L then $\mathcal{P}$ properly computes g_n.

Let T_g and C_g be the testing/self-correcting pair for g. We will create a testing/self-correcting pair T_L, C_L for L. The tester $T_L(\mathcal{Q}, 1^n)$ will just simulate $T_g(\mathcal{P}, 1^n)$ using the $\mathcal{P}$ described above. The checker $C_L(\mathcal{Q}, x)$ will just output $C_g(\mathcal{P}, (x_1, \ldots, x_n))$ where $x = x_1 \ldots x_n$. □

6.4. Comparison with the Blum–Luby–Rubinfeld Model. Blum, Luby and Rubinfeld [13] give the following series of definitions for self-testing/correcting pairs:

Let $\mathcal{D} = \{\mathcal{D}_n | n \geq 0\}$ be an ensemble of probability distributions such that $\mathcal{D}_n$ is a distribution on I_n. Let $\mathcal{P}$ be a program that purports to compute g. Let $\text{error}(g, \mathcal{P}, \mathcal{D}_n)$ be the probability that $\mathcal{P}(x) \neq g(x)$ when x is chosen from $\mathcal{D}_n$.

Let $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. An (ϵ_1, ϵ_2)-*self-testing* program for g with respect to $\mathcal{D}$ is a probabilistic polynomial-time program T such that

1. If $\text{error}(g, \mathcal{P}, \mathcal{D}_n) \leq \epsilon_1$ then $T(\mathcal{P}, 1^n)$ outputs "Pass" with probability at least 2/3.

2. If $\text{error}(g, \mathcal{P}, \mathcal{D}_n) \geq \epsilon_2$ then $T(\mathcal{P}, 1^n)$ outputs "Pass" with probability at most 1/3.

Let $0 \leq \epsilon < 1$. An ϵ-*self-correcting* program for f with respect to $\mathcal{D}$ is a probabilistic polynomial-time program C such that for all $x \in I_n$, if $\text{error}(g, \mathcal{P}, \mathcal{D}_n) \leq \epsilon$ then $C(\mathcal{P}, x) = g(x)$ with probability at least 2/3.

A *self-testing/correcting* pair for g over an input set I is a pair of programs (T, C) such that for some $\epsilon, \epsilon_1, \epsilon_2$ with $0 \leq \epsilon_1 < \epsilon_2 \leq \epsilon < 1$ and some ensemble of probability distributions $\mathcal{D}$ over I such that T is a (ϵ_1, ϵ_2)-self-testing program for g with respect to $\mathcal{D}$ and C is an ϵ-self-corretion for g with respect to $\mathcal{D}$.

Note that if g has a self-testing/correcting pair (T, C) over an input set I in the Blum–Luby–Rubinfeld model then g has a self-testing/correcting pair in our model using the same T and C.

LEMMA 6.6. *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 in the Blum–Luby–Rubinfeld model.*

PROOF. Use the function g defined in Lemma 6.5. The same tester and corrector T and C used in the proof of Lemma 6.5 also work here. Let $\mathcal{D}_n$ be the uniform distribution over $\mathbb{F}_{p_n}^n$. The tester T is a $(0, 1 - 1/n^2)$-self-testing program for g over $\mathcal{D}$. The corrector C is a $1 - 1/n^2$-self-correcting program for g over $\mathcal{D}$. □

COROLLARY 6.7. *There exist PSPACE-complete and EXP-complete functions that have self-testing/correcting pairs in the Blum–Luby–Rubinfeld model.*

It's not clear whether all PSPACE-complete or EXP-complete languages have self-testing/correcting pairs under the Blum–Luby–Rubinfeld model.

We can also do program verification in the spirit of Lipton (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 using Theorem 4.6 such that $\mathcal{P} = f$ on most inputs and then if $\mathcal{P} = f$ on most inputs we can create a correcting function C such that $C = f$ on all inputs with high probability. The proof is virtually identical to the proof of Lemma 6.5. We can also replace "multilinear" by "small-degree polynomial" as defined in Remark 4.9.

