

Checking Computations in Polylogarithmic Time

*László Babai*¹
Univ. of Chicago⁶ and
Eötvös Univ., Budapest

*Lance Fortnow*²
Dept. Comp. Sci.
Univ. of Chicago⁶

*Leonid A. Levin*³
Dept. Comp. Sci.
Boston University⁴

*Mario Szegedy*⁵
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum’s concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x , representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S .

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log \log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired*. (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the “theorem–candidate” is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log \|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\leq N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

¹Research partially supported by NSF Grant CCR-8710078. E-mail: laci@cs.uchicago.edu

²Research partially supported by NSF Grant CCR-9009936. E-mail: fortnow@cs.uchicago.edu

³Supported by NSF grant CCR-9015276. E-mail: Lnd@cs.bu.edu

⁴111 Cummington St., Boston MA 02215.

⁵Current address: AT&T Bell Laboratories, 600 Mountain Avenue, P.O. Box 636, Murray Hill, NJ 07974-0636.
E-mail: ms@research.att.com

⁶1100 E 58th St, Chicago IL 60637.

1 Introduction

1.1 Very long mathematical proofs

An interesting foundational problem is posed by some mathematical proofs which are too large to be checked by a single human.

The proof of the Four Color Theorem [AHK77], considered controversial at the time, started with a Lemma that the Theorem follows if certain computation terminates. It was completed with the experimental fact that the computation did indeed terminate within two weeks on contemporary computers.

The “Enormous Theorem” [Gor85] provides the classification of all finite simple groups. Its proof, spread over 15,000 pages in Gorenstein’s estimate, consists of a large number of difficult lemmas. Each lemma was proven by a member of a large team, but it seems doubtful that any one person was able to check all parts.

An even more difficult example is a statement that a given large tape contains the correct output of a huge computation (with program and input on the same tape). One might attempt to verify the claim by repeating the computation, but what if there is a systematic error in the implementation?

The first two examples are special cases of the last one. The requirements for mathematical proofs can be completely formalized. The statement of the theorem would have to incorporate the definitions, basic concepts, notation, and assumptions of the given mathematical area. They should also include the general axiom schemes of mathematics used (say, Set Theory), furthermore, the logical axioms and inference rules, parsing procedures needed to implement the logic, etc. The theorem will then become very large and ugly, but still easily manageable by computers. The computation would then just check that the proof adheres to the specifications.

1.2 Transparent proofs

Randomness offers a surprisingly efficient way out of the foundational problem. As we shall see, all formal proofs can be transformed into proofs that are checkable in *polylogarithmic* Monte Carlo time. Note that no matter how huge a proof we have, the logarithm of its size is very small: The logarithm of the number of particles in the visible Universe is under 300.

A *probabilistic proof system* is a $\|T, P\|^{O(1)}$ time algorithm $A(T, P, \omega)$. It has random access to the source ω of coin flips and two input arrays: T (“theorem candidate”) and P (“proof candidate”). T is given in an error correcting code: If T is not a valid code word, but is within, say, 10% Hamming distance of one, this valid codeword T' is uniquely defined and recoverable in nearly linear time. Each pair (T, P) is either

1. *correct*: accepted for all ω , or
2. *wrong*: rejected for most ω , or
3. *imperfect*: can be easily transformed into correct (T', P') , with unique T' close to T .

In the last case the checker is free either to reject (there are errors) or to accept (errors are inessential). Using a special error correcting code, we can guarantee the acceptance with high probability if there are $< 10\%$ of errors and make “easily transformed” to mean polylogarithmic time per digit.

P is a *proof* of T if (T, P) is correct. T is a *theorem* if it has a proof. A *deterministic* proof system is the special case with no use of ω . *Extension* is a system with more proofs but not more theorems.

A proof P of T is *transparent* if it is accepted in a (specific) poly-logarithmic time $(\log \|T, P\|)^{O(1)}$. The system is *friendly* if every proof P can be transformed in $\|T, P\|^{O(1)}$ time into a transparent proof of the same theorem.

Theorem 1.1 *Each deterministic proof system has a friendly probabilistic extension.*

It will be sufficient to consider just one proof system which is *NP*-complete with respect to polylogarithmic time reductions (see definition in Section 6.3). In fact, using a RAM model rather than Turing machines, we construct a proof system, complete for nearly-linear non-deterministic time with respect to

such reductions. This enables us, in time $\|T, P\|^{1+\varepsilon}$, for any $\varepsilon > 0$, to put the proofs into transparent form, verifiable in time $(\log \|T, P\|)^{O(1/\varepsilon)}$. It is an interesting problem to eliminate $1/\varepsilon$ from this exponent.

1.3 Comments

Polynomial and *polylog* above refer to $\|T, P\|^{O(1)}$ and $(\log \|T, P\|)^{O(1)}$, resp., where $\|x\|$ denotes the length of the string x . *Nearly linear* means linear times polylog.

Theorem 1.1 asserts that given T (as a valid code-word) and a correct proof P , one can compute in polynomial time another proof \tilde{P} of T which is accepted in polylog time (by the extended system).

Note that acceptance of (T, P) does not guarantee the correctness of (T, P) . But, if acceptance occurs with a noticeable probability, then both T and P can easily be corrected.

Correcting the Theorem. Error-correcting format (encoding) of the input refers to any specific polynomial-time computable encoding with the property that every codeword can be recovered in polynomial time from any distortion in a small constant fraction of the digits. Such codes can be computed very efficiently by log-depth, nearly linear size networks (e.g. variants of FFT).

The error-correcting encoding of the theorem-candidate is crucial; otherwise P could be a correct proof of a slightly modified (and therefore irrelevant) theorem, and it would not be possible to detect in polylog time the slight alteration. In case T fails to be in valid error-correcting form, acceptance of the proof means correctness of the unique T' to which T is close.

One would not need error-correcting encoding if we were only concerned about short theorems with long proofs. This is rare in computing, where inputs/outputs tend to be long. But even in mathematics there are good reasons to assume that truly self-contained theorem statements will be very long. Indeed, as discussed in Section 1.1, the statement of a theorem in topology, for instance, would include lots of relevant textbook material on logic, algebra, analysis, geometry, topology.

