Probabilistic Checking of Proofs: A New Characterization of NP*

Sanjeev Arora† *Shmuel Safra*‡

Abstract

We give a new characterization of NP: the class NP contains exactly those languages L for which membership proofs (a proof that an input x is in L) can be verified probabilistically in polynomial time using *logarithmic* number of random bits and by reading *sub-logarithmic* number of bits from the proof.

We discuss implications of this characterization; specifically, we show that approximating Clique and Independent Set, even in a very weak sense, is NP-hard.

CATEGORIES AND SUBJECT DESCRIPTORS: F.1.2 (Modes of Computation); F.1.3 (Complexity Classes); F.2.1 (Numerical Algorithms); F.2.2 (Nonnumerical Algorithms); F.4.1 (Mathematical Logic).

1 Introduction

Problems involving combinatorial optimization arise naturally in many applications. For many problems, no polynomial-time algorithms are known. The work of Cook, Karp, and Levin [Coo71, Kar72, Lev73] provides a good reason why: many of these problems are *NP-hard*. If they were to have polynomial-time algorithms, then so would every NP decision problem, and so P = NP. Thus if P $\neq$ NP — as is widely believed — then an NP-hard problem has no polynomial-time algorithm.

In the two decades following the Cook–Karp–Levin work, classifying computational problems as tractable (i.e., in P) or NP-hard has been a central endeavor in computer science. But one important family of problems, by and large, defies such a simple classification: the problem of computing *approximate* solutions to NP-hard problems. For a number $\alpha > 1$, an algorithm is said to *approximate* an optimization problem *within a factor* α if it produces, for every instance of the problem, a solution whose cost is within a factor α of the optimum cost. For example, let the *clique number* of a graph G, denoted $\omega(G)$, be the size of the largest subset of vertices of G whose every two members are adjacent to each other. (Computing ω is a well-known NP-hard problem.) To approximate the clique number within a factor α, an algorithm needs to output, for every graph G, a clique in G of size at least $\omega(G)/\alpha$. (Thus the closer α is to 1, the better the algorithm.)

Approximation versions of most NP-hard problems are not known to be in P, at least for "reasonable" factors of approximation. In all these cases, one might conjecture that approximation is NP-hard, but demonstrating this —even for very small factors— has proved difficult. The clique problem is a good example. The best polynomial-time algorithm

*A preliminary version of this paper [AS92] appeared in the *Proceedings of the 33rd IEEE Foundations of Computer Science*, pages 2-12, 1992.

†This work was done while the author was at CS Division, UC Berkeley, under support from NSF PYI Grant CCR 8896202 and an IBM graduate fellowship. Current email and address: `arora@cs.princeton.edu.` 35 Olden St., Princeton NJ 08544.

‡This work was done while the author was with Stanford University and IBM Almaden. Current email and address: `safra@math.tau.ac.il.`, Math Dept., Tel Aviv University, Israel.

approximates clique number within a factor $O(n/\log^2 n)$ [BH92], whereas, until recently, it was not known even if approximating within a factor $1+\epsilon$, for any fixed $\epsilon > 0$, is NP-hard.

Feige, Goldwasser, Lovász, Safra, and Szegedy [FGL$^+$91] recently provided a breakthrough, by showing that if the clique number can be approximated within any constant factor in polynomial time, then every NP problem can be solved deterministically in time $n^{O(\log\log n)}$. Since some NP problems (SAT, for example) are widely believed to have no subexponential-time algorithms, this result provides a strong reason to believe that the clique number has no good approximation algorithms. (The authors therefore proclaimed clique approximation "almost" NP-hard.)

At the core of the [FGL$^+$91] result is a new technique for doing reductions, which uses recent results from the theory of *interactive proofs.* The use of this radically new technique (not to mention the fact that it shows the problem is "almost"-NP-hard instead of NP-hard), suggests that the [FGL$^+$91] result, though impressive, is not the end of the story.

The results in this paper confirm this. We show, firstly, that approximating the clique number within any constant factor is NP-hard (in fact, we can also show the NP-hardness of approximating the clique number within a factor $2^{\log^{0.5-\epsilon} n}$, where n is the number of vertices in the graph and ϵ is an arbitrarily small positive constant). We use techniques derived from those in [FGL$^+$91] and some earlier papers.

Second, we provide a new, and surprising, characterization of the class NP. As we will describe soon, this characterization is the logical culmination of recent results about interactive proofs, which have provided new characterizations for traditional complexity classes such as PSPACE and NEXPTIME.

In fact, our NP-hardness result for clique approximation is a corollary of our new characterization of NP. An earlier draft of this paper posed the question whether other hardness results can be derived from this new characterization. This question has been answered — positively — by a series of swift developments that followed this paper. Section 6 discusses those developments.

Section 6 also discusses how our ideas have figured in subsequent research. Two of these are *verifier composition* and an improved *low degree test.*

1.1 Context of this work

We briefly discuss recent results in complexity theory and where our work fits in relation to them.

Interactive Proofs

The model of *interactive proofs* was introduced by Goldwasser, Micali and Rackoff [GMR89] for cryptographic applications, and by Babai [Bab85] as a game-theoretic extension of NP. The model consists of a probabilistic polynomial-time verifier V communicating with a prover P who tries to convince V that the input x is in a language L. A language L is in IP (for **i**nteractive **p**roofs) if there exists a verifier V that is always convinced when $x \in L$, but if $x \notin L$ then any prover P has only a small probability of convincing V to the contrary.

The model of *multi-prover interactive proofs* was introduced by Ben-Or, Goldwasser, Kilian, and Wigderson [BGKW88]. The model consists of a random polynomial-time verifier V communicating with two infinitely powerful provers who cannot communicate with each other during the protocol. The provers try to convince the verifier that the input x is in a language L. A language L is in the class MIP (for **m**ulti-prover **i**nteractive **p**roofs) if there exists a V that is always convinced when $x \in L$ and, but if $x \notin L$, then the provers have only a small probability of convincing V to the contrary.

Proof Verification

An equivalent formulation of the class MIP was suggested by Fortnow, Rompel and Sipser [FRS88]. In this model, a random polynomial-time Turing machine M is interacting with an oracle O_x that is trying to convince M that $x \in L$. As in the multi-prover model, when $x \in L$, there is an oracle that convinces the verifier always, and when $x \notin L$ any oracle has only a small probability of convincing the verifier to the contrary.

Since an oracle's replies (unlike a prover's) are fixed in advance, we can think of an oracle as a string of bits to which the verifier has random access (i.e., the verifier can read individual bits of this string). This string is expected to represent a proof that $x \in L$ (in other words, a *membership proof*). Hence, MIP can be characterized as all languages L for which a membership proof can be verified probabilistically in polynomial-time. (Realize that the verifier, having random access to the proof, could conceivably check proofs of even exponential size, since accessing any bit in the proof only requires writing its address.)

The Unexpected Power of Interaction

The classes IP and MIP seem quite unlike traditional complexity classes. Both contain NP, but were not even thought to contain co-NP. Some evidence to this effect was provided by the *relativization* results of [FS88, FRS88], which show the existence of a language O such that if Turing machines are given access to a membership oracle for O, then co-NP is not contained in MIP (or IP).

It therefore came as a surprise when Lund, Fortnow, Karloff, and Nisan [LFKN92] and Shamir [Sha92] showed, using techniques developed for program checking [BK89, BLR90, Lip89], that that IP = PSPACE. (The class PSPACE is believed to be quite larger than NP and co-NP.) Shortly afterwards, Babai, Fortnow and Lund [BFL91] introduced even more powerful techniques to show that MIP = NEXPTIME. Note that NEXPTIME is the set of languages that can be decided by nondeterministic exponential-time Turing machines.

Scaling Down MIP = NEXPTIME

Since NEXPTIME can be characterized as the set of languages that have exponential-size membership proofs, the result MIP = NEXPTIME [BFL91], combined with the oracle formulation of MIP [FRS88], shows that if a language has membership proofs of exponential size, then some probabilistic polynomial-time verifier can check those membership proofs.

There were two efforts to "scale down" the above result, i.e., to show efficient verification procedures for checking membership proofs for smaller nondeterministic classes, such as NP. Note that membership proofs for NP have polynomial size, so a straightforward meaning of "scaling-down" would require the running time of the verifier to be polylogarithmic. But this cannot be, since the verifier must take linear time simply to read the input.

Babai, Fortnow, Levin, and Szegedy [BFLS91] nevertheless obtained a scale-down result (including a scale-down in the running time of the verifier) by changing the model of computation: in the new model, the input has to be provided to the verifier in an encoded form using a specific error-correcting code. The authors showed that in this model, membership proofs for any language $L \in$ NP can be verified by a probabilistic verifier in polylogarithmic time. This counterintuitive result is possible because the verifier can randomly sample a small number of bits of the encoded input, use them to gain very "global" knowledge about the entire input, and then check that the membership proof is correct for the input. (Babai et al. also suggested an application of their ideas to mechanical checking of mathematical proofs. We return to this application in Section 6.)

However, if the model of computation has to be left unchanged, then a different scale-down result could still be possible: instead of scaling down the running time of the verifier, scale down just the number of *random* bits and *query* bits (number of bits looked at in the membership proof) it uses. This approach was suggested by Feige, Goldwasser, Lovász, Safra, and Szegedy [FGL+91], who showed that there exist verifiers for NP that run in polynomial time, but use only $O(\log n \log\log n)$ random bits and query bits.

MIP and the Hardness of Approximation

The above-mentioned developments were exciting. Even more exciting was a connection, discovered in [FGL+91], between their scaled-down version of MIP = NEXPTIME and the hardness of approximating the clique number.

We will describe this connection in greater detail later, but briefly stated, the two results shown were as follows. First, if there is a polynomial-time approximation procedure that approximates $\omega(G)$ even to within a factor of $2^{\log^{1-\epsilon} n}$, then any NP problem can be solved deterministically in quasi-polynomial($= n^{\log^{O(1)} n}$) time. This suggests that unless if all NP problems can be solved in "almost" polynomial time, there are no efficient algorithms for approximating $\omega(G)$ even in a very weak sense. Second, in an attempt to prove clique approximation as close to NP-hard as possible, Feige et al. showed that if $\omega(G)$ can be approximated to within any constant factor in polynomial time, then every NP problem can be solved deterministically in $n^{O(\log\log n)}$ time.

1.2 This Paper

The notion of efficiently checkable membership proofs is inherently interesting because it represents a new way of looking at classical complexity classes such as NP. The notion becomes even more intriguing in light of the possible trade-offs, hinted at in the results of [FGL+91, BFLS91], between the verifier's running time, random bits, and query bits. If these trade-offs can be improved, improved nonapproximability results for the clique problem follow, as we will soon see. To facilitate the study of such trade-offs, we define below a hierarchy of complexity classes PCP (for **p**robabilistically **c**heckable **p**roofs). Section 1.2.2 describes a new characterization of NP in terms of PCP, and shows how it leads to improved hardness results for the clique problem.

1.2.1 Probabilistically Checkable Proofs

A *verifier* is a probabilistic polynomial-time Turing machine M that is given an input x and an array of bits Π, called the *proof string* (or just *proof* for short). For an input x, random string τ and proof string Π, define $M^{\Pi}(x,\tau)$ to be 1 if M accepts x using τ after examining proof string Π. Otherwise $M^{\Pi}(x,\tau)$ is 0.

The next definition formalizes a concept dating back to [FRS88].

Definition 1 Let L be a language. A verifier M *checks membership proofs for* L if it behaves as follows for every input x.

1. If $x \in L$, there is a proof Π that causes M to accept for every random string, i.e.,
$$\Pr_{\tau}[M^{\Pi}(x,\tau)=1]=1.$$
2. If $x \notin L$, then for all proofs Π,
$$\Pr_{\tau}[M^{\Pi}(x,\tau)=1]<\frac{1}{2}.$$

(In both cases, the probability is over the choice of the random string τ.)

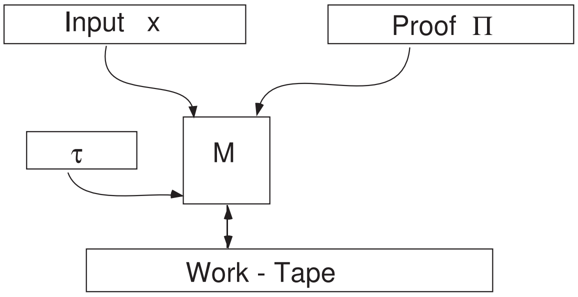

Figure 1: A Probabilistically Checkable Proof (PCP) System. Proof Π is an array of bits to which the verifier M has random access.

In [FGL+91], a new twist was introduced in this setting: there is a (startlingly low) limit on the number of random bits used by the verifier, and the number of bits it can read in the proof. Note that the verifier is allowed random access to proof Π; that is, it can read individual bits of Π. The operation of reading a bit of Π is called a *query*.

For integer valued functions r and q, we say that M is $(r(n), q(n))$*-restricted* if, on an input of size n, it uses at most $O(r(n))$ random bits for its computation, and queries at most $O(q(n))$ bits in the proof string.

More specifically, the $(r(n), q(n))$-restricted verifier behaves as follow on an input of size n. The verifier first reads the input x and the random string τ. Next, it computes[1], in poly(n) time, a sequence of locations $i_1(x,\tau), i_2(x,\tau), \ldots, i_{O(q(n))}(x,\tau)$. Then it reads the bits $\Pi[i_1(x,\tau)], \ldots, \Pi[i_{O(r(n)}(x,\tau)]$ from the proof onto its work-tape. Here $\Pi[i]$ denotes the ith bit of proof Π. Then the verifier computes further for poly(n) time before deciding to accept or reject.

Definition 2 A language L is in *PCP*$(r(n), q(n))$ if there is an $(r(n), q(n))$-restricted verifier that checks membership proofs for L.

By definition, NP is PCP$(0, poly(n))$, the class of languages for which membership proofs are checkable in deterministic polynomial-time. Further, MIP, the class of languages for which membership proofs can be checked by a probabilistic polynomial-time verifier, is just PCP(poly(n), poly(n)).

Also, we remark that $\text{PCP}(r(n), q(n)) \subseteq \text{Ntime}(2^{O(r(n))}q(n) + \text{poly}(n))$. The reason is that an $(r(n), q(n))$-restricted verifier has at most $2^{O(r(n))}$ possible runs —one for each choice of its random string— and in each run it reads at most $O(q(n))$ bits in the proof string. Hence, over all runs, it reads at most $2^{O(r(n))}q(n)$ bits from the proof string. To decide whether there exists a proof string which the verifier accepts with probability 1, a nondeterministic Turing machine "guesses" the proof string in $2^{O(r(n))}q(n)$ time, and then deterministically goes through every possible run of the verifier. Thus every language in PCP$(r(n), q(n))$ is in $\text{Ntime}(2^{O(r(n))}q(n))$.