6.5. Circuit Reductions: Uniform vs. Nonuniform Complexity. Karp and Lipton [26] have considered the effect of nonuniform simulation of large complexity classes by small circuits on uniform complexity classes. They credit A. Meyer for one of following results ($\mathcal{C} = EXP$):

THEOREM 6.8. (MEYER, KARP, LIPTON) *Let $\mathcal{C}$ be one of the following complexity classes: EXP, $PSPACE$, $P^{\#P}$. If $\mathcal{C}$ has polynomial size circuits (i.e., $\mathcal{C} \subset P/poly$) then $\mathcal{C} = \Sigma_2^P$.*

Recent results on the power of interactive proofs (including our main result) lead to a strengthening of the conclusion in each case, replacing Σ_2^P by its subclass MA. For $\mathcal{C} = P^{\#P}$, this is a result of LFKN [29].

The following result generalizes a corollary in Lund, Fortnow, Karloff, Nisan [29]. For the definition of the complexity of provers see Section 4.6.

COROLLARY 6.9. *If a language L has a multiple-prover interactive proof system with provers of complexity $\mathcal{C}$ (see Section 4.6) and if $\mathcal{C}$ has polynomial-size circuits then $L \in MA$.*

Here MA denotes the Merlin–Arthur class: Non-deterministic move first, followed by a random move. Arguably this represents the class of "publishable proofs" (not requiring direct interaction between prover and verifier). Babai [4] has shown $MA \subseteq \Sigma_2^P \cap \Pi_2^P$. We note that Santha [33] constructed an oracle under which MA is properly contained in AM, itself still a subclass of Π_2^P.

PROOF. Merlin produces the circuits for $L_1, \ldots, L_k$ that describe the responses for provers $P_1, \ldots, P_k$ respectively and Arthur simulates the verifier for L using the circuits to compute the provers' responses. □

In particular, if L has a function-restricted interactive proof and L has polynomial-size circuits then $L \in MA$.

Corollary 6.10. *If all of EXP has polynomial-size circuits then $EXP = \Sigma_2^P = \Pi_2^P = MA$. The same statement holds for $PSPACE$ and $P^{\#P}$ in the place of EXP.*

It seems remarkable that this result, which refers to standard concepts of structural complexity theory, has been proved via the theory of multi-prover interactive proof systems.

7. Concluding Remarks

7.1. Integers vs. Finite Fields. We have formulated our protocols over the integers. Adapting them to finite fields requires some extra thought. The main advantage of such an adaptation is that we should not need to compute with large numbers.

The multilinearity test works over any field. However, the "sum of squares" trick employed in Lemma 4.3 requires real numbers. Below we indicate how to eliminate this difficulty.

First we state a version of Lemma 4.3 which over fields (or integral domains with identity) of any characteristic.

Lemma 7.1. *Let p be a given prime number or zero, and $\mathbb{F} = \mathbb{Z}/(p)$ (either the field of order p or $\mathbb{Z}$). Given an instance (B, r, s) of oracle-3-satisfiability (where $B = B(w,t)$ is a Boolean formula in $r + 3s + 3$ variables), one can compute in polynomial time an arithmetic expression for a polynomial f with coefficients in $\mathbb{F}$ over the same set of $r + 3s + 3$ variable symbols such that a function $A : \{0,1\}^s \to \mathbb{F}$ constitutes a 3-satisfying oracle for B if and only if*

$$(\forall w \in \{0,1\}^{r+3s}) \quad f(w, A(b_1), A(b_2), A(b_3)) = 0. \qquad (7.17)$$

$$(\forall b \in \{0,1\}^s) \qquad A(b)(A(b) - 1)) = 0. \qquad (7.18)$$

Proof. As in the proof of Lemma 4.3, we take f to be the arithmetic expression for a polynomial representing B (Proposition 3.1). The rest of the proof follows the lines of the proof of Lemma 4.3. □

The question now is, how the Prover convinces the Verifier that each of these exponentially many quantities vanishes. Let g be a low degree polynomial of m variables over $\mathbb{F}$ for which we want to verify that g is identically zero over the Boolean cube $\{0,1\}^m$. We assume that values of g over some domain $\mathbb{I}$ of appropriately large size are held by an oracle. (If $\mathbb{F}$ is too small, we have to use a subset $\mathbb{I}$ of some extension field of $\mathbb{F}$.) We describe two verification schemes.