Correcting the Proof. If (T, P) has a noticeable chance of acceptance in the extended system then T (after error-correction, if not a valid code-word) is guaranteed to be a theorem, i.e. to have a correct proof. We do not guarantee that P itself is a correct proof, but a simple Monte-Carlo procedure with random access to P will correct it, revising each digit independently in polylog time. (This is a self-correction feature related to a self-correction concept introduced by Blum–Luby–Rubinfeld and Lipton. The proof uses the interpolation trick of Beaver–Feigenbaum–Lipton.)

2 Checking computations

We shall now interpret Theorem 1.1 in the context of checking of computations. There will be two parties involved: the Solver, competing in the open market to serve the user; and the Checker, a tiny but highly reliable device.

2.1 A universal Checker

A non-deterministic programming task is specified by a deterministic polynomial-time predicate $S(x, y, W)$ meaning W is a *witness* that (x, y) is an acceptable input–output pair in error-correcting form.

A *Solver* may present a program which is claimed to produce a (possibly long) witness W , and running S on a reliable machine with reliable software might be expensive and time-consuming. Instead, our result allows to modify the specification such that (a) the same input–output pairs will be accepted; (b) the same Solver can solve the modified task at only a small extra cost to him; (c) the result can be checked in polylogarithmic time. – We now rephrase Theorem 1.1 in the checking context.

The following corollary assumes t to be an upper bound on $\|x, y, W\|$ and on the running time of $S(x, y, W)$; ω is a random sequence of coin flips. We select a value $\varepsilon > 0$.

Corollary 2.1 *There exist machines E (Editor, $t^{1+\varepsilon}$ running time) and C (Checker, $(\log t)^{O(1/\varepsilon)}$ running time) such that for any S, x, y with error-correcting codes $\bar{S}, \bar{x}, \bar{y}$ and W :*

- *if $S(x, y, W)$ accepts then $C(\bar{S}, \bar{x}, \bar{y}, E(W), \omega)$ accepts for all ω .*
- *If $C(S, x, y, W, \omega)$ accepts for $> 1\%$ of all ω then S, x, y are within 1% Hamming distance from error-correcting codes of unique and easily recoverable S', x', y' . Moreover, $S'(x', y', W)$ accepts for some W .*

Only the Checker but neither the software nor the hardware used by the Solver/Editor need to be reliable. If reliability is regarded as a substantial cost factor, the stated result demonstrates that this *cost can be reduced* dramatically by requiring the *unreliable elements to work just a little harder*.

The relation to Theorem 1.1 is that $E(W)$ will be the “transparent proof” of the statement $(\exists W)S(x, y, W)$.

2.2 Space saving, parallelization

$E(W)$ from Corollary 2.1 could be long. While $\|E(W)\| \leq \|W\|^{1+\varepsilon}$, this ε should not be taken too small: that raises the exponent in the polylogarithmic time bound of the Checker. But we do not need the Solver to write down the entire $E(W)$. Instead, he can provide a devilish program, which computes the i^{th} bit of z , from i . This could save considerable space, and as a result, possibly even time.

However, such a program could then cheat by being adaptive, i.e. making its responses depend on previous questions of the Checker. We can eliminate this possibility by implementing the *multi-prover* model: run a replica of the program separately; after the Checker had asked all its questions from the original, select one of these questions at random and ask the replica. If the answer differs, reject. Repeat this process a polylogarithmic number of times. If no contradiction arises, accept (cf. [BGKW88], [FRS88], [BFL91]).

We can give a helping hand to the Solver to enable his program to respond in something like real time to the Checker’s questions, after some preprocessing.

Proposition 2.2 *Using the notation of Corollary 2.1, there exists an NC^2 algorithm to compute any entry of $E(W)$ from x, y, W , after a nearly linear (in $\|x, y\|$) preprocessing time.*

2.3 Comparison with Blum’s models

Our result says that any computation is only “ $N^{1+\varepsilon}$ time away” from a computation checkable in polylogarithmic time in a sense related to Blum’s.

Blum and Kannan [BK89] 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 , outputs with high probability:

1. “correct,” if \mathcal{P} correctly computes L for all inputs.
2. “ \mathcal{P} does not compute L ,” if $\mathcal{P}(x) \neq L(x)$.

In recent consecutive extensions of the power of interactive proofs, complete languages in the corresponding classes were shown to admit Blum–Kannan instance checkers: $P^{\#P}$ [LFKN92], $PSPACE$ [Sha92], EXP [BFL91]. (See [BF91] for more such classes.)

The Checker of the present paper runs in polylogarithmic time and needs no interaction with the program to be checked. There is some conceptual price to pay for these advantages.

What we check is the computation as specified by the User, rather than the language or function to be computed. We do allow the Solver to use a devilish program, though; conceivably such a program may be helpful not only in establishing the right output but also to compute the requested entries of $E(W)$ without writing all of it down. One objection might be that this excludes some more efficient algorithms (e.g. if we specify compositeness-testing by way of computing a divisor as witness, this may make the Solver’s task exceedingly hard).

For a comparison with the Blum–Kannan definition consider a program \mathcal{P} which is claimed to compute the entries of $E(W)$. Assuming now that x and S are given in error correcting encoding, our Checker $C^{\mathcal{P}}$

- outputs “correct,” if \mathcal{P} correctly computes all entries of $E(W)$.
- with high probability outputs “ \mathcal{P} does not compute a correct $E(W)$,” unless (except for a small fraction of easily and uniquely recoverable errors) x, S are in error-correcting form and x is acceptable (i.e. $(\exists W)S(x, W)$).

2.4 Applications to Probabilistically Checkable Programs

Arora and Safra [AS92] define a hierarchy of complexity classes PCP (for probabilistically checkable proofs), corresponding to the number of random and query bits required to verify a proof of membership in the language, as follows:

A *verifier* M is a probabilistic polynomial-time Turing machine with random access to a string Π representing a membership proof; M can *query* any bit of Π . Call M an $(r(n), q(n))$ -restricted verifier if, on an input of size n , it is allowed to use at most $r(n)$ random bits for its computation, and query at most $q(n)$ bits of the proof.

A language L is in $PCP(r(n), q(n))$ if there exists an $(r(n), q(n))$ -restricted verifier M such that for every input x :

1. If $x \in L$, there is a proof Π_x which causes M to accept for every random string, i.e., with probability 1.
2. If $x \notin L$, then for all proofs Π , the probability over random strings of length $r(n)$ that M using proof Π accepts is bounded by $1/2$.