The above-mentioned paper [FGL+91] implicitly defined a hierarchy of complexity classes similar to PCP. The hierarchy was unnamed there, so for sake of discussion we give it the name MO (for "memoryless oracle," a term used in that paper). For every integer-valued function c such that $c(n) \leq \text{poly}(n)$, the class $MO(c(n))$ is the same as our PCP$(c(n), c(n))$. The [FGL+91] paper showed that NP $\subseteq MO(\log n \cdot \log\log n)$. Since, as noted above, $MO(\log n \cdot \log\log n)$ is in turn contained in $\text{Ntime}(n^{O(\log\log n)})$, this result shows that

[1]Note that we are restricting the verifier to query the proof nonadaptively: the sequence of locations queried by it depend only on the input and the random string, and not upon the bits it may already have queried in Π. The original draft of this paper allowed verifiers to query the proof adaptively. This caused confusion among readers, since the verifiers we actually construct in the paper are nonadaptive. Also, the *Composition Lemma*, our most important lemma, relies crucially on verifiers being nonadaptive.

$MO(\log n \cdot \log\log n)$ is sandwiched between NP and Ntime($n^{O(\log\log n)}$). (We note that NP $\subseteq$ PCP(poly($\log n$), poly($\log n$)), a slightly weaker result, was also implicit in [BFLS91].)

1.2.2 A New Characterization of NP and its Applications

As mentioned above, NP = PCP(0, poly(n)). An interesting open question arising from the "sandwich" result implicit in [FGL+91] was whether NP has an exact characterization in terms of PCP. Our main theorem settles this question.

Theorem 1 (Main) *NP = PCP*$(\log n, \log n)$.

Theorem 1 is probably optimal, in the following sense: if NP $\subseteq$ PCP($o(\log n)$, $o(\log n)$), then NP = P. This implication is a consequence of a reduction in [FGL+91] (see Theorem 26 in the Appendix), which reduces every language in $L \in$ PCP($o(\log n)$, $o(\log n)$) to sublinear instances of the clique problem; that is, it reduces the membership problem for inputs of size n to clique instances of size $n^{o(1)}$. Thus if NP $\subseteq$ PCP($o(\log n)$, $o(\log n)$), we can reduce the clique problem on graphs of size n to the clique problem on graphs of size $n^{o(1)}$. By iterating this reduction, we can reduce the clique problem on graphs of size n to the clique problem on graphs of size $O(\log n)$, which is trivially in P.

However, the above argument leaves open the possibility that NP $\subseteq$ PCP($\log n$, $o(\log n)$). As a matter of fact, we can prove the following result.

Theorem 2 *For every fixed $\epsilon > 0$, NP = PCP*$(\log n, \log^{0.5+\epsilon} n)$.

Note that Theorems 1 and 2 together imply that PCP($\log n$, $\log n$) = PCP($\log n$, $\log^{0.5+\epsilon} n$), thus raising the question (which was actually raised in an early draft of this paper) whether the $O(\log^{0.5+\epsilon} n)$ query bits could be reduced further.

Soon after the initial circulation of the draft of this paper, it was noticed [AMS+92] that the techniques of this paper actually show NP = PCP($\log n$, $(\log\log n)^2$). Then Arora, Lund, Motwani, Sudan and Szegedy [ALM+92] showed that NP $\subseteq$ PCP($\log n$, 1). (See Section 6.)

Our new characterization of NP also allows us to prove the NP-hardness of approximating the clique problem, thus resolving an open question in [FGL+91].

Corollary 3 *For any positive constants c, ϵ, if a polynomial-time algorithm approximates the clique problem within a ratio $2^{c \log^{0.5-\epsilon} n}$, then $P = NP$.*

Proof: We show that the hypothesis implies a polynomial-time algorithm for SAT. We use a reduction from PCP to clique described in [FGL+91] (see Theorem 26 in the Appendix), and an error-amplification technique from [AKS87] (the idea of using this technique in the context of proof checking is from [Zuc91]).

Since NP = PCP($\log n$, $\log^{0.5+\epsilon/2} n$), there is a ($\log n$, $\log^{0.5+\epsilon/2} n$)-restricted verifier that checks membership proofs for SAT. The randomness-efficient error amplification in [AKS87] allows us to change the verifier into one that is ($\log n$, $\log n$)-restricted and has the following property. If $x \in SAT$, there is a proof Π_x that the verifier accepts with probability 1. But if $x \notin SAT$, the verifier accepts every proof with probability less than $p = 2^{-\frac{\log n}{\log^{0.5+\epsilon/2} n}} = 2^{-\log^{0.5-\epsilon/2} n}$.

Now apply the reduction from [FGL+91] (see Theorem 26 in the appendix) to this new verifier. Given a boolean formula of size n, the reduction produces graphs of size $2^{O(\log n)} = n^{O(1)}$. Let N denote this size. The reduction ensures that the clique number when $x \in SAT$ is higher than the clique number when $x \notin SAT$ by a factor $1/p = 2^{\log^{0.5-\epsilon/2} n}$. As a function of N, this gap is $2^{\theta(\log^{0.5-\epsilon/2} N)}$, which is asymptotically larger than $2^{c \log^{0.5-\epsilon} N}$ for every constant c. Hence if the approximation algorithm mentioned in the Lemma statement exists, then it can be used to distinguish between the cases $x \in SAT$ and $x \notin SAT$ in polynomial time, and so $P = NP$. $\square$

1.3 Other Related Works

Alternative characterizations of NP exist. Fagin [Fag74] gave a characterization in terms of spectra of second order formulae. This characterization has become the focus of renewed interest since the work of Papadimitriou and Yannakakis [PY91], in which they use Fagin's ideas to define restricted subclasses of NP optimization problems and define a notion of completeness (with respect to approximability) within the subclasses.

Recent work in program checking and interactive proof systems has resulted in other characterizations of NP. For instance, Lipton [Lip89] showed that membership proofs for NP can be checked by probabilistic logspace verifiers that have one-way access to the proof and use $O(\log n)$ random bits. It is easily seen that this is another exact characterization of NP. Lipton's work has recently been extended by Condon and Ladner [CL89] to give a somewhat stronger result. In these characterizations, the verifier, though very restricted, at least receives a polynomial number of bits of information about the proof.

Condon [Con93] used the bounded-away probability in such characterizations of NP to show the NP-hardness of approximating the Max-Word problem. This largely-unknown result was independent of (and slightly predated) the [FGL$^+$91] result.

The main difference between the above probabilistic characterizations of NP and our characterization is that they restrict the computational power of the verifier, whereas we restrict the amount of randomness available to it and the number of bits it can extract from the membership proof.

2 Overview

We prove Theorem 2 by *composing* verifiers, a new technique that is described below (see Lemma 5). This technique has played a pivotal role in all subsequent works in this area.

Our starting point is the verifier of [BFL91] in its scaled-down form [BFLS91, FGL$^+$91]. This verifier, because of its reliance on a special error-correcting code, has certain strong properties, some of which were earlier noted in [BFLS91]. We will use such properties to *compose* verifiers. Composition[2], when done correctly, improves efficiency (the resulting verifier reads very few bits from the proof). However, it requires that the verifiers have certain properties, and that one of the verifiers be in *normal form*. We will later describe how to construct such verifiers.

In addition to verifier composition, we also develop some other techniques to prove our main results. Chief among them is our improved analysis of the efficiency of the verifiers in [BFLS91, FGL$^+$91], specifically, of a procedure called the *low degree test*. (This improved analysis becomes possible through our Lemma 16.)

The rest of the paper is organized as follows. Section 2.1 defines some coding-related terms that will be used often (in fact encodings are inherent to the idea of composition). Section 3 describes the composition technique and normal form verifiers, and how they are used to prove Theorem 2. Section 4 describes a certain normal form verifier that is used in the proof of Theorem 2.

Throughout this paper we will often describe verifiers for the language 3SAT. Since 3SAT is NP-complete, verifiers for other NP languages are trivial modifications of this verifier. Throughout this paper, φ denotes a 3CNF formula that is the verifier's input, and n denotes the number of variables in it. We identify, in the obvious way, the set of bit strings in $\{0,1\}^n$ with the set of truth assignments to variables of φ.

[2]A preliminary version of this paper used the term "Recursive Proof Checking" instead of "verifier composition." We decided on this change because, as observed by several people, "recursion" was an incorrect term for the the process we were describing.

2.1 Codes and Encoding Schemes

Let Σ be a finite alphabet alphabet. If x and y are strings in Σ^m for some $m \geq 1$, then the *distance between two words x and y*, denoted $\Delta(x, y)$, is the fraction of coordinates in which they differ. (This distance function is none other than the well-known *Hamming* metric, but scaled to lie in $[0, 1]$.)

A *code $\mathcal{C}$ over the alphabet* Σ is a subset of Σ^m for some $m \geq 1$. Every word in $\mathcal{C}$ is called a *codeword*. A word y is *γ-close to $\mathcal{C}$* (or just *γ-close* when it is clear from the context what the code $\mathcal{C}$ is) if there is a codeword $z \in \mathcal{C}$ such that $\Delta(y, z) < \gamma$.

The *minimum distance* of code $\mathcal{C}$, denoted $\delta_{\min}(C)$ (or just $\delta_{\min}$ when C is understood from context), is the minimum distance between two codewords. Note that if word y is $\delta_{\min}/2$-close, there is exactly one codeword z whose distance to y is less than $\delta_{\min}/2$. For, if z' is another such codeword, then by the triangle inequality, $\Delta(z, z') \leq \Delta(z, y) + \Delta(y, z') < \delta_{\min}$, which is a contradiction. For a $\delta_{\min}/2$-close word y, denote by $\widetilde{y}$ the codeword nearest to y.

Codes are useful for *encoding* bit strings. For any integer k such that $2^k \leq |\mathcal{C}|$, let σ be a one-to-one map from $\{0, 1\}^k$ to $\mathcal{C}$ (such a map clearly exists; in our applications it will be defined in an explicit fashion). Note that the encoding satisfies, $\forall x, y \in \{0, 1\}^k$, that $\delta(\sigma(x), \sigma(y)) \geq \delta_{\min}$. We emphasize that the map σ need not be onto, that is, σ^{-1} is not defined for all codewords. An *encoding scheme* is a family of encodings, one for all string lengths.

Definition 3 (encoding scheme) An *encoding scheme* $\sigma = \{(\Sigma_n, \mathcal{C}_n, \sigma_n) : n \geq 1\}$ *with minimum distance* $\delta_{\min}$ is an infinite sequence of (alphabet, code, encoding) triples. Each Σ_n is an alphabet, each $\mathcal{C}_n$ is a code of minimum distance $\delta_{\min}$ over the alphabet Σ_n, and $\sigma_n : \{0, 1\}^n \to \mathcal{C}_n$ is an encoding of n-bit strings using $\mathcal{C}_n$.

For a string $s \in \{0, 1\}^n$, we use $\sigma(s)$ as a shorthand for $\sigma_n(s)$.

3 Normal Form Verifiers and their use in Composition

In this section we prove Theorem 2. The essential ingredients of this proof are the Composition Lemma (Lemma 5), which we will prove in this section, and Theorem 6, which we will prove in Section 4.

As already mentioned, our starting point are the verifiers for 3SAT that were constructed in [BFL91, BFLS91, FGL+91]. Now we mention some of their properties that will be of interest.

1) The proof can be viewed as a string over a non-binary alphabet. The verifier has an associated sequence of alphabets $\{\Sigma_n : n = 1, 2, 3, \ldots\}$. When the input boolean formula has size n, the verifier expects the proof to be a string over the alphabet Σ_n. A *query* of the verifier involves reading a symbol of Σ_n.

2). The verifier has a low decision time. Recall that, by definition, the verifier queries the proof nonadaptively. In other words, its computation can be viewed as having three stages. In the first, it reads the input and the random string, and decides which locations to examine in the proof. In the second stage it reads symbols from the proof string Π onto its work tape. In the third stage it decides whether or not to accept. Usually this third stage takes very little time compared to n (for concreteness, the reader may think of it as $\mathrm{poly}(\log n)$). To emphasize this fact we use a special name for the running time of the third stage: it is the verifier's *decision time*.

A clarification is in order about Property 1). As originally defined, the proof is an array of *bits*, to which the verifier has random access. This is still the case. However, the verifier

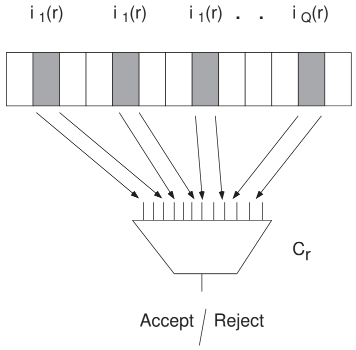

Figure 2: Using random string r, the verifier computes a tiny circuit C_r and a tuple of queries $(i_1(r), \ldots, i_Q(r))$. It accepts iff $\Pi[i_1(r)] \circ \Pi[i_2(r)] \circ \cdots \circ \Pi[i_Q(r)]$ is an input on which circuit C_r outputs "accept." The size of C_r is quadratic in the verifier's decision time.

treats this array as if it were partititioned into chunks of $\lceil \log |\Sigma_n| \rceil$ bits (each representing a symbol of Σ_n). The verifier either reads all the bits in a chunk or none at all. (Henceforth, whenever we say that the verifier "expects" the proof to have a certain structure, the reader should interpret that statement in a similar way.)

Now we encapsulate the two properties above in the definition of the complexity class RPCP (the letters stand for **R**estricted PCP).

Definition 4 (RPCP$(r(n), q(n), s(n), t(n))$) Let r, q, s, t be functions defined on the positive integers. A language L is in RPCP$(r(n), q(n), s(n), t(n))$ if there is a verifier that checks membership proofs for L and on inputs of size n obeys the following constraints: (i) It uses $O(r(n))$ random bits. (ii) It uses an alphabet of size $2^{O(s(n))}$ (i.e., an alphabet whose each symbol requires $O(s(n))$ bits to represent). (iii) It makes $O(q(n))$ queries (each of which reads an alphabet symbol). (iv) It has a decision time of $O(t(n))$.

Note that RPCP$(r(n), q(n), s(n), t(n)) \subseteq$ PCP$(r(n), q(n) \cdot s(n))$. For all our verifiers, $r(n) = \log n$. The parameter that differs most dramatically among our verifiers is the number of *bits* read from the proof, which is $O(q(n) \cdot s(n))$. The composition lemma gives a technique to reduce this parameter, provided there exists a "reasonably" efficient normal form verifier.

Roughly speaking, a *normal form* verifier is a verifier with an associated encoding scheme, σ. The verifier is able to do the following kind of verification extremely efficiently for any $p \geq 1$:

> *Given:* A circuit C on k inputs and p codewords $\sigma(a_1), \ldots \sigma(a_p)$, where each $a_i \in \{0,1\}^{k/p}$.
> *To Check:* $C(\ a_1 \circ a_2 \circ \cdots \circ a_p\ ) \ = \ $accept.

Suppose there is some "reasonably" efficient normal form verifier V_2. Now suppose $L \in$ RPCP$(r(n), q(n), s(n), t(n))$ and let V be a verifier for L. We indicate how to use V_2 to reduce the parameter $q(n)s(n)$. Let us fix an input for V, thus fixing the verifier's alphabet Σ, the number of queries Q, and the decision time, T. For any random string r, the verifier's decision to accept or reject a provided proof Π is based upon the contents of only Q locations in Π. Furthermore, this decision is arrived in time T. Thus by using a standard transformation from a time T computation to a size $O(T^2)$ size circuit [Pap94], we can think of the third stage as being represented by a circuit C_r of size $O(T^2)$ (see Figure 2) whose input[3] is a bit string of size $Q \cdot \lceil \log \Sigma \rceil$. The verifier accepts proof Π using r as a random

[3]Strictly speaking, the input to the decision circuit also includes upto T bits that the verifier may have written, ahead of time, on its work-tape. However, these bits can be "hardwired" into the circuit.

string iff

$$C_r\Big(\ \Pi[i_1(r)] \circ \Pi[i_2(r)] \circ \cdots \circ \Pi[i_Q(r)]\ \Big) \qquad = \qquad \text{accept}, \tag{1}$$

where $(i_1(r), \ldots, i_Q(r))$ is the Q-tuple of queries made by the verifier using r as a random string[4], $\Pi[j]$ = the contents of the jth location in proof Π (we are thinking of $\Pi[j]$ as a bit string) and $\circ$ denotes string concatenation. We will refer to this circuit C_r, which represents the verifier's third stage, as the *decision circuit.* We emphasize again that C_r is very small compared to the input size n.