For the first scheme, we have to assume that F has characteristic 2. Let us consider the sum $\sum_{w \in U} g(w)$ where U is a subset chosen at random from certain family of subsets. For instance, viewing $\{0,1\}^m$ as a linear space over F_2, the Verifier can choose U to be a random subspace of dimension d where $0 \leq d \leq m$ is selected at random. This way if g is not identically zero then the above sum will have at least $1/(4(m+1))$ chance of being nonzero according to a lemma of Rabin (see [42]). Now the characteristic function of U can be expressed as a polynomial of degree $\leq m$ and the protocol of Lemma 3.5 can be used. – Variations of this method use classes of hash-functions to specify the subset U.

Below we describe a different procedure with a self-contained proof.

We allow F to be an arbitrary finite field. Let us extend F to a field F' of order greater than 2^{m+1}. (Elements of F' can be represented as tuples of elements of F.) Now consider the univariate polynomial $p(x) = \sum_{w \in \{0,1\}^m} g(w) x^w$ where the binary string $w = \omega_0 \dots \omega_{m-1}$ written in the exponent refers to the integer $\sum_{i=0}^{m-1} \omega_i 2^i$. Then the probability that a random $\xi \in \mathsf{F}'$ is a root of p is 1 if g is identically zero and $\leq 1/2$ otherwise. Therefore the task of the Prover is to convince the Verifier that for a random ξ provided by the Verifier, $p(\xi) = 0$.

Let now $\xi_i = \xi^{2^i}$; then

$$\xi^w = \prod_{i=0}^{m-1} \xi_i^{\omega_i} = \prod_{i=0}^{m-1} (1 + (\xi_i - 1)\omega_i). \tag{7.19}$$

The Verifier, having computed the ξ_i, holds the explicit multilinear polynomial of w on the right hand side of (7.19). Therefore the protocol of Lemma 3.5 can be used to verify the equality $p(\xi) = 0$, assuming g is a good approximation of a low degree polynomial (which in our case is guaranteed by the multilinearity test for A).

7.2. Recent Developments. The *MIP* protocol described in this paper has recently found curious applications and extensions.

A *clique approximation* algorithm is an algorithm that computes the size of maximum cliques in a graph within a constant factor. Feige, Goldwasser, Lovász, and Safra [17] made the striking observation that our Main Theorem has the following fairly immediate consequence: *If there exists a polynomial-time clique approximation algorithm then $EXP = NEXP$.* They also proved that a slight modification yields (under the same assumption) the stronger consequence that NP is in *quasi-polynomial* time, where quasi-polynomial means $\exp((\log n)^{O(1)})$.

Digging considerably deeper into our protocols, Szegedy [38] achieved remarkable improvements, yielding in particular that *a polynomial time clique approximation algorithm implies that NP is in $DTIME(n^{O(\log\log n)})$*. This means *nearly polynomial* time, where "nearly polynomial" is defined as $n^{(\log\log n)^{O(1)}}$. The ultimate goal would be to infer $NP = P$ from the same hypothesis.

In another direction, in joint work with L. Levin and M. Szegedy, we have further explored the implications of our protocols to the verification of program instances and mathematical proofs. In particular, we have introduced a concept of *transparent proofs* [7]. Roughly speaking, a pair (T, P) of strings, where T is a "theorem–candidate" and P is a "proof–candidate", is in transparent form, if T is encoded in an error-correcting code, and the pair (T, P) can be verified by a probabilistic verifier in polylog(N)-time, where N is the combined length of (T, P), and the verifier has random access to the string (T, P). (The string T must be encoded because the verifier does not have enough time to read T so it could not observe slight changes in the statement of the theorem.) Improving several aspects of the protocols of this paper, we are able to prove that *every (deterministic) mathematical proof can be transformed in polynomial time into a transparent proof.* In particular, programs with (nondeterministic) polynomial time specifications can be viewed as provers of theorems (such as "the product of the matrices A and B is C; or "the graph X is Hamiltonian"). Our result says that, if the untrusted prover invests a polynomial amount of extra work, the result can be checked in polylogarithmic time.