Fortnow, Rompel and Sipser [FRS88] show that the languages accepted by multiple provers and $\cup_{k>0} PCP(n^k, n^k)$ are equivalent. Thus Babai, Fortnow and Lund [BFL91] show that $NEXP = \cup_{k>0} PCP(n^k, n^k)$.

The results in this paper show that $NP \subseteq \cup_{c>0} PCP(c \log n, \log^c n)$ since a careful analysis shows that we only need $O(\log n)$ coins in our protocol. In fact we get the stronger result that the proof has size $n^{1+\epsilon}$.

Arora and Safra [AS92] and Arora, Lund, Motwani, Sudan and Szegedy [ALM⁺92] build on this result to show that $NP = \cup_{c>0} PCP(c \log(n), c)$.

3 A Transparent Proof Based on MIP

In this section we will informally show how to convert results on multiple prover interactive proof systems to weak results on transparent proofs.

Instead of the standard definition of multiple prover interactive proof systems, we will use the following equivalent definition due to Fortnow, Rompel and Sipser [FRS88]:

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 there exists an oracle \mathcal{O} such that

1. For every $x \in L$, $M^{\mathcal{O}}$ 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.

Babai, Fortnow and Lund [BFL91] show that every language in nondeterministic exponential time is accepted by a probabilistic oracle machine. They also show that every language in deterministic exponential time has such a machine where the oracle \mathcal{O} can be computed in deterministic exponential time. Note that for an input of length n , only oracle questions of size at most n^k can be queried.

To create a transparent proof we just scale down this protocol. Let $N = 2^n$. Suppose we want to verify a computation running in time polynomial in N . This is an exponential-time computation in n . The Babai-Fortnow-Lund theorem we now have an oracle of size 2^{n^k} with the following properties:

1. The oracle can be computed in time $2^{n^k} = N^{(\log \log N)^{k-1}}$.
2. The computation can be verified probabilistically in time $n^k = (\log N)^k$ and thus look in only $(\log n)^k$ places in the oracle.

Thus we have a proof of only slightly more than polynomial size that can be verified probabilistically in polylog time.

However we can not trust that the oracle is actually a computation of the input that we are interested in. If we only check a polylogarithmic number of locations in the oracle, then a dishonest oracle could give a proof for a computation on an input that is one bit away from the real input and we would never catch it.

To handle this problem we require that the input is in an error-correcting code format. An error-correcting code format increases an input by a small amount and has the important property that the error-correcting code for every two inputs will differ in some constant fraction of their bits. Thus we can then check with only a constant number of queries that the input we have matches the one used by the computation in the oracle.

4 Low degree polynomials

This section presents technical preliminaries, of which an algorithmic result on codes might be of independent interest (Proposition 4.6).

4.1 Low degree extension

The “proof” in our proof system consists of an admissible coloring A of the graph G_n . We may assume A is a string of “colors” of length 2^n . We will explain the structure of these graphs in Section 6. The set C of colors will be viewed as a subset of a field \mathbf{F} . The transparent version will involve an extension of this string to a table of values of a multivariate polynomial over \mathbf{F} .

[BFL91] suggests to identify the domain V of A with $\{0, 1\}^n$, and, regarding $\{0, 1\}$ as a subset of \mathbf{F} , extend A to a multilinear function over \mathbf{I}^n where \mathbf{I} is a finite subset of \mathbf{F} . However, in order for the [LFKN92]-type protocol of [BFL91] to work, one requires \mathbf{I} to have order $\Omega(n^2)$, forcing the table of the multilinear function to have size $n^{\Omega(n)} = N^{\Omega(\log \log N)}$, where $N = 2^n$ is the length of A . This would render the “transparent proof” slightly superpolynomial.

Instead we select a small subset $H \subset \mathbf{F}$ of size $|H| = 2^\ell$ where $\ell = \Theta(\log n/\varepsilon)$ where $\varepsilon > 0$ is the quantity appearing in the comments after the statement of Theorem 1.1 as well as in Corollary 2.1 We may assume $m := n/\ell$ is an integer; and we identify V with the set H^m . The next Proposition allows us to extend A to a low degree polynomial in only m variables. Now the size of the table of the extended function has size

$$|\mathbf{I}|^m = N(|\mathbf{I}|/|H|)^m = N(\Theta(n^2))^m = N^{1+\Theta(\varepsilon)}, \quad (1)$$

using the stipulation, to be justified later, that the right size of \mathbf{I} is $\Theta(n^2|H|)$.

Proposition 4.1 (Low degree extension) *Let $H_1, \dots, H_m \subseteq \mathbf{F}$ and let $f : H_1 \times \dots \times H_m \rightarrow \mathbf{F}$ be a function. Then there exists a unique polynomial \tilde{f} in m variables over \mathbf{F} such that*

- \tilde{f} has degree $\leq |H_i|$ in its i^{th} variable;
- \tilde{f} , restricted to $H_1 \times \dots \times H_m$, agrees with f .

Proof. Let $u = (u_1, \dots, u_m) \in K := H_1 \times \dots \times H_m$. Consider the polynomial

$$g_u(x) = \prod_{i=1}^m \prod_{h \in H_i \setminus \{u_i\}} (x_i - h). \quad (2)$$

Now $g_u(u) \neq 0$ but $g_u(x) = 0$ for all $x \in K \setminus \{u\}$. Clearly, every function $K \rightarrow \mathbf{F}$ is a linear combination of the g_u , restricted to K . This proves the existence part of the statement. The uniqueness can be proved by an easy induction on m . \square

4.2 Low degree test

One of the key ingredients of the program that checks the “transparent proof” is a test that a function $f : \mathbf{F}^m \rightarrow \mathbf{F}$ is a low degree polynomial.

We say that two functions defined over a common finite domain are α -*approximations* of one another if they agree on at least a $(1 - \alpha)$ fraction of their domain.

An important feature of low degree polynomials is that they form an *error-correcting code* with large distance: two such polynomials can agree on a small fraction of their domain only, assuming the domain is not too small. This follows from a well known lemma of J. T. Schwartz [Sch80b] which we quote.

Lemma 4.2 (J. T. Schwartz) *Let $\mathbf{I} \subset \mathbf{F}$ be a finite subset of the field \mathbf{F} . Let $f : \mathbf{F}^m \rightarrow \mathbf{F}$ be an m -variate polynomial of (combined) degree $d \geq 0$. Then f cannot vanish in more than a $d/|\mathbf{I}|$ fraction of \mathbf{I}^m .*

(The proof is by a simple induction on m .)