The key idea in verifier composition is to eliminate the need for V to read the strings $\Pi[i_1(r)], \ldots, \Pi[i_Q(r)]$ in their entirety. Instead V expects the proof string to have a slightly different format: each symbol of Π is not present in "plaintext" but is present in an encoded form using V_2's encoding scheme σ. In other words, the proof is $\sigma(\Pi[1]), \sigma(\Pi[2]), \cdots$. Now, checking whether condition (1) holds is easy: our verifier V has random access to $\sigma(\Pi[i_1(r)]), \ldots, \sigma(\Pi[i_Q(r)])$, so it can just use the program of the normal-form verifier V_2 to do this check. Potentially, this reduces the number of bits read from the proof. The input to V_2 is the decision circuit C_r, so the number of bits V_2 reads from the proof is a function of $|C_r|$, and $|C_r| \ll n$. (Of course, this rough sketch has not addressed potential problems such as: what happens to the verifier if the entries in the presented proof are not codewords but some arbitrary strings? It turns out that the normal-form verifier can detect this situation and reject. The formal definition below clarifies this.)

Now we give a formal definition of "normal form verifiers." We slightly deviate from the rough sketch in that we define this notion using 3CNF formulae instead of circuits. We also remark at this point that the "encoded inputs" idea of [BFLS91] was a simpler version of this definition (specifically, they used $p = 2$ and didn't characterize the verifier using as many parameters).

Definition 5 (Normal Form Verifiers.) Let functions r, s, q, t be defined on the positive integers. An $(r(n), s(n), q(n), t(n))$*-constrained normal-form verifier* is a verifier V with an associated encoding scheme $\sigma = (\Sigma_n, \mathcal{C}_n, \sigma_n)$ of some minimum distance $\delta_{\min}$.

Given any 3CNF formula φ with n variables and an integer p that divides n, the verifier has the following behavior.

a) It can check whether a given p-part split-encoded assignment satisfies φ. The verifier is provided random access to a "proof-string" $z_1 \sharp \cdots \sharp z_p \sharp \pi$, where the z_i's and π are strings over the alphabet $\Sigma_{n/p}$ and $\sharp$ is a special symbol. The verifier's behavior falls into one of the following cases.

- If $z_1, \ldots, z_p$ are codewords such that each $\sigma^{-1}(z_i)$ is an $\frac{n}{p}$-bit string and $\sigma^{-1}(z_1) \circ \cdots \sigma^{-1}(z_p)$ is a satisfying assignment to φ, then there is a π such that

$$\Pr[\text{ verifier accepts } z_1 \sharp \cdots \sharp z_p \sharp \pi] = 1.$$

- If $\exists i: \ 1 \le i \le p$ such that z_i is not $\delta_{\min}/3$-close, then for all π,

$$\Pr[\text{ verifier accepts } z_1 \sharp \cdots \sharp z_p \sharp \pi] < \frac{1}{2}.$$

- If each z_i is $\delta_{\min}/3$-close, but $\sigma^{-1}(N(z_1)) \circ \cdots \circ \sigma^{-1}(N(z_p))$ is not a satisfying assignment, where $N(z_i)$ is the codeword nearest to z_i, then again for all π

$$\Pr[\text{ verifier accepts } z_1 \sharp \cdots \sharp z_p \sharp \pi] < \frac{1}{2}.$$

[4]The sequence of queries actually depends on both the input and the random string r (see the remark before Definition 2). However, in the current discussion the verifier's input has been fixed.

b) Can do a) while obeying certain resource constraints. While doing the check in part (a), the verifier obeys the following constraints. (i) It uses $O(r(n))$ random bits (ii) It uses an alphabet $\Sigma_{n/p}$ of size $2^{O(s(n))}$ (that is, an alphabet whose each symbol requires $O(s(n))$ bits to represent) (iii) It reads $O(p + q(n))$ symbols from the proof (iv) It has decision time $O(p \cdot t(n))$.

□

Remarks: (i) It should be surprising that the number of queries in part (b) is $O(p+q(n))$. As we increase p, the verifier reads an average of $O(1+q(n)/p)$ symbols from each of the $p+1$ parts of the proof. Naively, one would expect it to read $O(q(n))$ symbols per part, for a total of $O(p \cdot q(n))$ queries. (ii) If the verifier is checking a p-part split-encoded assignment and is observed to accept some proof with probability $> 1/2$, then we can conclude that the input boolean formula is satisfiable. We expand upon this remark in the following proposition.

Proposition 4 *If there exists an $(r(n), q(n), s(n), t(n))$-constrained normal-form verifier, then NP $\subseteq$ RPCP$(r(n), q(n), s(n), t(n))$.*

Proof: The main point is that a normal-form verifier can be used to check membership proofs for 3SAT (in the sense of Definition 1). Suppose V is an $(r(n), q(n), s(n), t(n))$-constrained verifier and σ is its encoding scheme. We use V to check 1-part split-encoded assignments. This means that if the given formula has a satisfying assignment s, then there exists a proof of the form $\sigma(s) \sharp \pi$ that the verifier accepts with probability 1. If the formula is not satisfiable, then no proof-string of the form $z \sharp \pi$ is accepted with probability more than $1/2$.

Furthermore, in checking such a proof the verifier reads only $O(q(n))$ symbols, where each symbol is represented by $s(n)$ bits. We conclude that 3SAT $\in$ RPCP$(r(n), q(n), s(n), t(n))$. □

Lemma 5 (Composition) [5]

Let r, q, s, t be any functions defined on the natural integers. Suppose there is a normal-form verifier V_2 that is $(r(n), s(n), q(n), t(n))$-constrained. Then for all functions R, Q, S, T,

$$\mathit{RPCP}(R(n), S(n), Q(n), T(n)) \quad \subseteq \quad \mathit{RPCP}(R(n) + r(\tau),\ s(\tau),\ Q(n) + q(\tau),\ Q(n)\, t(\tau)),$$

where τ is a shorthand for $O((T(n))^2)$.

Remark: (i) When we say τ is a shorthand for $O((T(n))^2)$, we mean for example that $s(\tau)$ should be interpreted as $s(O(T(n)^2)$. (ii) To understand the usefulness of the lemma, realize that we are showing how to use V_2 to convert a verifier that reads $O(S(n)Q(n))$ bits into a verifier that reads only $O(s(\tau) \cdot (q(\tau) + Q(n)))$ bits from the proof. Whenever we use the lemma, τ and $s(n)$ are $\ll n$. Thus the saving is potentially large. This will become clearer in our proof of Theorem 2 below.

The proof of the following theorem will take up Section 4.

Theorem 6 *Let h, m be positive integers such that $(h+1)^{m-1} < n \le (h+1)^m$ and $h \ge \log n$. Then there is a $(\log n, h \log(h), m, poly(h))$-constrained normal-form verifier.*

Note that in Theorem 6, $m \approx \log n / \log h < \log n \le h$. Now we prove Theorem 2.

Proof: (*of Theorem 2*) For any positive fraction ϵ, we show that 3SAT has a $(\log n, \log^{0.5+\epsilon} n)$-restricted verifier.

[5] This lemma is a fleshed-out version of the original lemma in [AS92]. The concept of a normal-form verifier was not made very explicit in that paper.

Let V_1 be the verifier whose existence is guaranteed in Theorem 6 when we choose $h+1 = 2^{\sqrt{\log n}}$, and $m = O(\sqrt{\log n})$. Then V_1 is $(\log n,\ 2^{O(\sqrt{\log n})},\ \sqrt{\log n},\ 2^{O(\sqrt{\log n})})$-constrained. Thus by Proposition 4,

$$\text{3SAT} \in \text{RPCP}(\ \log n,\ 2^{O(\sqrt{\log n})},\ \sqrt{\log n},\ 2^{O(\sqrt{\log n})}\ ). \tag{2}$$

Verifier V_1 uses $O(\log n)$ random bits, which is acceptable. But the number of bits it reads from the proof (even for $p = 1$) is $2^{O(\sqrt{\log n})} \cdot 2^{O(\sqrt{\log n})} = 2^{O(\sqrt{\log n})}$. First we use the existence of V_1 and apply the Composition Lemma to statement (2). Let $t(n) = 2^{O(\sqrt{\log n})}$ denote the decision time before composition. The various parameters of the resulting verifier are obtained as follows.

1. *Number of random bits:* Changes from $O(\log n)$ to

$$O(\log n + \log(t(n)^2)) \quad = \quad O\left(\log n + \sqrt{\log n}\right) \quad = \quad O(\log n).$$

2. *Logarithm of alphabet size:* Changes from $s(n) = O(\sqrt{\log n})$ to

$$O(s(t(n)^2)) \quad = \quad O\left(\sqrt{\log t(n)}\right) \quad = O(\log^{1/4} n).$$

3. *Number of queries:* Changes from $q(n) = O\left(\sqrt{\log n}\right)$ to

$$O(q(n) + q(t(n)^2)) \quad = \quad O\left(\sqrt{\log n} + \sqrt{\log t(n)}\right) \quad = \quad O(\sqrt{\log n}).$$

4. *Decision time:* Changes from $t(n) = 2^{O(\sqrt{\log n})}$ to

$$O(q(n) \cdot t(t(n)^2)) \quad = \quad O\left(\sqrt{\log n} \cdot 2^{O(\sqrt{\log t(n)})}\right) \quad = \quad 2^{O(\log^{1/4} n)}.$$

In other words,

$$\text{3SAT} \in \text{RPCP}(\ \log n,\ 2^{O(\log^{1/4} n)},\ \sqrt{\log n},\ 2^{O(\log^{1/4} n)}\ ). \tag{3}$$

Let V_2 be this new verifier for 3SAT . We again use the existence of V_1 to apply the Composition Lemma to statement (3). The reader can check that this gives

$$\text{3SAT} \in \text{RPCP}(\ \log n,\ 2^{O(\log^{1/8} n)},\ \sqrt{\log n},\ 2^{O(log^{1/8} n)}\ ) \tag{4}$$

Let V_3 be this newest verifier for 3SAT.

Continuing this way for $1+\lceil \log \frac{1}{\log \epsilon} \rceil$ steps, at each step applying the Composition Lemma to the verifier obtained in the previous step, we end up with

$$\text{3SAT} \in \text{RPCP}(\ \log n, 2^{O(\log^{\epsilon/2} n)}, \sqrt{\log n}, 2^{O(\log^{\epsilon/2} n)}\ ) \tag{5}$$

Denote this verifier by V.

Though getting better, our verifier is still reading $O(\sqrt{\log n} \cdot 2^{O(\log^{\epsilon/2} n)})$ bits from the proof. Furthermore, $O(1)$ applications of the Composition Lemma using verifier V_1 will not reduce the decision time (or the logarithm of the alphabet size) below $2^{O(\log^c n)}$ where $c > 0$ is some fixed constant. (We do not wish to invoke the Composition Lemma more than $O(1)$ times because it hid constant factors in its $O(\cdot)$ notation and those constant factors could grow very fast.) What we need instead is the existence of a normal form verifier whose decision time is $\log^3 n$, say, since $\log^3(2^{O(\log^c n)}) = O(\log^{3c} n)$, which is sublogarithmic if $c < 1/3$.

We return to Theorem 6, but choose $h = O(\log n)$ and $m = \log n / \log h = O(\log n / \log\log n)$. Let V_{savior} be the normal form verifier whose existence is thus guaranteed. Note that V_{savior} is $(\log n,\ \log n \log\log n,\ \log n/(\log\log n),\ \text{poly}(\log n))$-constrained.

We use the existence of verifier V_{savior} and apply the Composition Lemma on statement (5). This gives a verifier V_{final} that uses an alphabet of size $O(\log t_1(n) \log\log t_1(n))$, where $t_1(n) = 2^{O(\log^{\epsilon/2} n)}$ is the decision time of V. Thus the alphabet size is $\log^{\epsilon/2+o(1)} n$. Furthermore, the number of queries made by V_{final} is $O(q_1(n) + \log(t_1(n)^2)/\log\log t_1(n))$, where $q_1(n) = O(\sqrt{\log n})$ is the number of queries made by V. Thus the number of queries is $O(\sqrt{\log n})$. Similarly the decision time of V_{final} is $\text{poly}(\log(t_1(n)^2))$. we conclude that

$$3\text{SAT} \in \text{RPCP}(\ \log n,\ \log^{\epsilon/2+o(1)} n,\ \sqrt{\log n},\ \text{poly}(\log n)\ ) \tag{6}$$

Hence we have shown that $3\text{SAT} \in \text{PCP}(\log n, \log^{1/2+\epsilon/2+o(1)} n)$, and thus $3\text{SAT} \in \text{PCP}(\log n, \log^{1/2+\epsilon} n)$. This finishes the proof of Theorem 2.

$\Box$

Remark: If we only wish to show that $SAT \in \text{PCP}(\log n, o(\log n))$, we can just use the existence of V_{savior} and apply the Composition Lemma on statement (3). This shows $SAT \in \text{RPCP}(\ \log n, \log^{1/4} n, (\log n)^{1/2+o(1)}, \text{poly}(\log n))$.

Now we prove the Composition Lemma.

Proof:(*Composition Lemma*)

Let L be a language in RPCP($R(n), Q(n), S(n), T(n)$), and let V_1 be the corresponding verifier for it. We will assume that the probability $1/2$ in the definition of "checking membership proofs" has been replaced by $1/4$. (This can be achieved by repeating V_1's actions twice using independent random strings. This is allowable since $R(n), Q(n), S(n), T(n)$ were specified using $O(\cdot)$ notation anyway.) We will assume the same of verifier V_2.

Let x be an input of size n. Let R, S, Q, and T denote, respectively, the number of random bits, the logarithm of the alphabet size, the number of queries made, and the size of the decision circuit of V_1 on this input. (The hypothesis of the lemma implies that Q, R, S, T are $O(Q(n))$, $O(R(n))$, and $O((T(n))^2)$.)

For a random string $w \in \{0,1\}^R$, we denote by C_w the decision circuit computed by V_1 using random string w, and we denote by $(i_1(w), i_2(w), \ldots, i_Q(w))$ the Q-tuple of queries made by verifier V_1. Let $\Pi[j]$ denote the jth symbol of proof Π (we think of it below as a string of S bits). Then V_1 accepts proof Π using w as a random string iff

$$C_w(\ \Pi[i_1(w)] \circ \Pi[i_2(w)] \circ \cdots \circ \Pi[i_Q(w)]\ ) \quad = \quad \text{accept}. \tag{7}$$

The Cook-Levin theorem for circuits (see [Pap94] for a description) gives an effective way to change this circuit C_w into a 3SAT formula ψ_w of size $O(T)$ such that V_1 accepts using random string w iff

$$\exists y_w :\ \ \Pi[i_1(w)] \circ \Pi[i_2(w)] \circ \cdots \circ \Pi[i_Q(w)] \circ y_w \quad \text{satisfies } \psi_w. \tag{8}$$

Note that the y_w part corresponds to auxiliary variables used to transform a circuit into a 3CNF formula. We assume w.l.o.g. (by "padding" the formula with irrelevant variables) that each $\Pi[j]$ has the same number of bits as y_w.