7.3. Open Problems. Many open questions remain about multi-prover interactive proof systems including:

- Does all of $NEXP$ have bounded-round two-prover interactive proof systems? Note that this strengthening of Cai's result would not necessarily imply the collapse of the polynomial-time hierarchy. We remark that if we allow a polynomial number of provers then a bounded number of rounds does suffice [21].

- What complexity of provers do we need to prove $coNP$ and $NEXP$ languages (see Section 4.6)?

- Finally, there seems occasion to cautiously express hope that the techniques discussed above might lead to a solution of some long standing separation problems such as BPP vs. $NEXP$ (cf. [24], [25]). Although there exist oracles which collapse these classes, this fact no longer seems as discouraging as it used to be, in view of a substantial mass of new *techniques that do not relativize.*

Acknowledgments

The first author was partially supported by NSF Grant CCR-8710078. The second author is partially supported by NSF Grant CCR-9009936. - The authors are grateful to Claus Schnorr for his insistent criticism which was helpful in improving the presentation of the material. Discussions with Leonid Levin and Mario Szegedy were illuminating.

References

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading MA, 1974.

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

[3] L. BABAI, Trading group theory for randomness, in *Proc. 17th Ann. ACM Symp. Theory of Computing*, 1985, 421-429.

[4] L. BABAI, E-mail and the unexpected power of interaction, in: Proc. 5th Ann. IEEE Structures in Complexity Theory Conf., 1990, 30-44.

[5] L. BABAI AND P. ERDŐS, Representation of group elements as short products, *Annals of Discrete Mathematics* **12** (1982), 27-30.

[6] L. BABAI AND L. FORTNOW, Arithmetization: a new method in structural complexity theory, *Computational Complexity* **1** (1991), 41-66. (Preliminary version appeared as: A characterization of $\#P$ by arithmetic straight line programs, in *Proc. 31st Ann. IEEE Symp. Foundations of Comp. Sci.*, 1990, 26-34.)

[7] L. BABAI, L. FORTNOW, L. LEVIN, AND M. SZEGEDY, Checking computations in polylogarithmic time, in: *Proc. 23rd ACM Symp. Theory of Computing*, 1991, to appear.

[8] L. BABAI, L. FORTNOW, AND C. LUND, Non-deterministic exponential time has two-prover interactive protocols (extended abstract), *Proc. 31st Ann. IEEE Symp. Found. Comp. Sci.*, 1990, 16-25.

[9] L. BABAI AND P. FRANKL, *Linear Algebra Methods in Combinatorics, I*, Preliminary Version, University of Chicago, Dept. C. S. 1988.

[10] D. Beaver and J. Feigenbaum, Hiding instances in multioracle queries, in *Proc. 7th Symp. on Theoretical Aspects of Comp. Sci.*, *Lecture Notes in Comp. Sci.* **415** (1990), 37-48.

[11] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, Multi-prover interactive proofs: How to remove the intractability assumptions, in *Proc. 20th Ann. ACM Symp. Theory of Computing*, 1988, 113-131.

[12] M. Blum and S. Kannan, Designing programs that check their work, in *Proc. 21st Ann. ACM Symp. Theory of Computing*, 1989, 86-97.

[13] M. Blum, M. Luby, and R. Rubinfeld, Self-testing and self-correcting programs, with applications to numerical programs, in *Proc. 22nd Ann. ACM Symp. Theory of Computing*, 1990, 73-83.

[14] L. Babai and S. Moran, Arthur–Merlin games: a randomized proof system, and a hierarchy of complexity classes, *J. Comp. Sys. Sci.* **36** (1988), 254-276.

[15] J. Cai, PSPACE is provable by two provers in one round, manuscript, 1990.

[16] S. A. Cook, The complexity of theorem proving procedures, in *Proc. 3rd Ann. ACM Symp. Theory of Computing*, 1971, 151–158.

[17] U. Feige, S. Goldwasser, L. Lovász, and S. Safra, On the complexity of clique approximation, in preparation.

[18] P. Feldman, The Optimum Prover lives in *PSPACE*, manuscript, 1986.

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

[20] L. Fortnow, 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.