An important property of low degree polynomials over not too small finite fields is that they are *random self-reducible*, as observed by Beaver–Feigenbaum [BF90] and Lipton [Lip91]. They show that an m -variate polynomial $p : \mathbf{F}^m \rightarrow \mathbf{F}$ of degree d can be recovered from an α -approximation assuming $\alpha \leq 1/(2d)$ and $|\mathbf{F}| \geq d+2$. A. Wigderson has pointed out to us that the bound on α can be improved to a constant, say 15%, using known error-correction techniques in the way used by Ben-Or, Goldwasser, and Wigderson [BGW88] in their “secret sharing with cheaters” protocol. We briefly review the technique and state the result.

Proposition 4.3 *Let p be an unknown polynomial of degree d in m variables over the finite field \mathbf{F} . Let $f : \mathbf{F}^m \rightarrow \mathbf{F}$ be an α -approximation of p , where $\alpha = .15$. Let n be a divisor of $|\mathbf{F}| - 1$, and assume $n \geq 3d + 1$. Then there is a Monte Carlo algorithm which for any $x \in \mathbf{F}^m$ computes $p(x)$ with large probability in time polynomial in n, m , and $\log |\mathbf{F}|$ if allowed to query f .*

Proof. Let ω be a primitive n^{th} root of unity in \mathbf{F} . We select a random $r \in \mathbf{F}^m$, query the values $\varphi(\omega^i) = f(x + r\omega^i)$ for $i = 0, \dots, n - 1$. Let $\tilde{\varphi}$ be defined analogously, using p in place of f . Now $\tilde{\varphi}$ is a univariate polynomial of degree $\leq d \leq (n - 1)/3$, and with some luck, no more than $(n - 1)/3$ of the queried values of f differ from the corresponding values of p . If this is the case, the polynomial $\tilde{\varphi}$ can be recovered from the values of φ . Indeed, as observed in [BGW88, p. 5], this is a case of correcting errors in a generalized Reed-Muller code, cf. [PW72, p. 283].

Repeating this process we are likely to succeed and obtain the correct value of $p(x)$ for a majority of the random choices of r . \square

Let us say that a multivariate polynomial f is *h -smooth* if it has degree $\leq h$ in each variable. A function is α -*approximately h -smooth* if it is an α -approximation of an h -smooth function. 1-smooth polynomials are called multilinear. One of the key ingredients of the $MIP = NEXP$ protocol in [BFL91] is a multilinearity/low degree test.

Theorem 4.4 ([BFL91]) *Let \mathbf{F} be a field, h, m positive integers, and \mathbf{I} a finite subset of \mathbf{F} . There exists a Monte Carlo algorithm which tests approximate h -smoothness of a function $f : \mathbf{I}^m \rightarrow \mathbf{F}$ in the following sense:*

- (i) if f is h -smooth, the algorithm accepts;

(ii) if f is not α -approximately h -smooth, the algorithm rejects with high probability, where $\alpha = 4m^2h/|\mathbf{I}|$.

The algorithm queries $hm^{O(1)}$ values of f .

For the proof, see [BFL91, Theorem 5.13 and Remark 5.15]. (Specific citations refer to the journal version.) A more efficient version of the algorithm was recently found by Szegedy (see [FGL⁺91]).

We shall now combine this result with the ideas of Beaver, Feigenbaum, Lipton, Ben-Or, Goldwasser, and Wigderson to upgrade the test, incorporating a strong self-correction feature which makes the test tolerant to errors of up to a substantial fraction of the domain. In order to do so, we have to take $\mathbf{I} = \mathbf{F}$.

Corollary 4.5 *Let \mathbf{F} be a finite field, and h, m positive integers. Assume $|\mathbf{F}| \geq 28m^2h$. There exists a Monte Carlo algorithm which tests approximate h -smoothness of a function $f : \mathbf{F}^m \rightarrow \mathbf{F}$ in the following stronger sense:*

(i) if f is .15-approximately h -smooth, the algorithm accepts with high probability; and for any $x \in \mathbf{F}^m$, computes the value of the unique h -smooth approximation of f at x ;

(ii) if f is not .16-approximately h -smooth, the algorithm rejects with high probability.

The algorithm runs in time $(|\mathbf{F}|m)^{O(1)}$, including the queries to values of f .

Proof. Let us first pretend that f is .15-approximately h -smooth. Apply the procedure of Proposition 4.3 to $(hm)^{O(1)}$ points $x \in \mathbf{F}^m$ to establish values (random variables) $\tilde{f}(x)$. If the procedure fails to produce a value or the value produced is different from $f(x)$ more than 15.5% of the time, reject. Else, perform the h -smoothness test of Theorem 4.4 to the values of \tilde{f} rather than those of f .

If indeed f is .15-approximately h -smooth then let g be its unique smooth approximation. For every x it has very large $(1 - \exp(-(hm)^c))$ probability that $\tilde{f}(x) = g(x)$, so the smoothness test will accept. On the other hand, if the self-correction procedure does not reject, then it is likely that there exists a function $g(x)$ such that for almost every x , $\tilde{f}(x) = g(x)$ with large probability. Now if the smoothness test accepts, then g must very closely approximate an h -smooth function g' . Now we are almost certain that f and g agree on all but 15.5% of the places, so all put together f is very likely to .16-approximate the h -smooth g' . \square

4.3 An efficiently correctable code

We describe an error-correcting code with a polylogarithmic error correction algorithm, capable of restoring each bit from a string which may have errors in a substantial fraction. This code plays a multiple role in this paper. In addition to being the code of choice for the “theorem-candidate”, it can be added onto the transparent proof to make the proof itself error-tolerant. Moreover, the ideas involved motivate parts of the construction of the “transparent proof”.

Theorem 4.6 *Given $\varepsilon > 0$ and a sufficiently large positive integer N_0 , there exists an integer N , $N_0 \leq N < N_0(\log N_0)^{1/\varepsilon}$ and an error-correcting code with the following properties:*

(a) given a message string $x \in \{0, 1\}^N$, one can compute in time $N^{1+\varepsilon}$ the codeword $E(x)$ of length $m(N) \leq N^{1+\varepsilon}$;

(b) the Hamming distance of any two valid codewords is at least 75% of their length;

(c) given random access to a string y of length $m(N)$, a $(\log N)^{O(1/\varepsilon)}$ -time Monte Carlo algorithm will, with large probability, output “accept” if y is within 10% of a valid codeword, and “reject” if y is not within 15% of a valid codeword;

(d) if y is within 15% of a (unique) valid codeword y' then a $(\log N)^{1/\varepsilon}$ -time Monte Carlo algorithm will be able to compute any digit of y' with large confidence.