To make our description cleaner, we assume from now on that Π contains an additional 2^R locations, one for each choice of the random string w. The wth location among these supposedly contains the string y_w. Further, we assume that verifier V_1, when using the random string w, makes a separate query to the wth location to read y_w. Let $i_{Q+1}(w)$ denote the address of this location. Thus V_1 accepts using the random string w iff

$$\Pi[i_1(w)] \circ \cdots \circ \Pi[i_Q(w)] \circ \Pi[i_{Q+1}(w)] \ \text{ satisfies } \psi_w \tag{9}$$

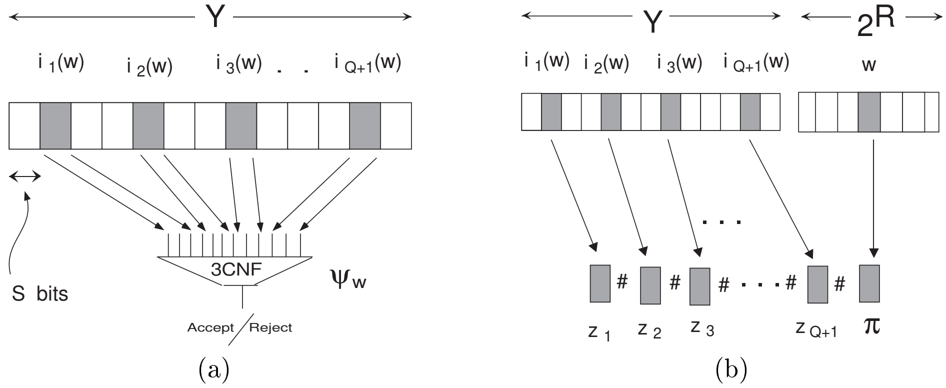

Figure 3: (a) Verifier V_1 expects a proof with Y symbols. Using $w \in \{0,1\}^R$ as a random string it queries locations $i_1(w), \ldots, i_{Q+1}(w)$. (b) Verifier V_{new} expects a proof with $Y + 2^R$ words. Using $w \in \{0,1\}^R$ it selects $Q+2$ words; these are shaded in the figure. These words are viewed as a $Q+1$-part split-encoded assignment $z_1 \sharp z_2 \sharp \cdots \sharp z_{Q+1} \sharp \pi$, which V_{new} checks using the normal-form verifier V_2.

After this change, let (see Figure 3(a))

$$Y = \text{Number of locations that } V_1 \text{ expects in a proof for input } x. \tag{10}$$

Now we describe the new verifier V_{new}. It will check membership proofs for input x by using the ability of V_2 to check $(Q+1)$-part split-encoded assignments to a formula of size $O(T)$ (this formula is ψ_w, the decision formula of V_1 for a particular random string). Let R', Q', S', T' respectively denote the four parameters describing V_2 in such a situation. Since V_2 is $(r(n), q(n), s(n), t(n))$-constrained, and checking a $(Q+1)$-part split-encoded assignment,

$$R' = O(r(T)), \;\; Q' = O(Q + q(T)), \;\; S' = s(T), \;\; T' = O(Q \cdot t(T)). \tag{11}$$

Let σ denote the encoding scheme of V_2 and suppose it encodes strings of size $O(T/(Q+1))$ by words of length Y' over an alphabet Σ' (where $\lceil \log \Sigma' \rceil = S'$).
Program of V_{new}: Verifier V_{new} expects the proof Π_{new} for x to contain $Y + 2^R$ words, where each word is over the alphabet Σ' and has length Y'. We denote by $\Pi_{new}[i]$ the ith word.

Step 1: V_{new} picks a random string $w \in \{0,1\}^R$. Then it simulates verifier V_1 on input x and random string w to generate the decision formula ψ_w and the queries $(i_1(w), \ldots, i_{Q+1}(w))$. Note that thus far V_{new} has not queried the proof.

Step 2: For $j = 1, \ldots, Q+1$ let us use the shorthand z_j for $\Pi_{new}[i_j(w)]$ and let π be shorthand for $\Pi_{new}[Y + w]$ (here $Y + w$ denote the sum of Y and the integer represented by w).

V_{new} picks a random string $w' \in \{0,1\}^{R'}$ and uses it to simulate the normal-form verifier V_2 on the input ψ_w and the $Q+1$-part split-encoded assignment

$$z_1 \sharp z_2 \sharp \cdots \sharp z_{Q+1} \sharp \pi$$

If V_2 accepts during this simulation, then V_{new} accepts. (Note: In simulating V_2, it is not necessary to read the words $z_1, \ldots, z_{Q+1}, \pi$ in entirety. Instead V_{new}, since it has random access to the proof, can just look at those symbols in $z_1, \ldots, \pi$ which V_2 decides to query using w' as a random string. Thus V_{new} reads only Q' symbols.)

Complexity of V_{new}: We analyze V_{new}'s efficiency. The verifier uses $R + R'$ random bits. The remaining parameters are the same as those of V_2 when it is given an input of size $O(T)$ and asked to check $Q + 1$-part split-encoded assignments. Thus the alphabet size is S', the number of queries is Q' and the decision time is T'. By examining (11), we see that the parameters of V_{new} are as claimed in the statement of the lemma.

Proof of Correctness: Now we prove that V_{new} probabilistically checks membership proofs for language L.

Case 1: $x \in L$.

In this case there is a proof Π with Y symbols such that

$$\Pr_{w \in \{0,1\}^R}[\text{verifier } V_1 \text{ accepts } \Pi \text{ using } w \text{ as random string}] = 1.$$

Construct as follows a proof Π_{new} that has $Y + 2^R$ words. First, encode each entry of Π with σ, the encoding scheme of V_2. These are the first Y words of Π_{new}. Then for each random string $w \in \{0,1\}^R$ add a new location (with the address $Y + w$) to the proof, and put in it a word π such that on input ψ_w

$$\Pr_{w' \in \{0,1\}^{R'}}[V_2 \text{ accepts } \sigma(\Pi[i_1(w)]) \sharp \cdots \sharp \sigma(\Pi[i_Q(w)]) \sharp \sigma(\Pi[i_{Q+1}(w)]) \sharp \pi \text{ using } w'] = 1.$$

(Such a word π exists because of condition (9) and the definition of "checking split-encoded assignments.") By construction,

$$\Pr_{w \in \{0,1\}^R, w' \in \{0,1\}^{R'}}[V_{new} \text{ accepts } \Pi_{new} \text{ using } (w, w') \text{ as random string}] = 1.$$

Case 2: $x \notin L$.

In this case, V_1 is known to accept every membership proof with probability less than $1/4$.

We show that V_{new} accepts every membership proof with probability less than $1/2$. Assume for contradiction's sake that there is a candidate proof Π_{new} with $Y + 2^R$ words such that

$$\Pr_{w \in \{0,1\}^R, w' \in \{0,1\}^{R'}}[V_{new} \text{ accepts } \Pi_{new} \text{ using } (w, w')] \geq \frac{1}{2}. \tag{12}$$

We construct a table Π with Y locations, in which the symbol in the jth location is $\Pi[j] = \sigma^{-1}(N(\Pi_{new}[j]))$, where $N(\Pi_{new}[j])$ denotes the codeword nearest to $\Pi_{new}[j]$ (note: if $\sigma^{-1}(N(\Pi_{new}[j]))$ is not defined, we use an arbitrary string of bits instead). Let

$$p \;=\; \Pr_{w \in \{0,1\}^R}[V_1 \text{ accepts } \Pi \text{ using } w \text{ as a random string}] < \frac{1}{4},$$

where the inequality is by the fact that $x \notin L$.

Note that if $w \in \{0,1\}^R$ is such that V_1 rejects Π using w as a random string, then by the definition of "checking split-encoded assignments,"

$$\Pr_{w' \in \{0,1\}^{R'}}[V_{new} \text{ accepts } \Pi_{new} \text{ using } (w, w')] < \frac{1}{4}.$$

Hence an upperbound on the probability with which V_{new} accepts Π_{new} is

$$p + (1 - p) \cdot \frac{1}{4} \leq \frac{1}{4} + \frac{3}{4}\frac{1}{4} < \frac{1}{2}.$$

This contradicts (12), and the proof for Case (ii) is finished.

□

Remark: By using a more careful analysis of our composition technique and Theorem 6, it is possible to show that NP $=$ PCP$(\log n, (\log\log n)^2)$. We omit this (very complicated) proof from this paper, since the result has been superseded by the result NP $=$ PCP$(\log n, 1)$ [ALM$^+$92] anyway.

4 Proof of Theorem 6

In this section we prove Theorem 6. The proof is largely based upon that in [BFLS91, FGL+91], with ingredients dating back to [LFKN92, BFL91]. The only new parts are our improved analysis of the low degree test, and our technique for showing that the verifier is in normal form (in particular, that it can check split-encoded assignments).

Underlying the description of the verifier is an algebraic representation of a 3SAT formula. The representation uses a simple fact: every assignment can be encoded as a multivariate polynomial that takes values in a finite field (see Section 4.1). A polynomial that encodes a satisfying assignment is called a *satisfying polynomial.* Just as a satisfying boolean assignment can be recognized by checking whether or not it makes all the clauses of the 3SAT formula true, a satisfying polynomial can be recognized by checking — using some special algebraic procedures described below— whether it satisfies some set of equations involving the operations $+$ and $\cdot$ of the finite field.

Section 4.1 describes the encoding of assignments with polynomials, and defines terms used in the rest of the paper. Section 4.2 contains a description of the main ideas used to construct the verifier, including an algebraic representation of 3SAT. Theorem 6 is proved in two parts in Sections 4.3 and 4.4. This proof uses certain algebraic procedures, whose detailed descriptions and analyses appear in Sections 4.5 and 4.6. Some results in Sections 4.4 and 4.6 use algebraic results on polynomials that are proved later in Section 5.

4.1 Polynomial Codes and their use

Let F be the finite field GF(q) and k, d be positive integers. A *k-variate polynomial of degree d over F* is a sum of terms of the form $ax_1^{j_1}x_2^{j_2}\cdots x_k^{j_k}$ where $a \in \mathrm{F}$ and each of the integers $j_1, \ldots, j_k$ is at most d. Let $\mathrm{F}_d[x_1, \ldots, x_k]$ be the set of functions from F^k to F that can be described by a polynomial of degree d. [6]

We will be interested in representations of polynomials *by value.* A k-variate polynomial defines a function from F^k to F, so it can be expressed by $|\mathrm{F}|^k = q^k$ values. In this representation a k-variate polynomial (or any function from F^k to F for that matter) is a word of length q^k over the alphabet F.

Definition 6 *The code of k-variate polynomials of degree d* (or just *polynomial code* when k, d are understood from context) is the code $\mathrm{F}_d[x_1, \ldots, x_k]$ in F^{q^k}.

Now we can define *distance* between two words in F^{q^k} and terms such as δ-close, just as in Section 2.1.

Definition 7 The *distance* between two functions $f, g\colon \mathrm{F}^k \to \mathrm{F}$ is the fraction of points in F^k they disagree on.

The *distance* of a function $f\colon \mathrm{F}^k \to \mathrm{F}$ *to the polynomial code* $\mathrm{F}_d[x_1, \ldots, x_k]$, denoted $\Delta_d(f)$, is the distance of f to the polynomial in $\mathrm{F}_d[x_1, \ldots, x_k]$ that is closest to to it. If $\Delta_d(f) < \delta$, we say f is *δ-close to $F_d[x_1, \ldots, x_k]$* (or just *δ-close* when the degree d can be inferred from the context).

Now we observe (for a proof see Fact 22 in the Appendix) that the polynomial code has large minimum distance.

Fact 7 (Schwartz) *Two distinct polynomials in $F_d[x_1, \ldots, x_k]$ disagree on at least $1-dk/q$ fraction of points in F^k.* $\Box$

[6] The use of F_d above should not be confused with the practice in some algebra texts of using F_q as a shorthand for GF(q).

Wherever this paper uses polynomial codes, $dk < q/2$. Thus if $f\colon \mathrm{F}^k \to \mathrm{F}$ is δ-close for $\delta < 1/4$, then the polynomial in $\mathrm{F}_d[x_1, \ldots, x_k]$ that agrees with f in at least $1-\delta$ fraction of the points is unique. (In fact, no other polynomial describes f in more than even $\delta + kd/q$ fraction of the points.)

Definition 8 If $f : \mathrm{F}^k \to \mathrm{F}$ is a δ-close function where $\delta < 1/4$, then the symbol $\widetilde{f}$ denotes the (unique) polynomial nearest to it.

Polynomials are useful to us as encoding objects. We define below a canonical way (due to [BFLS91]) to encode a sequence of bits with a polynomial. For convenience, we describe a more general method that encodes a sequence of field elements with a polynomial. Encoding a sequence of bits is a sub-case of this method, since $0, 1 \in \mathrm{F}$.

Theorem 8 *Let h be an integer such that set of integers $[0, h]$ is a subset of field F. For every function $s : [0, h]^m \to F$, there is a unique function $\widehat{s} \in F_h[x_1, \ldots, x_m]$ such that $s(y) = \widehat{s}(y)$ for all $y \in [0, h]^m$.*

Remark: Readers uncomfortable with thinking of h as both the degree of a polynomial (that is, an integer) and as a field element should think of $[0, h]$ as any subset of the field F that has size $h + 1$.

Proof: We only prove the existence of $\widehat{s}$; the reader can verify uniqueness from our construction.

For $\overline{u} = (u_1, \ldots, u_m) \in [0, h]^m$, let $L_{\overline{u}}$ be the polynomial defined as

$$L_{\overline{u}}(x_1, \ldots, x_m) = \prod_{i=1}^{m} l_{u_i}(x_i),$$

where l_{u_i} is the unique degree-h polynomial in x_i that is 1 at $x_i = u_i$ and 0 at $x_i \in [0, h]\backslash\{u_i\}$. (That $l_{u_i}(x_i)$ exists follows from Fact 21 in the Appendix.) Note that the value of $L_{\overline{u}}$ is 1 at $\overline{u}$ and 0 at all the other points in $[0, h]^m$. Also, its degree is h.

Now define the polynomial $\widehat{s}$ as

$$\widehat{s}(x_1, \ldots, x_m) = \sum_{\overline{u} \in [0,h]^m} s(\overline{u}) \cdot L_{\overline{u}}(x_1, \ldots, x_m).$$

□

Example 1 Let $m = 2, h = 1$. Given any function $f : [0, 1]^2 \to \mathrm{F}$, we can map it to a bivariate degree 1 polynomial, $\widehat{f}$, as follows.

$$\begin{aligned} \widehat{f}(x_1, x_2) &= (1 - x_1)(1 - x_2)f(0, 0) + x_1(1 - x_2)f(1, 0) \\ &\quad +(1 - x_1)x_2 f(0, 1) + x_1 x_2 f(1, 1). \end{aligned}$$

Definition 9 Let h be an integer such that $[0, h] \subseteq \mathrm{F}$. For a function $s : [0, h]^m \to \mathrm{F}$, the *polynomial extension of* s is the polynomial $\widehat{s} \in \mathrm{F}_h[x_1, \ldots, x_m]$ defined in Theorem 8.