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

[22] L. Fortnow and M. Sipser, Are there interactive protocols for co-NP languages?, *Inf. Process. Letters* **28** (1988), 249-251.

[23] S. Goldwasser, S. Micali, and C. Rackoff, The knowledge complexity of interactive proofs, *SIAM J. Comput.* **18** (1989), 186-208. (Preliminary version appeared in *Proc. 18th Ann. ACM Symp. Theory of Computing*, 1985, 291-304.)

[24] J. Hartmanis, N. Immerman, and V. Sewelson, Sparse sets in $NP - P$: $EXPTIME$ versus $NEXPTIME$, *Inf. and Control* **65** (1985), 158-181.

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

[26] R. Karp, R. Lipton, Some Connections between Nonuniform and Uniform Complexity Classes, *Proc. 12th Ann. ACM Symp. Theory of Computing*, 1980, 302-309.

[27] L. Levin, Universal'nyĭe perebornyĭe zadachi (Universal search problems, in Russian), *Problemy Peredachi Informatsii* **9** (1973), 265-266. A corrected English translation appears in an appendix to Trakhtenbrot [39].

[28] R. Lipton, New directions in testing, in *Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography*, 1989, to appear.

[29] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, Algebraic methods for interactive proof systems, in *Proc. 31st Ann. IEEE Symp. Foundations of Comp. Sci.*, 1990, 1-10.

[30] P. Orponen, Complexity Classes of Alternating Machines with Oracles, *Proc. 10th ICALP*, *Lecture Notes in Comp. Sci* **154** (1983), 573-584.

[31] C. Papadimitriou, Games against Nature, *Proc. 24th Ann. IEEE Symp. Foundations of Comp. Sci.*, 1983, 446-450.

[32] G. Peterson and J. Reif, Multiple-person alternation, *Proc. 20th Ann. IEEE Symp. Foundations of Comp. Sci.*, 1979, 348-363.

[33] M. Santha, Relativized Arthur–Merlin versus Merlin–Arthur games, *Inf. and Computation* **80** (1989), 44–49.

[34] J. Seiferas, M. Fischer, and A. Meyer, Separating Nondeterministic Time Complexity Classes, *J. Assoc. Comput. Mach.* **25** (1978), 146-167.

[35] A. Shamir, IP = PSPACE, in *Proc. 31st Ann. IEEE Symp. Foundations of Comp. Sci.*, 1990, 11-15.

[36] J. SIMON, On Some Central Problems in Computational Complexity, Ph.D. Thesis, *Cornell University, Computer Science, Tech. Report* TR 75-224, 1975.

[37] J. T. SCHWARTZ, Fast probabilistic algorithms for verification of polynomial identities, *J. Assoc. Comput. Mach.* **27** (1980), 701-717.

[38] M. SZEGEDY, Efficient *MIP* protocol and a stronger condition on clique approximation, in preparation.

[39] B. A. TRAKHTENBROT, A survey of Russian approaches to *Perebor* (brute-force search) algorithms, *Annals of the History of Computing* **6** (1984), 384-400.

[40] S. TODA, On the computational power of *PP* and $\oplus P$, in *Proc. 30th Ann. IEEE Symp. Foundations of Comp. Sci.*, 1989, 514–519.

[41] L. VALIANT, The complexity of computing the permanent, *Theoretical Computer Science* **8** (1979), 189-201.

[42] L. VALIANT, V. VAZIRANI, *NP* is as Easy as Detecting Unique Solutions, *Theoretical Computer Science* **47** (1986), 85-93.

Manuscript received 30 March 1990

LÁSZLÓ BABAI
University of Chicago
Chicago, IL 60637
and
Eötvös University, Budapest, Hungary
`laci@cs.uchicago.edu`

LANCE FORTNOW
Department of Computer Science
University of Chicago
1100 E. 58th St.
Chicago, IL 60637
`fortnow@cs.uchicago.edu`

CARSTEN LUND
Department of Computer Science
University of Chicago
1100 E. 58th St.
Chicago, IL 60637

Current Address of CARSTEN LUND:
DIMACS
P.O. Box 1179
Rutgers University
Piscataway, NJ 08855-1179
`clund@dimacs.rutgers.edu`