Proof. We choose N in the form $N = 2^{m\ell} \cdot k$ where $k - \ell$ is slightly greater than $2 \log m$; and $(k - \ell)/\ell < \varepsilon$. So $\ell \approx 2 \log m / \varepsilon$. Let \mathbf{F} be a finite field of order $q = 2^k$ and $H \subset \mathbf{F}$ be a subset of size 2^ℓ . We identify \mathbf{F} with $\{0, 1\}^k$ and thereby the message string with a function $x : H^m \rightarrow \mathbf{F}$. A message of N breaks into $2^{m\ell}$ tokens, each an element of \mathbf{F} .

We shall perform a two-stage (concatenated) encoding. The first stage associates with x its unique $|H|$ -smooth extension to \mathbf{F}^m which we denote by $E_0(x)$ (Proposition 4.1). $E_0(x)$ is a token-string of length 2^{mk} .

Clearly, the bit-length of these strings is $N(N/k)^{(k-\ell)/\ell} < N^{1+\varepsilon}$; and they are computable (if done in proper order) at polylogarithmic cost per token.

The degree of $E_0(x)$ as a polynomial of m variables is $\leq m|H|$. Therefore, by the quoted result of J. T. Schwartz (Lemma 4.2) this polynomial cannot vanish on more than a $m|H|/|\mathbf{F}|$ fraction of its domain \mathbf{F}^m . We observe:

$$m|H|/|\mathbf{F}| = m/2^{k-\ell} < 1/m. \quad (3)$$

But $m \log m \approx \varepsilon \log N$, so $1/m$ will be small for fixed ε and sufficiently large N . This justifies statement (b) (for tokens). (c) and (d) (for tokens) follow from Corollary 4.5.

Now to switch to bits from tokens, we have to apply an encoding of the tokens themselves. Here we have a large degree of freedom; e.g. Justesen's codes will work (cf. [MS77, Chap.10.11]). The properties required are now easily verified. \square

5 LFKN-type protocols

5.1 Verification of large sums

We shall have to consider the following situation: let \mathbf{F} be a field. (We shall use $\mathbf{F} = \mathbf{Q}$.) Assume we are given a polynomial of low degree in m variables over \mathbf{I}^m where \mathbf{I} is a (small) finite subset of \mathbf{F} . We have to verify that

$$\sum_{u \in H^m} f(u) = a \quad (4)$$

for some small $H \subseteq \mathbf{I}$ and $a \in \mathbf{F}$. ($|H|$ will be polylogarithmic.) We assume that we have random access to a database of values of f as well as their partial sums

$$f_i(x_1, \dots, x_i) := \sum_{x_{i+1} \in H} \cdots \sum_{x_m \in H} f(x_1, \dots, x_m). \quad (5)$$

over \mathbf{I} . Clearly, $f_m = f$, and

$$f_{i-1} = \sum_{h \in H} f_i(x_i = h) \quad (6)$$

(using self-explanatory notation).

Assumption on the database. We assume that the database gives the correct values of f but we allow that it give false values of the other f_i .

Protocol specifications. The protocol is required to *accept* if the entire database is correct and equation (4) holds; and *reject* with large probability if equation (4) does not hold (regardless of the correctness of the database).

We review the protocol which is a slight variation of the one used for an analogous purpose in [BFL91, Proposition 3.3]. The protocol builds on the technique of Lund, Karloff, Fortnow, and Nisan [LFKN92].

The protocol will work assuming the degree of f is $\leq d$ in each variable (f is d -smooth), and $|\mathbf{I}| \geq 2dm$. The protocol proceeds in rounds. There are m rounds.

At the end of round i , we pick a random number $r_i \in \mathbf{I}$; and compute a “stated value” b_i . We set $b_0 = a$. It will be maintained throughout that unless our database is faulty, for each i , including $i = 0$,

$$b_i = f_i(r_1, \dots, r_i). \quad (7)$$

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

Now we use the database to obtain the coefficients of the univariate polynomial

$$g_i(x) = f_i(r_1, \dots, r_{i-1}, x) \quad (8)$$

(by making $d + 1$ queries and interpolating). Let \tilde{g}_i denote the polynomial thus obtained. We perform a **Consistency Test**; with equation (6) in mind, we check the condition

$$b_{i-1} = \sum_{h \in H} \tilde{g}_i(h). \quad (9)$$

If this test fails, we reject; else we generate the next random number $r_i \in \mathbf{I}$ and declare $b_i := \tilde{g}_i(r_i)$ to be the next “stated value”. After the m^{th} round we have the stated value b_m and the random numbers r_1, \dots, r_m ; and we perform the **Final Test**

$$b_m = f(r_1, \dots, r_m). \quad (10)$$

We accept 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 equation (4) does not hold but the data pass the Consistency Tests then we obtain false relations involving polynomials with fewer and fewer variables; eventually reaching a constant, the correctness of which we can check by a *single* substitution into the polynomial behind the summations.

Proof of correctness of the protocol. Assume first that equation (4) holds and the database is correct. Then we shall always have $\tilde{g}_i = g_i$ and eventually accept.

Assume now that at some point, there is a mistake: $\tilde{g}_{i-1} \neq g_{i-1}$. Here we allow $i = 1$; we define the constant polynomial $\tilde{g}_0 := b_0 = a$. Then with probability $\geq 1 - d/|\mathbf{I}|$, $b_{i-1} = \tilde{g}_{i-1}(r_{i-1}) \neq g_{i-1}(r_{i-1})$ since two different univariate polynomials of degree $\leq d$ cannot agree at more than d places. Assuming now that the next Consistency Test is passed (equation (9)) it follows that the same error must occur in the next round: $\tilde{g}_i \neq g_i$.

If now equation (4) does not hold, then the constant $a = b_0 = \tilde{g}_0$ differs from $g_0 = f_0$, an error occurs in round 0. It follows that unless one of the Consistency Tests fails, with probability $\geq 1 - dm/|\mathbf{I}|$, the same error will have to occur in each round. But the error in the last round is discovered by the Final Test. \square

5.2 Simultaneous vanishing

In [BFL91], simultaneous vanishing of all values $f(x), x \in D$ was reduced to the statement $\sum_{x \in D} f(x)^2 = 0$. This trick works over subfields of the reals and will be used in the main procedure (Section 8). However, if we wish to avoid large-precision arithmetic, a different approach is required. We modify a procedure described in [BFL91, Section 7.1].