The encoding. We define a method to encode sequences of field elements with polynomials. Since $0, 1 \in \mathrm{F}$, the method can also be used to encode bit strings. Let h be an integer such that $[0, h] \subseteq \mathrm{F}$, and l an integer such that $l = (h + 1)^m$ for some integer m. Define a one-to-one map from F^l to $\mathrm{F}_h[x_1, \ldots, x_m]$ (in other words, from sequences of l field elements to polynomials in $\mathrm{F}_h[x_1, \ldots, x_m]$) as follows. Identify in some canonical way the set of integers $\{1, \ldots, l\}$ and the set $[0, h]^m \subseteq \mathrm{F}^m$. (For instance, identify the integer $i \in \{1, \ldots, l\}$ with its m-digit representation in base $h + 1$.) Thus a sequence s of l field elements may be viewed as a function s from $[0, h]^m$ to F. Map the sequence s to the polynomial extension $\widehat{s}$

of this function. This map is one-to-one because if polynomials $\widehat{f}$ and $\widehat{g}$ are the same, then they agree everywhere and, in particular, on $[0,h]^m$, which implies $f = g$.

The inverse map of the above encoding is obvious. A polynomial $f \in F_h[x_1, \ldots, x_m]$ is the polynomial extension of the function $r : [0,h]^m \to F$ defined as $r(x) = f(x), \forall x \in [0,h]^m$.

Note that we are encoding sequences of length $l = (h+1)^m$ by sequences of length $|F|^m = q^m$. Whenever we use this encoding scheme, this increase in size is not too much. The applications depend upon some algebraic procedures to work correctly, for which it suffices to take $q = \mathrm{poly}(h)$. Then q^m is $h^{O(m)} = \mathrm{poly}(l)$. Hence the increase in size is polynomially bounded.

4.1.1 Restrictions of Polynomials: Some Definitions

We define some more terms that will be useful later. For a function $f: F^m \to F$ and a subset $S \subseteq F^m$, the *restriction of f on S* is the function from S to F whose value at any point $u \in S$ is $f(u)$. We will be interested in restrictions on very special subsets of F^m: those obtained by fixing some of the coordinates.

Definition 10 For a function $f: F^m \to F$ and field-element $a \in F$, the *restriction of f obtained by fixing $x_1 = a$* is the function $f|_{x_1=a}: F^{m-1} \to F$ defined as

$$f|_{x_1=a}(x_2, \ldots, x_m) = f(a, x_2, \ldots, x_m) \quad \forall (x_2, \ldots, x_m) \in F^{m-1}. \tag{13}$$

We likewise define, for any $l < m$, any point $(a_1, \ldots, a_l) \in F^l$, and any sequence of indices $i_1, \ldots, i_l \le m$, the restriction $f|_{(x_{i_1}, \ldots, x_{i_l}) = \vec{a}}$.

Note that if f is a degree-h polynomial, then so are all its restrictions defined in Definition 10.

4.2 Description of the Verifier: Preliminaries

We present some of the main ideas in the design of this verifier.

Recall that φ denotes the instance of 3SAT given to the verifier. Throughout this section, we let n denote both the number of clauses and the number of variables in φ. (We defend the use of n for both quantities on the grounds that they can be made equal: just add redundant variables, which don't appear in any clauses, to the formula.) Also, h, m are the integers appearing in the hypothesis of Theorem 6. In fact, we assume — by adding some irrelevant variables and clauses to φ — that $n = (h+1)^m$. Since $h > \log n$, we have $m = \log n / \log(h+1) < h$. Finally, F denotes a finite field with $\Theta(h^3 m^2)$ elements[7]. Since $m < h$, this field size is $O(h^5)$.

Now we give an overview of the verifier's program. It uses the fact (see Definition 9) that every assignment of φ, since it is a string of $n = (h+1)^m$ bits, can be encoded by its polynomial extension.

Definition 11 A polynomial in $F_h[x_1, \ldots, x_m]$ is a *satisfying polynomial* for the 3CNF formula φ if it is the polynomial extension of a satisfying assignment for φ.

Definition 12 For a function $g: F^k \to F$ and a set $S \subseteq F^k$, *the sum of g on S* is the value $\sum_{x \in S} g(x)$.

The verifier expects the proof to contain a satisfying polynomial $f: F^m \to F$. (Note that such a function is represented by $|F|^m = (h^3 m^2)^m = O((h^5)^m) = O(n^5)$ values.) The verifier uses two algebraic procedures to check that f is a satisfying polynomial. The first, called the low degree test (Procedure 2), probabilistically examines the proof in a few places, and

[7]Most of our Lemmas work with smaller field sizes. But Lemma 16 requires the field to be this large.

rejects with high probability if the function f is not 0.01-close. So assume for argument's sake that f is indeed 0.01-close. Next, the verifier tries to decide whether $\widetilde{f}$, the polynomial nearest to f, is a satisfying polynomial. Here Lemma 9 is useful: it gives a probabilistic method to construct a polynomial $P^{\widetilde{f}}$ such that the following holds. If $\widetilde{f}$ is a satisfying polynomial, then $P^{\widetilde{f}}$ sums to 0 (in the sense of Definition 12) on a certain fixed subset $S \subseteq \mathrm{F}^{4m}$. But if $\widetilde{f}$ is not a satisfying polynomial, then with high probability, $P^{\widetilde{f}}$ does not sum to 0 on S. Now the verifier can use a simple algebraic procedure from [LFKN92], called the Sum-Check (Procedure 1) to verify that $P^{\widetilde{f}}$ indeed sums to 0 on S.

Further details are provided below. Section 4.2.1 describes the algebraic conditions that a satisfying polynomial must obey. Section 4.2.2 describes some algebraic procedures that the verifier will use. The proof of Theorem 6 is split in two parts, which are proved in Sections 4.3 and Section 4.4.

4.2.1 Algebraic Representation of 3SAT

In Lemma 9, we give an algebraic characterization of satisfying polynomials. This lemma is similar in spirit to a lemma in [BFL91], although the precise formulation given here is due to [BFLS91].

Lemma 9 (Algebraic View of 3SAT) *Given $A \in F_h[x_1, \ldots, x_m]$, there is a polynomial-time constructible sequence of poly(n) polynomials $P_1^A, P_2^A, \ldots \in F_{7h}[x_1, \ldots, x_{4m}]$ such that*

1. *If A is a satisfying polynomial for φ, then the sum of each P_i^A on $[0,h]^{4m}$ is 0. But if A is not a satisfying polynomial, this sum is 0 for at most $1/8$th of the P_i^A's.*
2. *For each point $w \in F^{4m}$, there are three points $w_1, w_2, w_3 \in F^m$ such that computing the value of each P_i^A at w requires only the values of A at w_1, w_2 and w_3, and moreover, this computation requires only poly($mh \log F$) time. Furthermore, if w is uniformly distributed in F^{4m}, then each w_i is uniformly distributed in F^m.*

Proof: Since $(h+1)^m = n$, we can identify the cube $[0,h]^m$ with the set of integers $\{1, \ldots, n\}$. By definition, polynomial A is a satisfying polynomial iff the sequence of values $(A(v) : v \in [0,h]^m)$ represents a satisfying assignment.

For $j = 1,2,3$, let $\chi_j(c,v)$ be the function from $[0,h]^m \times [0,h]^m$ to $\{0,1\}$ such that $\chi_j(c,v) = 1$ if v is the j^{th} variable in clause c, and 0 otherwise. Similarly let $s_j(c)$ be a function from $[0,h]^m$ to $\{0,1\}$ such that $s_j(c) = 1$ if the j^{th} variable of clause c is unnegated, and 0 otherwise. Since the OR of three boolean variables is 1 iff at least one of them is 1, it follows that A is a satisfying polynomial iff for every clause $c \in [0,h]^m$ and every triple of variables $v_1, v_2, v_3 \in [0,h]^m$, we have

$$\prod_{j=1}^{3} \chi_j(c, v_j) \cdot (s_j(c) - A(v_j)) = 0, \tag{14}$$

that is to say, iff

$$\prod_{j=1}^{3} \widehat{\chi_j}(c, v_j) \cdot (\widehat{s_j}(c) - A(v_j)) = 0, \tag{15}$$

where in the previous condition we have replaced functions χ_j and s_j appearing in condition (14) by their degree-h polynomial extensions, $\widehat{\chi_j}\colon \mathrm{F}^{2m} \to \mathrm{F}$ and $\widehat{s_j}\colon \mathrm{F}^m \to \mathrm{F}$ respectively. Conditions (15) and (14) are equivalent because, by definition, the polynomial extension of a function takes the same values on the underlying cube (which is $[0,h]^m$ for s_j and $[0,h]^{2m}$ for χ_j) as the function itself.

Define a polynomial $g_A\colon \mathrm{F}^{4m} \to \mathrm{F}$ as

$$g_A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}) = \prod_{j=1}^{3} \widehat{\chi_j}(\overline{z}, \overline{w_j}) \cdot (\widehat{s_j}(\overline{z}) - A(\overline{w_j})) \tag{16}$$

where each of $\overline{z}, \overline{w_1}, \overline{w_2}$, and $\overline{w_3}$ takes values in F^m. Since each $\widehat{\chi_j}$ and $\widehat{s_j}$ has degree h, and so does A, the degree of g_A is $3 \cdot 2 \cdot h = 6h$.

Then we may restate condition (15) as: A is a satisfying polynomial iff

$$g_A \text{ is } 0 \text{ at every point of } [0,h]^{4m}. \tag{17}$$

Lemma 10 (below) asserts that condition (17) is equivalent to requiring, for every R_i in a fixed set of degree h polynomials called the the "zero-testers," that the following condition holds.

$$R_i \cdot g_A \text{ sums to } 0 \text{ on } [0,h]^{4m}. \tag{18}$$

Further, Lemma 10 implies that if the condition in (17) is false, then condition (18) is false for at least $7/8$ of the "zero-tester" polynomials.

Now define the desired family of polynomials $\{P_1^A, P_2^A, \ldots,\}$ by

$$P_i^A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}) = R_i(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3})\ g_A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}),$$

where R_i is the ith "zero-tester" polynomial. Note that P_i^A is a polynomial of degree $7h$. Also, evaluating P_i^A at a randomly-chosen point in F^m requires the value of g_A at that point, which requires (as is clear from inspecting Equation (16)) the value of A at three random points in F^m.

Constructibility. The construction of the polynomial extension in the proof of Theorem 8 is effective. We conclude that the functions $\widehat{\chi_j}, \widehat{s_j}$ can be constructed in $\mathrm{poly}(n)$ time. The family of Lemma 10 is likewise constuctible in $\mathrm{poly}(q^m) = \mathrm{poly}(n)$ time. Having constructed all the above, computing the value of a function P_i^A at a point $w \in \mathrm{F}^{4m}$ is very fast, and takes $\mathrm{poly}(hm \log \mathrm{F})$ time. This is because we only need to compute a value of g_A, which, by inspecting (16), requires three values of A and $O(1)$ field operations.

Thus, assuming Lemma 10, Lemma 9 has been proved.

□

The following lemma concerns a family of polynomials that is useful for testing whether or not a function is identically zero on the cube $[0,h]^j$ for any integers h, j. We give its proof in the Appendix.

Lemma 10 *["Zero-tester" Polynomials, [BFLS91, FGL⁺91]] Let $F = GF(q)$ and integers m, h satisfy $32mh < q$. Then there exists a family of q^{4m} polynomials $\{R_1, R_2, \ldots\}$ in $F_h[x_1, \ldots, x_{4m}]$ such that if $f : [0,h]^{4m} \to F$ is any function not identically 0, then if R is chosen randomly from this family,*

$$\Pr[\sum_{y \in [0,h]^{4m}} R(y) f(y) = 0] \leq \frac{1}{8}. \tag{19}$$

This family is constructible in $q^{O(m)}$ time.

4.2.2 The Algebraic Procedures

Now we give a "black-box" description of the the Sum-Check and the Low Degree test. Note that both procedures require, in addition to the polynomial in question, a table with $|\mathrm{F}|^{O(m)} = \mathrm{poly}(n)$ entries. Also, their randomness requirement is $O(\log |\mathrm{F}|^m)$ bits, which is $O(\log n)$ for us. Finally, the procedures' queries to the provided tables are *nonadaptive*:

they depend only upon the random string and not upon the bits already inspected in the tables.

Sections 4.5 and 4.6 provide further details on the procedures and establish the desired properties and complexities.

Procedure 1 (Sum-Check) Let integers d, l and field $F = \mathrm{GF}(q)$ satisfy $4dl < q$.
Given: Polynomial $B \in \mathrm{F}_d[y_1, \ldots, y_l]$, a subset $H \subseteq \mathrm{F}$, a value $c \in \mathrm{F}$, and a table T whose each entry is a string of $O(d \log q)$ bits.
Properties of the procedure: If the sum of B on H^l is not c, the procedure rejects with probability at least $1 - dl/q$ irrespective of the contents of table T. But if the sum is c, then there is a table T such that the procedure accepts with probability 1.
Complexity. The procedure uses the value of B at one random point in F^l and reads another $O(l)$ entries from T. It makes these queries nonadaptively. It uses $\log(|\mathrm{F}|^l)$ random bits and runs in time $\mathrm{poly}(l + d + |H| + \log |\mathrm{F}|)$.

Procedure 2 (Low Degree Test) Let $F = \mathrm{GF}(q)$ and d, l be integers satisfying $100d^3l^2 < q$.
Given: $f\colon \mathrm{F}^l \to \mathrm{F}$, a number $\delta < 0.01$, and a table T whose each entry is a string of $O(d \log q)$ bits.
Properties of the procedure: If $f \in \mathrm{F}_d[y_1, \ldots, y_l]$, then there is a table T such that the procedure accepts with probability 1. If f is not δ-close to $F_d[y_1, \ldots, y_l]$, then the procedure rejects with probability at least 3/4 irrespective of the contents of table T.
Complexity. The procedure uses the value of f at $O(1/\delta)$ points in F^l and reads another $O(l/\delta)$ entries from T. The queries are nonadaptive. It uses $O(\log(|\mathrm{F}|^l)/\delta)$ random bits and runs in time $\mathrm{poly}(l + d + \log |\mathrm{F}|)$.

4.3 Proof of Theorem 6, part (a)

Now we prove Theorem 6. For convenience, we prove it in two parts, (a) and (b). In this section we prove part (a), where we prove that the verifier can check membership proofs for 3SAT. In part (b) (in Section 4.4) we prove that the verifier can check split-encoded assignments (in the sense of Definition 5), and so is in normal form.

Proof: (*Of Theorem 6; part (a) of the proof*)

Our verifier expects the proof to contain a function $f\colon \mathrm{F}^m \to \mathrm{F}$, and a set of tables that allow it to perform the following two steps.

First, the verifier does a low degree test to check that f is 0.01-close. Note that the proof has to contain a table that allows the verifier to do this test. Further, if f is not 0.01-close, the test will reject with probability at least 3/4, regardless of this table's contents. (Recall that the definition only asks that the verifier reject with probability at least 1/2. But part (b) will need this probability to be at least 3/4.) So assume for argument's sake that f is indeed 0.01-close.

Next, the verifier uses $O(\log n)$ random bits to select a polynomial $P_i^{\widetilde{f}}$ uniformly at random from the family described in Lemma 9. It uses the Sum-Check (Procedure 1) to check that $P_i^{\widetilde{f}}$ sums to 0 on $[0, h]^{4m}$. Note that the proof has to contain a sequence of tables, one for each polynomial in the above family, that allows a Sum-Check to be performed on that polynomial. Out of this sequence of tables, the verifier merely uses the one corresponding to the polynomial it actually picks.

The Sum-Check requires the value of the selected polynomial $P_i^{\widetilde{f}}$ at one random point, which, by the statement of Lemma 9, requires values of $\widetilde{f}$ at 3 random points. Getting these may seem like a problem, since the verifier has a table for f and not for $\widetilde{f}$. Luckily, f and $\widetilde{f}$

differ in at most 0.01 fraction of points, and the verifier only needs the values of $\widetilde{f}$ at three random points. So the verifier just uses values of f for the Sum-Check, and hopes for the best. Indeed, the probability that these are also the values of $\widetilde{f}$ is at least $1-3\times 0.01 = 0.97$.

Finally, the verifier accepts iff neither the low degree test nor the Sum-Check fails.

Correctness: Suppose φ is satisfiable. The verifier clearly accepts with probability 1 any proof that contains the polynomial extension of a satisfying assignment, and the proper tables required by the various procedures.

Now suppose φ is not satisfiable. If f is not 0.01-close, the low degree test accepts with probability at most $1/4$. So assume, w.l.o.g., that f is 0.01-close. Then the verifier can accept only if one of the three events happens. (i) The selected polynomial $P_i^{\widetilde{f}}$ sums to 0 on $[0,h]^{4m}$. By Lemma 9 this event can happen with probability at most $1/8$. (ii) $P_i^{\widetilde{f}}$ does not sum to 0, but the Sum-Check fails to detect this. The probability of this event is upperbounded by the error probability of the Sum-Check, which is $O(mh/q)$. (iii) At one of the three points at which the Sum-Check requires the value of $\widetilde{f}$, the functions $\widetilde{f}$ and f disagree. It is not clear that the Sum-Check fails in this case, but even if does, the probability of this event is upperbounded by $3\times 0.01 \le 0.03$.

To sum up, if φ is not satisfiable, the probability that the verifier accepts is at most $1/8 + 0.03 + O(mh/q)$, which is less than $1/4$ since $q = \Theta(h^3m^2)$.

Nonadaptiveness: Although it may not be clear from the above description, the verifier can query the proof nonadaptively. The reason is that the Sum-Check does not need any results from the low degree test, so the verifier can read, in one go, all the information required for both the tests. (Of course, the *correctness* of the Sum-Check step cannot be guaranteed if f is not 0.01-close, but in that case the low degree test rejects with high probability anyway.) Now, since the Sum-Check and the low degree test are nonadaptive procedures as well, we conclude that the verifier is nonadaptive.

Complexity: Recall that we have to show that the verifier is $(\log n, h\log h, m, \mathrm{poly}(h))$-constrained. By inspecting the complexities of the Sum-Check and the low-degree test we see that the verifier needs only $O(\log |\mathrm{F}|^{4m}) = O(\log n)$ random bits for its operation. The tables in the proof contain entries of size $s = O(h\log|\mathrm{F}|) = O(h\log h)$. We think of each entry as a letter in a alphabet of size 2^s. By examining the complexities of the Sum-Check and the low degree test, we see that the verifier only examines $O(m)$ of these entries.

In order to make the decision time $\mathrm{poly}(h)$, the the verifier has to do things in a certain order. In its first stage it reads the input, selects the above-mentioned polynomial $P_i^{\widetilde{f}}$ and carries out all the steps in the construction of $P_i^{\widetilde{f}}$ (as described in the proof of Lemma 9) except the part that actually involves reading values of $\widetilde{f}$. All this takes $\mathrm{poly}(n)$ time, and does not involve reading the proof. The rest of the verification requires reading the proof, and consists of the the low degree test and the Sum-Check, and the evaluation of $P_i^{\widetilde{f}}$ at one point (by reading three values of f). All these procedures require time $\mathrm{poly}(h+m+\log|\mathrm{F}|) = \mathrm{poly}(h)$.

To finish our claim that the verifier is in normal form, we have to show that it can check split-encoded assignments. We do this separately in Section 4.4. $\square$

4.4 Proof of Theorem 6: Split-encoded Assignments

This is part (b) of the proof of Theorem 6. We show how to modify the verifier in part (a) (Section 4.3) can check split-encoded assignments consisting of p parts for any positive integer p.

Recall (Definition 5) that in this setting the verifier defines an encoding method σ, and expects the proof string to be of the form $\sigma(a_1)\circ\cdots\circ\sigma(a_p)\circ\pi$, where π is some information that allows an efficient check that $a_1\circ\cdots\circ a_p$ is a satisfying assignment to φ ($\circ =$

concatenation of strings). Recall that we assume that each a_i includes the same number of variables, namely, n/p.

Assume, as in part (a), that $n = (h+1)^m$ where h, m are the same integers as in the statement of Theorem 6. Assume further that p is a power of $(h+1)$, so $n/p = (h+1)^l$ for some integer l. This last assumption is without loss of generality, since the verifier can use the usual trick of adding irrelevant variables to the formula; the proof string then contains only the assignments to the original variables, and the verifier supplies some trivial values for the irrelevant variables.

Since n and n/p are powers of $(h+1)$, we can encode bits strings in $\{0,1\}^n$ and $\{0,1\}^{n/p}$ by their degree-h polynomial extensions in $\mathrm{F}_h[x_, \ldots, x_m]$ and $\mathrm{F}_h[x_, \ldots, x_l]$ respectively. Let σ_1 and σ denote these encodings; i.e., $\sigma_1 : \{0,1\}^n \to \mathrm{F}_h[x, \ldots, x_m]$ and $\sigma : \{0,1\}^{n/p} \to \mathrm{F}_h[x_, \ldots, x_l]$.

Now we describe how the verifier checks split-encoded assignments. It expects the proof to contain a function $f: \mathrm{F}^m \to \mathrm{F}$, along with tables that allow a quick check — as in part (a) — that f is the polynomial extension of some satisfying assignment. The verifier also expects the proof to contain p functions $f_1, \ldots, f_p: \mathrm{F}^l \to \mathrm{F}$ that are, supposedly, $\sigma(a_1), \ldots, \sigma(a_p)$, for some bit strings $a_1, \ldots, a_p$. Furthermore, the polynomials f and $f_1, \ldots, f_p$ are supposed to satisfy:

$$\sigma_1^{-1}(f) = \sigma^{-1}(f_1) \circ \cdots \circ \sigma^{-1}(f_p). \tag{20}$$

Checking such a proof-string involves two steps. In the first step, the verifier checks that the provided function f is 0.01-close, and that $\widetilde{f}$ is the polynomial extension of a satisfying assignment. If not, the verifier accepts with probability less than 1/4. (This step is the same as in part (a).) In the second step, the verifier checks (using the Procedure in Figure 4.4) that the functions $f, f_1, \ldots, f_p$ satisfy the *concatenation property*, which is defined below. If the functions do not satisfy this property, then this part accepts with probability less than 1/4.

This finishes our description of how the verifier, given a p-part split-encoded assignment, checks that it represents a satisfying assignment.

Definition 13 Let σ and σ_1 be as defined above. Let $f: \mathrm{F}^m \to \mathrm{F}$ be 0.02-close and $f_1, \ldots, f_p$ be functions from F^l to F. Then $f, f_1, \ldots, f_p$ satisfy the *concatenation property* if

$$\text{each } f_i \text{ is 0.03-close} \tag{21}$$

and

$$\sigma_1^{-1}(\widetilde{f}) = \sigma^{-1}(\widetilde{f_1}) \circ \sigma^{-1}(\widetilde{f_2}) \circ \cdots \circ \sigma^{-1}(\widetilde{f_p}). \tag{22}$$

Thus to finish the description of this verifier, we describe how to check the concatenation property by reading $O(1)$ values of each of $f, f_1, \ldots, f_p$ and using $O(\log n)$ random bits. See Figure 4.4.

Complexity: The test can query the tables for $f, f_1, \ldots, f_p$ nonadaptively, since it can construct L_i ahead of time, and then perform all 1000 steps simultaneously. Furthermore, the test uses $O(m \log |\mathrm{F}|)$ random bits, which is $O(\log n)$ in our context, and examines 1000 values of each of $f, f_1, \ldots, f_p$.

Correctness of the Procedure: First we note that if all the functions are degree h polynomials and satisfy Condition (20), then the procedure accepts with probability 1. To see this, note that by definition, $\sigma_1^{-1}(f)$ is the sequence of values of f on $[0,h]^m$ and each $\sigma^{-1}(f_i)$ is the sequence of values of f_i on $[0,h]^l$. Further, i ranges over $\{1, \ldots, p\} = [0,h]^{m-l}$, so Condition (20) holds iff

$$f(i, u) = f_i(u) \quad \forall\, i \in [0,h]^{m-l}, u \in [0,h]^l. \tag{23}$$

The Procedure:

Let integers h, m, l be the same as in above paragraphs and $\mathrm{F} = GF(q)$ where $q > \Omega(m^2h^3)$.
The procedure can query the following tables: A 0.01-close function $f : \mathrm{F}^m \to \mathrm{F}$ and p functions $f_1, \ldots, f_p : \mathrm{F}^l \to \mathrm{F}$

Identify the elements of $\{1, \ldots, p\}$ *and* $[0, h]^{m-l}$ *in a one-to-one fashion.*
For $i \in [0, h]^{m-l}$, *let* L_i: $F^{m-l} \to F$ *be the degree-h polynomial extension of a function that is* 1 *at* i *and* 0 *at* $[0, h]^{m-l} \setminus \{i\}$.

Pick 1000 points uniformly at random in F^m.
Accept iff for all 1000 of them, the following is true.
If the point is $(a_1, \ldots, a_m)$, where each $a_i \in \mathrm{F}$, then
$f(a_1, \ldots, a_{m-l}, a_{m-l+1}, \ldots, a_m) =$
$\sum_{i\in[0..h]^{m-l}} L_i(a_1, \cdots, a_{m-l}) \cdot f_i(a_{m-l+1}, \ldots, a_m)$

Figure 4: Procedure to Check the Concatenation Property

But since the polynomial L_j defined in the description of the procedure is 1 at $j \in [0, h]^{m-l}$ and 0 at every point in $[0, h]^{m-l} \setminus \{j\}$, Condition (23) is equivalent to

$$f(i, u) = \sum_{j\in[0,h]^{m-l}} L_j(i) \cdot f_j(u) \quad \forall\, i \in [0, h]^{m-l}, u \in [0, h]^l.$$

But f and $\sum_{j\in[0,h]^{m-l}} L_j \cdot f_j$ are degree-h polynomials in m variables, so if they agree on $[0, h]^m$, they are the same polynomial (this follows from the uniqueness of the polynomial extension; see Theorem 8). Hence the procedure accepts with probability 1.

The following theorem shows that if the procedure accepts with high probability, then the concatenation property holds.

Theorem 11 *Suppose the procedure described above is given a* 0.01*-close function* f *and some functions* $f_1, \ldots, f_p$, *and it accepts with probability more than* $1/4$. *Then* $f, f_1, \ldots, f_p$ *satisfy the concatenation property.*

Proof: Let g: $\mathrm{F}^m \to \mathrm{F}$ be defined as

$$g(x_1, \ldots, x_m) = \sum_{i\in[0..h]^{m-l}} L_i(x_1, \cdots, x_{m-l}) \cdot f_i(x_{m-l+1}, x_2, \ldots, x_m), \tag{24}$$

where L_i is 1 at i and 0 elsewhere in $[0..h]^{m-l}$. Note that the procedure compares the values of f and g at 1000 points, accepting iff they are the same. Hence if $\Delta(f, g) \geq 0.01$, then the procedure rejects with probability at least $(1 - 0.01)^{1000} > 3/4$. According to the hypothesis, the procedure accepts with probability at least $1/4$. Hence $\Delta(f, g) < 0.01$.

By hypothesis, f is 0.01-close, so it follows from triangle inequality that g is 0.02-close, and furthermore, that the polynomials closest to g and f are the same: $\widetilde{f} = \widetilde{g}$. We now show that the concatenation property follows. (We will use Lemma 14, which is proved below in Section 5.)

For a function $t: \mathrm{F}^m \to \mathrm{F}$, and points $\vec{b} \in \mathrm{F}^l$ and $\vec{a} \in \mathrm{F}^{m-l}$, let $t|_{\vec{x^1}=\vec{a}}$ denote the restriction of t obtained by fixing its first $m-l$ arguments according to $(x_1, \ldots, x_{m-l}) = \vec{a}$ and let $t|_{\vec{x^2}=\vec{b}}$ denote the restriction of t obtained by fixing its last l arguments according to $(x_{m-l+1}, \ldots, x_m) = \vec{b}$.

Notice that for each $\vec{b} \in \mathrm{F}^l$, the function $g|_{\vec{x^2}=\vec{b}}$ is given by

$$g|_{\vec{x^2}=\vec{b}}(x_1, \ldots, x_{m-l}) = \sum_{i \in [0..h]^{m-l}} L_i(x_1, \cdots, x_{m-l}) \cdot f_i(\vec{b}) \tag{25}$$

Since each L_i is a degree-h polynomial, so is $g|_{\vec{x^2}=\vec{b}}$.

Lemma 14 below applies to functions from F^m to F that, like g, are 0.02-close, and have the property that fixing their last l arguments always gives a degree-h polynomial. (In terms of Definition 18 of Section 5, such functions are $(m-l)$-*nice*.) The Lemma shows that every such function has the following property (assuming the field is "large enough"): fixing its first $m-l$ arguments always gives a 0.03-close function. In other words, for all $\vec{a} \in \mathrm{F}^{m-l}$, the restriction $g|_{\vec{x^1}=\vec{a}}$ is 0.03-close. Furthermore, the Lemma tells us that the polynomial nearest to $g|_{\vec{x^1}=\vec{a}}$ is just the corresponding restriction of $\widetilde{g}$, namely, $\widetilde{g}|_{\vec{x^1}=\vec{a}}$. We conclude that $\Delta(g|_{\vec{x^1}=\vec{a}}, \widetilde{g}|_{\vec{x^1}=\vec{a}}) \leq 0.03$ for all $\vec{a} \in \mathrm{F}^{m-l}$.

Now note that $g|_{\vec{x^1}=\vec{a}}$ is merely the function

$$g|_{\vec{x^1}=\vec{a}}(x_{m-l+1}, \ldots, x_m) = \sum_{i \in [0..h]^{m-l}} L_i(\vec{a}) \cdot f_i(x_{m-l+1}, \ldots, x_m).$$

Consider $g|_{\vec{x^1}=i}$, where $i \in [0,h]^{m-l}$. Since $L_i(\vec{a}) = 0$ for $\vec{a} \in [0,h]^{m-l} \setminus \{i\}$, we have,

$$g|_{\vec{x^1}=i} = f_i \qquad \forall i \in [0,h]^{m-l}.$$

Thus we conclude that each f_i is 0.03-close, and one half of the concatenation property (Condition (21)) is proved.