The situation is similar to Section 5.1, except that rather than verifying equation (4), we have to *verify that $f(u) = 0$ for each $u \in H^m$* . In this section we show how to reduce this problem to the result of Section 5.1.

Let us extend the (small) field \mathbf{F} to a large field \mathbf{K} where $2|H|^m \leq |\mathbf{K}| < 2|H|^m|\mathbf{F}|$. Let $|H| = d$. Let $\rho : H \rightarrow \{0, 1, \dots, d-1\}$ be a bijection; and for $u = (u_0, \dots, u_{m-1}) \in H^m$, set $\sigma(u) = \sum_{i=0}^{m-1} d^i \rho(u_i)$. So, $\sigma : H^m \rightarrow \{0, \dots, d^m - 1\}$ is a bijection.

Let us now consider the univariate polynomial $p(t) = \sum_{u \in H^m} f(u)t^{\sigma(u)}$. Unless all the $f(u)$ are zero, a random $\xi \in \mathbf{K}$ has probability $\leq 1/2$ to be a root. We show that checking $p(\xi) = 0$ for a given $\xi \in K$ is an instance of equation (4).

Indeed, let $\xi_i = \xi^{d^i}$; then

$$\xi^{\sigma(u)} = \prod_{i=0}^{m-1} \xi_i^{\rho(u_i)} = \prod_{i=0}^{m-1} \left(\sum_{h \in H} \xi_i^{\rho(h)} L_h(u_i) \right) \quad (11)$$

where the L_h are Lagrange interpolation polynomials: for $h, h_1 \in H$, $L_h(h_1) = 1$ if $h = h_1$ and zero otherwise. The right hand side is therefore a product of polynomials of degree $\leq (d-1)$ of each u_i , and the protocol of Section 5.1 can be used to verify that $p(\xi) = 0$.

Having verified this for a number of independent random choices of $\xi \in \mathbf{K}$, we are assured that all the $f(u)$ are zero.

6 Pointer machines and the Kolmogorov–Uspenskiĭ thesis

In order to apply the theory efficiently to actual computations and mathematical proofs, we need a formalization of the concept of proofs, more accurately reflecting their perceived length.

Within an accuracy of polynomial-time transformations a formalization of proofs is quite established. It is based on the nondeterministic Turing machine model. Proofs could be represented as witnesses to instances of any *NP*-complete problem (say 3-coloring). Their length and checking time will be bounded by a polynomial of $\|T, P\|$ for any reasonable formal system.

We need a greater accuracy. First, our polylogarithmic time checker has no direct way to verify the polynomial time transformations. We can afford only very trivial (polylog-time) reductions on instances (below, they will simply prefix the input with short strings). Second, we intend to transform very long mathematical proofs into transparent form. Squaring the length of the proof of the Enormous Theorem would seem too much.

Better accuracy is harder to achieve in machine-independent terms. One possibility is to accept the thesis of Kolmogorov and Uspenskiĭ [KU58] that the Pointer Machine model of computation proposed there (the original and cleaner version of RAM; see below) simulates, with *constant* factor time overhead, any other realistic model (including formal mathematical proofs).

This thesis suggests the following solution. We define the class *NF* of problems solvable in nearly linear time on nondeterministic pointer machines (or, equivalently, RAM’s). (*NF* = “Non-deterministic Fast”; we use the term “*fast*” as a synonym to “in nearly linear time”.) We find an *NF*-complete problem with respect to our very restrictive reduction concept. We use the witnesses of this problem as “proofs”. This defines a specific proof system, for which we design our checker. Proofs in other systems can then be padded to allow nearly-linear time verifications and reduced to our complete problem.

6.1 Pointer Machines and RAMs

Kolmogorov-Uspenskiĭ Machines (often called Pointer Machines) are an elegant combinatorial model of realistic computation. Pointer Machines are equivalent to RAM’s within the accuracy relevant for us (polylog factors). We describe a slightly generalized version (to directed graphs) due to A. Schönhage [Sch80a]. This definition is not required for the technical details of the proofs but it may contribute to conceptual clarity.

The memory configuration of a *Pointer Machine* (PM) is a directed graph with labeled edges. The set of labels (*colors*) is finite and predefined independently of the input. Edges coming out of a common node must differ in colors (thus constant outdegree). There is a special node O , called the *central node*. All nodes must be reachable from O via directed paths.

The input of a PM is the starting memory configuration. Based on the configuration of the constant depth neighborhood of O (w.l.o.g. depth 2), the *program* of PM chooses and performs a set of *local* operations: delete or create an edge, create a new node connected to O , halt. The operations transform the configuration step by step until the halt. The final configuration represents the output. The nodes (and their edges) which become unreachable from O are considered deleted.

Now we compare this PM model to RAM's. Our variety of RAM has an array of registers R_i with content $m(R_i)$. These include R_1, \dots, R_n , initially storing the *input*, and a Central Register $R_0 = (s, a, w, t)$. R_0 has a time counter t and $O(\|t, n\|)$ other bits. The RAM works as a Turing Machine on R_0 and, at regular intervals, *swaps* the contents of s and $R_{m(a)}$. A copy t' of t is maintained in a part of the field s , except that at each swap it is overwritten for a moment. At the start, the RAM reads consecutively the entire input.

Proposition 6.1 *PM's and RAM's can simulate each other in nearly linear time.*

6.2 An NF -complete Problem.

Let f be a function which transforms strings w called “witnesses” into “acceptable instances” $x = f(w)$. We say that f is fast if it is computable in time nearly linear in $\|f(w)\|$. An NF problem is a task to invert a fast function. (Note that this definition requires a sufficiently strong model, such as Pointer Machines or RAM's.) Representation of objects (say graphs) as strings is flexible, since nearly linear time is sufficient for translation of various representations into each other.

A *fast reduction* of problem P to Q is a pair of mappings f, h . The *instance transformation* f , computable in polylogarithmic time, maps the range of P into the range of Q . The *witness transformation* h , computable in nearly-linear time, maps the inverses: $P(h(w)) = x$ whenever $f(x) = Q(w)$.

We need a combinatorial problem with witnesses reflecting space-time histories of nearly-linear time computations.