As for the second half (Condition (22)), note that since $f_i = g|_{\vec{x^1}=i}$ for each $i \in [0,h]^{m-l}$, we have $\Delta(f_i, \widetilde{g}|_{\vec{x^1}=i}) \leq 0.03$. Thus $\widetilde{f}_i$, the polynomial nearest to f_i, is $\widetilde{g}|_{\vec{x^1}=i}$. But since we proved earlier that $\widetilde{g} = \widetilde{f}$, we conclude that $\widetilde{f}_i = \widetilde{f}|_{\vec{x^1}=i}$.

The rest of the argument is similar to that given in the paragraphs before this lemma. By definition of the polynomial extension, the degree-h polynomial $\widetilde{f}: \mathrm{F}^m \to \mathrm{F}$ is $\sigma_1(z)$, where z is the sequence of values that $\widetilde{f}$ takes on $[0,h]^m$. Hence $\sigma^{-1}(\widetilde{f})$ is z. Similarly, for each $i \in [0,h]^{m-l}$, $\widetilde{f}|_{\vec{x^1}=i}$ is $\sigma(z_i)$, where z_i is the sequence of values of $\widetilde{f}$ on $\{(i,u) : u \in [0,h]^l\}$. Therefore we have trivially

$$\sigma_1^{-1}(\widetilde{f}) = \sigma^{-1}(\widetilde{f}|_{\vec{x^1}=1}) \circ \cdots \circ \sigma^{-1}(\widetilde{f}|_{\vec{x^1}=p}).$$

But we proved above that $\widetilde{f}_i = \widetilde{f}|_{\vec{x^1}=i}$. Hence the second half of the concatenation property, Condition (22), now follows. $\square$

4.5 The Sum-Check

We describe Procedure 1, the Sum-Check. The procedure is due to [LFKN92]; we include it here chiefly for completeness, and to point out that the procedure can query the proof nonadapatively (this fact is not clear in existing descriptions).

The inputs to the procedure consist of a degree-d polynomial B in l variables, a set $H \subseteq \mathrm{F}$, and a value $c \in \mathrm{F}$. The procedure has to verify that the sum of the values of B on the subset H^l of F^l is c. It will need, in addition to the table of values of B and the integers l and d, an extra table. We first describe first what the procedure expects in the table.

When we fix all arguments of B but one, we get a univariate polynomial of degree at most d in the unfixed argument. It follows that for i s.t. $1 \le i < l$, and $a_1, \ldots, a_{i-1} \in \mathrm{F}$, the sum

$$\sum_{x_{i+1},..,x_l \in H} B(a_1, .., a_{i-1},\, x_i,\, x_{i+1}, .., x_l) \tag{26}$$

is a degree-d univariate polynomial in the variable x_i. We denote this sum by $B_{a_1,..,a_{i-1}}(x_i)$. (For $i = 1$ we use the notation $B_\epsilon(x_1)$.)

Example 2 The univariate polynomial $B_\epsilon(x_1)$ is represented by $d+1$ coefficients. When we substitute $x_1 = a$ in this polynomial, we get the value $B_\epsilon(a)$, which, by definition, is the sum of B on the following sub-cube:

$$\{(x_1, \ldots, x_l) : x_1 = a, \text{ and } x_2, \ldots, x_l \in H\}.$$

(Equivalently, we can view the value $B_\epsilon(a)$ as the sum of the values of the restriction $B|_{x_1=a}$ on H^{l-1}.) Thus $B_\epsilon(x_1)$ is a representation of $q = |\mathrm{F}|$ sums using $d+1$ coefficients. Suppose $f(x_1)$ is another degree-d univariate polynomial different from $B_\epsilon(x_1)$. Then the two polynomials agree at no more than d points. Hence for $q - d$ values of a, the value $f(a)$ is *not* the sum of $B|_{x_1=a}$ on H^{l-1}. This observation is useful in designing the Sum-Check.

Definition 14 A *table of partial sums* is any table containing for every i, $1 \le i \le l$, and every $a_1, ..., a_{i-1} \in \mathrm{F}$, a univariate polynomial $g_{a_1,..a_{i-1}}(x_i)$ of degree d. The entire table is denoted by g.

Now we describe Procedure 1. It expects the table T to be a table of partial sums. (*In a good proof, the table contains the set of polynomials defined in (26).*)

Sum-Check

Inputs: $B \in \mathrm{F}_d[x_1, \ldots, x_l]$, $c \in \mathrm{F}$, and $H \subseteq F$.
Goal: Verify that the sum of B on H^l is c.
Allowed to query: Table of partial sums, g.

current-value = c
Pick random $a_1, \ldots, a_l \in \mathrm{F}$
For $i = 1$ to l **do**
  if current-value $\neq \sum_{x_i \in H} g_{a_1,..,a_{i-1}}(x_i)$
    output REJECT; **exit**
  else
    current-value = $g_{a_1,..,a_{i-1}}(a_i)$.

/ ⋆⋆*Remark: When for loop ends, current-value* = $g_{a_1,...,a_{l-1}}(a_l)$. ⋆⋆ /
If current-value $\neq B(a_1, \ldots, a_l)$
  output REJECT
else
  output ACCEPT

Complexity: The procedure needs $l \log q$ random bits to generate elements $a_1, \ldots, a_l$ randomly from F. It needs the value of B at one point, namely, $(a_1, \ldots, a_l)$. In total, it reads l entries from the table of partial sums, where each entry is a string of size at most $(d+1)\log q$. It performs $O(ldh)$ field operations, where $h = |H|$. Therefore the running time is poly$(ldh \log q)$.

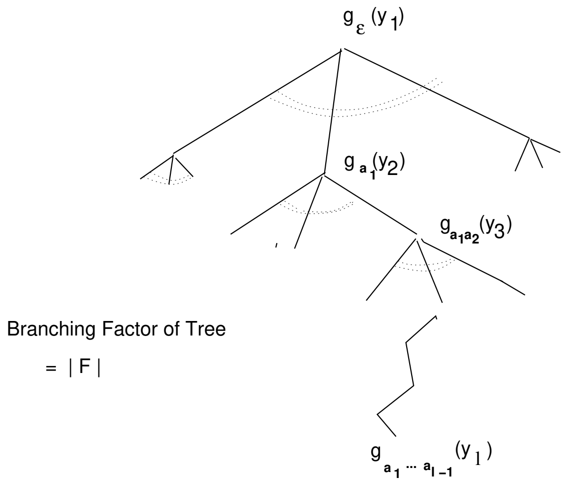

Figure 5: A table of partial sums may be conceptualized as a tree of branching factor $q = |\mathrm{F}|$. The Sum-Check follows a random path down this tree.

Correctness: Suppose B sums to c on H^l. The procedure clearly accepts with probability 1 the table of partial sums containing the univariate polynomials B_ϵ, $B_{a_1}(x_2)$, etc. defined in (26).

Suppose B does not sum to c. The next lemma shows that then the procedure rejects with high probability (when $dl \ll q$).

Lemma 12 *Let l, d be integers and $F = GF(q)$ be any field. Then for every $B \in F_d[x_1, \ldots, x_l]$ and $c \in F$, if B does not sum to c on H^l then*

$$\Pr[\textit{the Sum-Check outputs REJECT}] \geq 1 - \frac{dl}{q},$$

regardless of what the table of partial sums contains.

Proof: The proof is by induction on the number of variables, l. Such an induction works because the Sum-Check is essentially a recursive procedure: it randomly reduces the problem of checking the sum of a polynomial in l variables to checking the sum of a polynomial in $l-1$ variables.

To see this, view the table of partial sums as a tree of branching factor q (see Figure 5). The polynomial $g_\epsilon(x_1)$ is stored at the root of the tree, and the set of polynomials $\{g_{a_1}(x_2) : a_1 \in \mathrm{F}\}$ are stored on the children of the root, and so on.

The first step in the Sum-Check verifies that that the sum of the values taken by g_ϵ on the set H is c. Suppose the given multivariate polynomial B does not sum to c on H^l. Then the sum of the values taken by B_ϵ on H is not c, and the first step can succeed only if $g_\epsilon \neq B_\epsilon$. But if $g_\epsilon \neq B_\epsilon$ then, as observed in Example 2, $g_\epsilon(a) \neq B_\epsilon(a)$ for $q-d$ values of a in F. That is to say, for $q-d$ values of a, the value $g_\epsilon(a)$ is *not* the sum of the restriction $B|_{x_1=a}$ on H^{l-1}. Since $d \ll q$, it suffices to picks a value for x_1 randomly out of F, say a_1, and check (recursively) that $B|_{x_1=a_1}$ sums to $g_\epsilon(a_1)$ on H^{l-1}. (Note: While checking the sum of $B|_{x_1=a_1}$ on H^{l-1}, the recursive call must use as the table of partial sums the sequence of polynomials stored in the a_1th sub-tree of the root.) This is exactly what the remaining steps of the Sum-Check do. In this sense the Sum-Check is a recursive procedure.

Now we do the inductive proof.

Base case: $l = 1$. This is easy, since $B(x_1)$ is a univariate polynomial, and $B_\epsilon = B$. The table contains only one polynomial g_ϵ. If $g_\epsilon = B$, then g_ϵ doesn't sum to c either, and is rejected with probability 1. If $g_\epsilon \neq B$, then the two disagree in at least $q-d$ points. Therefore $\Pr_{a_1}[g_\epsilon(a_1) \neq B(a_1)] \geq 1 - d/q$. Thus the base case is proved.

Inductive Step: Suppose the lemma statement is true for all polynomials in $l-1$ variables. Now there are two cases.

Case (i): $g_\epsilon = B_\epsilon$. In this case,

$$\sum_{x_1 \in H} g_\epsilon(x_1) = \sum_{x_1 \in H} B_\epsilon(x_1) \neq c,$$

so the procedure will output REJECT rightaway (i.e., with probability 1). So the inductive step is complete.

Case (ii): $g_\epsilon \neq B_\epsilon$. In this case, as observed in Example 2, for $q-d$ values of a,

$$g_\epsilon(a) \neq B_\epsilon(a). \tag{27}$$

Let $a_1 \in \mathrm{F}$ be such that $g_\epsilon(a_1) \neq B_\epsilon(a_1)$ (i.e., $g_\epsilon(a_1)$ is not the sum of $B|_{x_1=a_1}$ on H^{l-1}). By the inductive assumption, no table of partial sums can convince the Sum-Check with probability more than $d(l-1)/q$ that $g_\epsilon(a_1)$ is the sum of $B|_{x_1=a_1}$ on H^{l-1}. In particular, the table of partial sums stored in the subtree rooted at the a_1th child of the root cannot make the Sum-Check accept with probability more than $d(l-1)/q$. Since this is true for $q-d$ values of a_1, the overall probability of rejection is at least $(\frac{q-d}{q}) \cdot (1 - \frac{d(l-1)}{q}) \geq 1 - dl/q$.

In either case, the inductive step is complete.

□

4.6 Low Degree Test

This section describes Procedure 2, the *low degree test*. We remind the reader that for a function $f : \mathrm{F}^l \to \mathrm{F}$, $\Delta_h(f)$ denotes the distance of f to $\mathrm{F}_h[x_1, \ldots, x_l]$, the code of degree h polynomials.

The procedure is given a function $f : \mathrm{F}^m \to \mathrm{F}$, an integer h, and a fraction $\delta > 0$. It has to determine whether f is δ-close, that is, $\Delta_h(f) \leq \delta$. It accepts every polynomial in $\mathrm{F}_h[x_1, \ldots, x_m]$ with probability 1 and rejects every function that is not δ-close with probability at least $3/4$. The procedure described here is an obvious modification of the ones in [FGL+91, She91] (which were described for degree $h = 1$ and were called *multilinearity tests*). We improve its analysis to show that it needs to examine only $O(m)$ entries in the proof, instead of the $O(mh)$ entries required by the earlier analysis. Our improvement uses ideas from algebra and coding theory (as opposed to the counting arguments used in earlier papers), and has the adverse side-effect of requiring $|\mathrm{F}| = \Omega(h^3 m^2)$ (for the existing analysis, somewhat smaller fields suffice).

The procedure requires, in addition to the table of values of f, an extra table whose contents we describe next.

Definition 15 For an $(m-1)$-tuple $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m) \in \mathrm{F}^{m-1}$, the subset of F^m given by

$$\{(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_m) \ : \ x \in \mathrm{F}\}$$

is called a *dimension-i line*, and denoted by $\{a_1, \ldots, a_{i-1}, *, a_{i+1}, \ldots, a_m\}$.

Note that if $f : \mathrm{F}^m \to \mathrm{F}$ is a polynomial in $\mathrm{F}_h[x_1, \ldots, x_m]$, then its restriction on every dimension-i line is a univariate polynomial in x_i of degree h.

Definition 16 If h, m are positive integers, then an *h-decomposition of dimension m* (or just *decomposition* when h and m are clear from the context) is a table that contains, for each i, $1 \leq i \leq l$ and for each dimension-i line of F^m, a univariate degree-h polynomial.

The procedure expects the extra table in the proof to contain an h-decomposition of dimension m that supposedly describes the restrictions of f on all lines in F^m. Let g denote this decomposition and $g[a_1, \ldots, a_{i-1}, *, a_{i+1}, \ldots, a_m]$ denote the univariate polynomial provided in this decomposition for the line $\{a_1, \ldots, a_{i-1}, *, a_{i+1}, \ldots, a_m\}$.

Note that we can view the decomposition g as a sequence of m functions, $g^1, \ldots, g^m$, where $g^i\colon \mathrm{F}^m \to \mathrm{F}$ is formed using the univariate polynomials provided in g for dimension-i lines.

$$g^i(a_1, \ldots, a_m) = g[a_1, \ldots, a_{i-1}, *, a_{i+1}, \ldots, a_m](a_i). \tag{28}$$

By definition, the restriction of g^i to every dimension-i line is a univariate degree-h polynomial. Furthermore, since the decomposition is supposed to describe f, the procedure can expect it to obey, $\forall i, j \in [1, m]$ and $\forall (a_1, \ldots, a_m) \in \mathrm{F}^m$,

$$\begin{aligned} f(a_1, \ldots, a_m) &= g^i(a_1, \ldots, a_m) \\ &= g^j(a_1, \ldots, a_m). \end{aligned} \tag{29}$$

In particular, the previous statement should hold for $j = i+1$. This motivates the following definition.

Definition 17 A *checkpoint* is a member of $\{\langle i, a_1, ..., a_m\rangle : 1 \le i < m \text{ and } a_1, ..., a_m \in \mathrm{F}\}$. A decomposition g is *consistent* at a checkpoint $\langle i, a_1, ..., a_m\rangle$ if

$$g^i(a_1, \ldots, a_m) = g^{i+1}(a_1, \ldots, a_m) \tag{30}$$

Low Degree Test

Inputs: $f\colon \mathrm{F}^m \to \mathrm{F}$, integers h, and $\delta > 0$.
To Verify: f is δ-close to $\mathrm{F}_h[x_1, \ldots, x_m]$.
Is allowed to query: g, a decomposition of dimension m.