The *history* of the work of a RAM is the the sequence of *records* $m(R_0)$, for all moments of time. Each record contains two time values t and t' in s : copied from t or swapped from $R_{m(a)}$. Each time value may appear in two records: at its own time and at the next time the same register $R_{m(a)}$ will be accessed again. Clearly, any error in the history creates an inconsistency of two records (including the inputs). It can be either a Turing error or some “read” inconsistent with the “write” at the last access time or two “reads” indicating the same last access time. We can easily find the errors by comparing two copies of the history: sorted by t and sorted by t' . The contradicting records will collide against each other. So we have accomplished three goals:

- The length of our history is nearly linear in time: a contrast to quadratic lengths (linear space times linear time) of customary (Turing machine or RAM) histories.
- Its verification can be done on a sorting network of fixed structure. (Simulation by a fixed structure network was first done in [Ofman 65].)
- The verification takes polylogarithmic *parallel* time. Thus the space-time record of the verification is also nearly linear. Note that only the *verification*, not the construction of the history can be so parallelized. So, only non-deterministic problems allow such reductions.

To implement the verification we need a sorting network of nearly linear width and polylogarithmic depth. Of course, the sorting gates will be simple circuits of binary gates comparing numbers in binary. This network will also need to perform a simple computation verifying other minor requirements.

The history of this verification (assigning the “color” i.e. the bit stored at each gate at each time) is a coloring of the *Carrier* Graph on which the network is implemented. The verification is successful if the coloring of the carrier graph satisfies certain local requirements. So, we obtain the following NF -complete problem: *Extend a given partial coloring (the input part) of a sorting network to a total coloring, so that only a fixed finite set of permitted types of colored neighborhoods of given fixed depth occurs.* We may extend the number of colors to represent such a whole neighborhood as the color of a single node. Then consistency need only be required from pairs of colors of adjacent nodes.

6.3 The Domino Problem

We now describe a simplified NF -complete problem. We choose a family of *standard* directed graphs G_n of size $\Theta(2^n)$. We denote by $g_{n,k,c}(i)$ the k^{th} digit of the binary name of the c^{th} neighbor of node i . It does not

make much difference what G_n we choose (say, Shuffle Exchange Graph), as long as g can be represented by a boolean formula of the digits of i , computable from n, c, k in time $n^{O(1)}$ and a polylog time sorting network can be implemented on G_n . A coloring of a graph is a mapping of its nodes into integers (colors). The *base* is the subgraph of $(0, 1)$ -colored nodes. We require it to be an initial segment of consecutive nodes of G_n . A *domino* is a two-node induced colored subgraph, with nodes unlabeled. The *Domino Problem* is to color the standard graph to achieve given domino set and base (specified as a string of colors).

Proposition 6.2 *The Domino Problem is NF-complete, i.e. all NF problems have fast reductions to it.*

The instance transformation will leave the base string intact and supplement it with a fixed domino set, specific for the problem. Alternatively, we can use a fixed *universal* domino set and prepend the input with a problem-specific prefix to get the base string. The idea of the construction was outlined in section 6.2. Other (straightforward) details will be given in the journal version.

7 Arithmetization of admissibility of coloring

The purpose of this section is to turn the essentially Boolean conditions describing the admissibility of a coloring A of the standard graph G_n of the Domino Problem (Section 6.3) into algebraic conditions of the following type: *some family of low degree polynomials of several variables must vanish on all substitutions of the variables from some small set.*

Let C be the set of colors and D the set of admissible dominoes. Let G_n be the standard graph referred to in the domino problem, with edge set E , vertex set V , and functions $B_1, B_2 : E \rightarrow V$ and $B_3 : E \rightarrow \{0, 1\}$ describing the head, the tail, and the type of each edge. (The “type” is either directed or undirected.)

Let now D_0 be the set of all conceivable dominoes. For $\delta \in D_0$, let $\psi_\delta : C \times C \times \{0, 1\} \rightarrow \{0, 1\}$ be the predicate describing the statement $\psi_\delta(\gamma_1, \gamma_2, \epsilon)$ that an edge of type δ has type ϵ with its head colored γ_1 and its tail colored γ_2 .

Let \mathbf{F} be a field. We assume $C \subset \mathbf{F}$. Let $A : V \rightarrow \mathbf{F}$ be a function. We wish to express by arithmetic formulas that A is an admissible coloring of V . By an *arithmetic formula* we mean a correctly parenthesized expression involving the operations $+$, $-$, \times , variable symbols, and constants from \mathbf{F} .

First we observe that the statement that A is an admissible coloring of G_n is equivalent to the following:

$$\begin{aligned} & (\forall \delta \in D_0 \setminus D)(\forall e \in E) \\ & (\psi_\delta(A(B_1(e)), A(B_2(e)), B_3(e)) = 0); \\ & (\forall v \in V)(A(v) \in C). \end{aligned} \tag{12} \tag{13}$$

As in Section 4.1, let $H \subset \mathbf{F}$ be a set of size 2^ℓ where $\ell = \log n / \varepsilon$. (We choose ε so as to make this an integer.) We may assume also that $m := n / \ell$ is an integer. We embed both E and V into a Cartesian power of H : $E, V \subseteq H^m$. Furthermore, we identify H with $\{0, 1\}^\ell$. So the elements of H are represented as binary strings of length ℓ . For $h \in H$, let $\pi_j(h)$ denote the j^{th} bit of h ($1 \leq j \leq \ell$). For $v = (h_1, \dots, h_m) \in V$, let us call the h_i the *tokens* of v ; and the bits of the h_i the *bits* of v . So v has m tokens and $m\ell = n$ bits. We use the same terminology for E . After a slight technical trick we may assume that $V = E = H^m$.

We assume B_1, B_2, B_3 are given in the form of a family of Boolean functions defining the bits of the output in terms of the bits of the input. We assume that these functions are given by Boolean formulas, where the entire collection of these formulas is computable from n in time n^{c_1} .

Let us *arithmetize* these formulas; i.e. create equivalent arithmetic expressions in the same n variable symbols which give the same value on Boolean substitutions. This is easily accomplished by first eliminating \vee 's from the Boolean formulas (replacing them by $\neg \wedge \neg$), and subsequently replacing each \wedge by multiplication and each subformula of the form $\neg f$ by $(1 - f)$. The length of the arithmetic expression created is linear in the length of the initial Boolean formula.

So we now may assume that B_1, B_2, B_3 are given by arithmetic expressions over \mathbf{F} describing families of polynomials of degree n^{c_1} in $m\ell$ variables. We now wish to turn this representation in n variables over $\{0, 1\}$ into a representation in m variables over H .

Each projection function $\pi_j : H \rightarrow \{0, 1\} \subset \mathbf{F}$ can be viewed as a polynomial of degree $\leq |H| = n^{1/\varepsilon}$ in a single variable over \mathbf{F} . Combining these families of polynomials, we obtain the families of composite polynomials

$$P_i(u_1, \dots, u_m) = B_i(\pi_1(u_1), \pi_2(u_1), \dots, \pi_\ell(u_m)). \quad (14)$$

The degree of this polynomial is $\leq n^{c_1+1/\varepsilon}$.

Similarly, for each $\delta \in D_0$, ψ_δ is a polynomial of degree $\leq |C|$ in each of γ_1 and γ_2 and linear in ϵ so its total degree is $\leq 2|C| + 1$ (by Proposition 4.1).

Let now $A : V \rightarrow \mathbf{F}$ be an arbitrary function.

First, let us consider the following polynomial f of degree $|C|$ in a single variable:

$$f(t) = \prod_{\gamma \in C} (t - \gamma) = 0. \quad (15)$$

Now the statement that A is a coloring of V is equivalent to

$$(\forall v \in V)(f(A(v)) = 0). \quad (16)$$

Next, we consider the polynomials P_i defined above. Setting

$$\varphi_\delta^A(\epsilon) = \psi_\delta(A(P_1(\epsilon)), A(P_2(\epsilon)), P_3(\epsilon)) \quad (17)$$

we observe that the coloring A is admissible precisely if

$$(\forall \delta \in D_0 \setminus D)(\forall \epsilon \in E)(\varphi_\delta^A(\epsilon) = 0) \quad (18)$$

where each edge ϵ is viewed as a member of H^m . We summarize the result.

Proposition 7.1 *There exists a family of arithmetic formulas $P_1, P_2, P_3, \psi_\delta, f$ computable from n in time $|H|n^{O(1)}$ such that a function $A : V \rightarrow \mathbf{F}$ represents an admissible coloring of G_n if and only if conditions (16) and (18) hold.*

This result almost accomplishes our goal, except that A is not a polynomial. Let us now consider the unique extension $\tilde{A} : \mathbf{F}^m \rightarrow \mathbf{F}$ of A as a polynomial which has degree $\leq |H|$ in each variable (4.1). Replacing A by \tilde{A} in the formulas (16) and (18), we obtain arithmetic conditions in terms of the vanishing of certain polynomials of degree $\leq n^{1+c_1+1/\varepsilon}$ over H^m (the set which was identified with V and E).

8 The procedure

We use the code of Theorem 4.6 to encode the “theorem-candidate”. The proof system includes this information and is encoded into our instance of the domino problem. The “proof-candidate” is therefore a coloring of the standard graph in the Domino Problem.

We define the parameters n, m, ℓ and the set H as suggested in Section 4.1, with $\mathbf{F} = \mathbf{Q}$ and $|\mathbf{I}| = \Theta(n^2|H|)$. The Solver is asked to extend the coloring A to an $|H|$ -smooth polynomial over \mathbf{F}^m .

The verification consists of the LFKN-type protocol (Section 5.1), employed to verify that the sum of squares of the quantities in equations (16) and (18) vanishes.

The “transparent proof” will consist of a collection of databases: one for the extended A ; others for each LFKN-type partial sum encountered in the verification (according to Sections 5, 7). We test $|H|$ -smoothness of the extended A .

Let P_0 be the collection of the databases obtained. It should be clear that P_0 qualifies as a transparent proof, except that it does not have the error-tolerance stated in Section 1.2: at this point the Checker is allowed to reject on the grounds of a single bit of error.

In order to add error-tolerance, we encode P_0 according to Theorem 4.6. to obtain the transparent proof P' . The Checker operates on P' by locally reconstructing each bit of P_0 as needed. If P_0 was incorrect even in a single bit, then P' will be more than 10% away from any correct proof.

Remarks. **1.** The encoding of the Theorem-candidate does not need to use the encoding of Theorem 4.6. Any error-correcting code will do; the code of Theorem 4.6 can be incorporated in the transparent proof and serve the same purpose (correction of any bit of T in polylog time). **2.** Rather than working over \mathbf{Q} , we could use the trick of Section 5.2 in the protocol. This would require a separate LFKN protocol for each $\xi \in \mathbf{K}$, thus squaring the length of the transparent proof. This blowup can be avoided with the following 3/2-round interactive protocol: 1. Prover: low degree extension of coloring; 2. Verifier: random $\xi \in K$; 3. Prover: $p(\xi) = 0$.

Acknowledgements. We are grateful to Manuel Blum, Joan Feigenbaum, and Avi Wigderson for their insightful comments.

References

- [AHK77] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.
- [ALM⁺92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 14–23. IEEE, New York, 1992.
- [AS92] S. Arora and S. Safra. Approximating clique is NP-complete. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13. IEEE, New York, 1992.
- [BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Berlin, 1990.
- [BF91] L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1(1):41–66, 1991.
- [BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.
- [BGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 113–131. ACM, New York, 1988.
- [BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 1–10. ACM, New York, 1988.
- [BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proceedings of the 21st ACM Symposium on the Theory of Computing*, pages 86–97. ACM, New York, 1989.
- [FGL⁺91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 2–12. IEEE, New York, 1991.
- [FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Proceedings of the 3rd IEEE Structure in Complexity Theory Conference*, pages 156–161. IEEE, New York, 1988.
- [Gor85] D. Gorenstein. The enormous theorem. *Scientific American*, 253(6):104–115, December 1985.

- [KU58] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *Uspehi Mat. Nauk*, 13(4):3–28, 1958. Translation in [KU63].
- [KU63] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *American Mathematical Society Translations*, 29:217–245, 1963.
- [LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.
- [Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191 – 202. American Mathematical Society, Providence, 1991.
- [MS77] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.
- [PW72] W. Peterson and W. Weldon, Jr. *Error-correcting Codes*. MIT Press, 1972.
- [Sch80a] A. Schönhage. Storage modification machines. *SIAM Journal on Computing*, 9(3):490–508, 1980.
- [Sch80b] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.
- [Sha92] A. Shamir. $IP = PSPACE$. *Journal of the ACM*, 39(4):869–877, 1992.