Output ACCEPT **iff** g passes the following checks.

(i) Pick $4/\delta$ random points in F^m and verify that f and g^1 agree on them.

(ii) Pick $8m/\delta$ random checkpoints and verify that the decomposition is consistent at each.

Correctness of the Procedure: Clearly, if $f \in \mathrm{F}_h[x_1, \ldots, x_m]$, and the decomposition g contains restrictions of f on all the lines, then this test accepts with probability 1.

Suppose f is not δ-close, where $\delta < 0.1$. We show that the test then rejects with probability more than $\frac{3}{4}$, regardless of what g contains. If $\Delta(f, g^1) \ge \delta/2$, then the probability that part (i) of the procedure fails is at least $1 - (1 - \delta/2)^{4/\delta} > 3/4$. So we may assume that $\Delta(f, g^1) < \delta/2$. But then g^1 is not $\delta/2$-close (since otherwise by the triangle inequality, f would be δ-close). Lemma 13 below (see also the remark following that lemma) shows that if g^1 is not $\delta/2$-close, then the fraction of inconsistent checkpoints in the decomposition is at least $\delta/4m$. Thus part (ii) of the test rejects with probability at least $1 - (1 - \delta/4m)^{8m/\delta} > 3/4$. Thus in every case, the procedure rejects with probability at least $3/4$.

Complexity: Each table entry is a degree-h univariate polynomial, and so is represented by $O(h \log |\mathrm{F}|)$ bits. The procedure reads $O(m)$ such entries. Also, it performs $\mathrm{poly}(mh)$ field operations, so its running time is $\mathrm{poly}(mh \log |\mathrm{F}|)$. Its randomness requirement may appear at first sight to be $O(m) \times O(\log |\mathrm{F}|^m)$ random bits, which is more than was claimed. So we indicate how to do the procedure with $O(\log |\mathrm{F}|^m)$ random bits.

As we pointed out, Lemma 13 shows that if g^1 is not $\delta/2$-close, then the fraction of inconsistent checkpoints in the decomposition is at least $\frac{\delta}{4m}$. Hence, by sampling in a pairwise independent fashion (see [CG89] for example), the verifier can choose a set of $O(m/\delta)$ checkpoints such that with probability $\frac{3}{4}$, at least one of them is inconsistent. This sampling uses only $O(m \log(|\mathrm{F}|))$ random bits when δ is a constant.

To finish the proof of correctness, we have to prove the claim about the fraction of checkpoints at which the decomposition is inconsistent.

Lemma 13 *Let $|F| = \Theta(h^3 l^2)$ and $l \geq 2$. Let $\Upsilon_h(l, \epsilon)$ denote the minimum fraction of inconsistent checkpoints among all h-decompositions of dimension l in which the function g^1 satisfies $\Delta_h(g^1) \geq \epsilon$. Then,*

$$\Upsilon_h(l, \varepsilon) \geq \frac{\gamma^{2(l-1)}}{l-1} \cdot \min\{0.1, \varepsilon\},$$

where $\gamma = 1 - \sqrt{\frac{h}{|F|}}$.

Remark: We are interested in the case $\varepsilon < 0.1$ and $h/|\mathrm{F}| < 1/100l^2$. Then $\gamma^{2(l-1)} \geq (1 - 1/10l)^{2(l-1)} \geq 1/5$, so we get $\Upsilon_h(l, \varepsilon) \geq \varepsilon/4l$.

Proof: Let $(g^1, \ldots, g^l)$ be a decomposition of dimension l in which $\Delta_h(g^1) \geq \epsilon$. We use induction on l to lowerbound the fraction of checkpoints at which it is inconsistent.

Base case: (l =2) The Claim below implies that for a fraction γ of $a_1 \in \mathrm{F}$ we have

$$\Delta_h(g^1|_{x_1=a_1}) \geq \gamma \cdot \min\{0.1, \varepsilon\}.$$

By definition, $g^2|_{x_1=a_1}$ is a univariate polynomial. It follows that for a fraction γ of $a_1 \in \mathrm{F}$,

$$\delta(g^1|_{x_1=a_1}, g^2|_{x_1=a_1}) \geq \gamma \cdot \min\{0.1, \varepsilon\}.$$

So the fraction of inconsistencies in the decomposition is

$$\begin{aligned} \delta(g^1, g^2) &= \frac{1}{|\mathrm{F}|} \sum_{a_1 \in \mathrm{F}} \delta(g^1|_{x_1=a_1}, g^2|_{x_1=a_1}) \\ &\geq \gamma^2 \cdot \min\{0.1, \varepsilon\}, \end{aligned}$$

which proves the base case.

Induction step. Assuming the claimed lowerbound is true for $l-1$, we prove it for l.

For any $a_1 \in \mathrm{F}$ and $i \geq 2$, the dimension-i line $\{a_1, a_2, \ldots, a_{i-1}, \star, a_{i+1}, \ldots, a_l\}$ in F^l can be viewed as a dimension-$(i-1)$ line in the following subspace (of size $|\mathrm{F}|^{l-1}$):

$$\{(x_1, \ldots, x_m) : \text{ each } x_i \in \mathrm{F} \text{ and } x_1 = a_1\}.$$

Thus in the decomposition $g = (g^1, g^2, \ldots, g^l)$, we can view $(g^2, \ldots, g^l)$ as a disjoint union of the following $|\mathrm{F}|$ decompositions of dimension $l-1$.

$$(g^2|_{x_1=a_1}, \ldots, g^l|_{x_1=a_1}) \quad \forall a_1 \in \mathrm{F}. \tag{31}$$

Since each checkpoint involves a check along dimension i for $i \in [1, l-1]$, we account for the inconsistent checkpoints in g as coming from two sources: for each $a_1 \in \mathrm{F}$ there are (1) checkpoints on which $g^1|_{x_1=a_1}$ differs from $g^2|_{x_1=a_1}$, (this corresponds to checkpoints in which $i = 1$) and (2) inconsistent checkpoints in $(g^2|_{x_1=a_1}, g^3|_{x_1=a_1}, \ldots, g^m|_{x_1=a_1})$, a decomposition of $l-1$ dimensions. This number can be lowerbounded, using the inductive hypothesis, as a function of $\Delta_h(g^2|_{x_1=a_1})$.

Note that the total number of checkpoints in a decomposition of dimension l is $(l-1)\left|\mathrm{F}\right|^{l}$, and the number of inconsistent ones is at least $\Upsilon(l,\varepsilon)\cdot(l-1)\left|\mathrm{F}\right|^{l}$. Using an averaging over $a_1 \in \mathrm{F}$, we can lowerbound the latter by

$$\sum_{a_1\in\mathrm{F}}\left(\left|\mathrm{F}\right|^{l-1}\cdot\Delta(g^1|_{x_1=a_1},g^2|_{x_1=a_1})+(l-2)\left|\mathrm{F}\right|^{l-1}\cdot\Upsilon(l-1,\Delta_h(g^2|_{x_1=a_1}))\right). \tag{32}$$

Dividing throughout by $(l-1)\left|\mathrm{F}\right|^{l}$ and using the inductive hypothesis, we get

$$\begin{aligned}\Upsilon(l,\varepsilon) &\geq \frac{1}{\left|\mathrm{F}\right|(l-1)}\cdot\sum_{a_1\in\mathrm{F}}\left(\Delta(g^1|_{x_1=a_1},g^2|_{x_1=a_1})+(l-2)\cdot\Upsilon(l-1,\Delta_h(g^2|_{x_1=a_1}))\right)\\ &\geq \frac{\gamma^{2(l-2)}}{\left|\mathrm{F}\right|(l-1)}\cdot\sum_{a_1\in\mathrm{F}}\left(\Delta(g^1|_{x_1=a_1},g^2|_{x_1=a_1})+\min\left\{0.1,\Delta_h(g^2|_{x_1=a_1})\right\}\right).\end{aligned}$$

By the triangle inequality, $\Delta_h(g^2|_{x_1=a_1})+\Delta(g^1|_{x_1=a_1},\,g^2|_{x_1=a_1})\geq\Delta_h(g^1|_{x_1=a_1})$, so we can simplify the last expression to

$$\Upsilon(l,\varepsilon)\geq\frac{\gamma^{2(l-2)}}{\left|\mathrm{F}\right|(l-1)}\cdot\sum_{a_1\in\mathrm{F}}\min\left\{0.1,\Delta_h(g^1|_{x_1=a_1})\right\}. \tag{33}$$

The Claim below implies that for at least a fraction γ of $a_1\in\mathrm{F}$,

$$\Delta(g^1|_{x_1=a_1})\geq\gamma\cdot\min\left\{0.1,\varepsilon\right\}. \tag{34}$$

Restricting the sum in Expression (33) to these $\gamma\left|\mathrm{F}\right|$ values of $a_1\in\mathrm{F}$, we get the following lower-bound for $\Upsilon(l,\varepsilon)$:

$$\Upsilon(l,\varepsilon)\geq\frac{\gamma^{2(l-1)}}{(l-1)}\cdot\min\left\{0.1,\varepsilon\right\}.$$

This completes the induction.

Now we state and prove the claim mentioned above.

Claim: *Let $t\colon F^l\to F$ be a function whose restriction on every dimension-1 line is a univariate polynomial of degree h. If $\varepsilon=\Delta_h(t)$, then for at least a fraction γ of $a_1\in F$,*

$$\Delta_h(t|_{x_1=a_1})\geq\gamma\cdot\min\left\{0.1,\varepsilon\right\}, \tag{35}$$

where $\gamma=1-\sqrt{\frac{h}{|F|}}$ and $|F|=\Theta(h^3l^2)$.

Proof of the Claim: Note that the function t is 1-nice (in the sense of Definition 18 below) and therefore Lemma 16 and Corollary 15 apply to it. We prove the Claim by considering two cases: $\varepsilon>0.2$ and $\varepsilon\leq0.2$.

Assume first that $\varepsilon>0.2$. Then we claim that the restriction $t|_{x_1=a_1}$ is 0.1-close for no more than $10h$ values $a_1\in\mathrm{F}$. For, if the number of such a_1's exceeds $10h$, then by Lemma 16 we would conclude that t is 1/9-close, which contradicts $\varepsilon>0.2$. So the fraction of a_1's such that $\Delta_h(t|_{x_1=a_1})\geq0.1$ is at least $1-\frac{10h}{|\mathrm{F}|}\geq1-\sqrt{\frac{h}{|\mathrm{F}|}}=\gamma$.

Now assume $\varepsilon\leq0.2$. Then Corollary 15 below implies that $\Delta_h(t|_{x_1=a_1})\geq\gamma\cdot\varepsilon$ for a fraction γ of $a_1\in\mathrm{F}$.

In either case, the Claim has been proved.

This finishes the proof of Lemma 13. $\square$

5 Two Lemmas About Polynomials

This section proves two lemmas, Lemma 14 and 16, about the restrictions of k-*nice* functions, which are defined below. We remind the reader that restrictions of functions were defined in Definition 10 in Section 4.1.1.

In all the lemmas and definitions in this section , h and m stand for arbitrary positive integers and F denotes a field of size at least $\Omega(m^2h^3)$. (Some lemma statements are true even if $|\mathrm{F}|$ is smaller that $\Omega(m^2h^3)$, but we don't dwell on that.) Both $1/m$ and $1/h$ are considered to be very small. As usual, $\Delta_h(f)$ denotes the distance of function f to the nearest degree-h polynomial.

Definition 18 Let integer k be such that $1 \le k < m$. A function $f\colon \mathrm{F}^m \to \mathrm{F}$ is k-*nice*, if for every sequence of $m-k$ values $a_1, a_2, \ldots, a_{m-k} \in \mathrm{F}$, the restriction $f|_{(x_{m-k+1},\ldots,x_m)=(a_1,\ldots,a_{m-k})}$, obtained by fixing the last $m-k$ arguments of f to $a_1, \ldots, a_{m-k}$, is a degree-h polynomial.

For example, a 1-nice function is one whose restriction on every dimension-1 line is a univariate degree-h polynomial.

5.1 δ-close functions with nice restrictions

By definition, a k-nice function g behaves "nicely" when we fix its last $m-k$ arguments. The next lemma (which was used in Section 4.4) shows that if g is in addition 0.2-close, then it also behaves "nicely" when we fix its first k arguments. Specifically, part 1 of the lemma upperbounds, as a function of $\Delta_h(g)$, the distance between this restriction and the degree-h polynomial closest to it. Part 2 lowerbounds that same distance, for "most" restrictions.

Lemma 14 *Let k be an integer such that $1 \le k < m$. For $g\colon F^m \to F$ and $\vec{b} \in F^k$, let $g|_{\vec{x}^1=\vec{b}}$ denote the restriction of g obtained by fixing the first k arguments according to $(x_1, \ldots, x_k) = \vec{b}$.*

Let g be k-nice and $\Delta_h(g) < 0.2$. Then restrictions of g satisfy the following, where $\delta = \Delta_h(g)$:

1. *$\Delta_h(g|_{\vec{x}^1=\vec{b}}) \le \beta \cdot \delta$ for every $\vec{b} \in F^k$, where $\beta = \frac{1}{1-\frac{hk}{|F|}}$.*

2. *$\Delta_h(g|_{\vec{x}^1=\vec{b}}) \ge \gamma \cdot \delta$ for at least γ fraction of $\vec{b} \in F^k$, where $\gamma = 1 - \sqrt{\frac{hk}{|F|}}$.*

Proof: Consider an $\mathrm{F}^k \times \mathrm{F}^{m-k}$ matrix whose rows (resp., columns) are indexed by points in F^k (resp., F^{m-k}). For counting purposes, let us put a $*$ at the intersection of row $\vec{b} \in \mathrm{F}^k$ and column $\vec{a} \in \mathrm{F}^{m-k}$ if $g(\vec{b}, \vec{a}) \neq \widetilde{g}(\vec{b}, \vec{a})$, where $\widetilde{g}$ is the degree h polynomial nearest to g. By hypothesis, the fraction of entries with $*$'s is δ.

$$
\begin{array}{cc}
 & \begin{array}{ccccc} & & \vec{a} \in \mathrm{F}^{m-k} & & \\ & & \downarrow & & \end{array} \\
\vec{b} \in \mathrm{F}^k \to & \left[\begin{array}{ccccc}
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \star & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right]
\end{array}
$$

Let $\vec{a} \in \mathrm{F}^{m-k}$ be a column. The restrictions of g and $\widetilde{g}$ to it are degree-h polynomials (for g this follows from the hypothesis that g is k-nice; for $\widetilde{g}$, by virtue of the fact that restrictions of a degree-h polynomial are also degree-h polynomials). Therefore if there is even one $*$ in this column, the two restrictions are unequal, and hence disagree on at least

$1 - hk/|\mathrm{F}| = \beta^{-1}$ fraction of the points in the column. That is to say, if there exists a $*$ in the column, then at least β^{-1} fraction of the entries in that column are $*$'s.

If p is the fraction of columns with at least one $*$, then the previous observation implies that the fraction of $*$'s in the matrix is at least $p \cdot \beta^{-1}$. But this fraction is δ, so $p \leq \delta \cdot \beta$.

We conclude that at most $\delta \cdot \beta$ of the columns have any $*$'s at all, which implies that the fraction of $*$'s in each row is at most $\delta \cdot \beta$. But the fraction of $*$'s in a row $\vec{b} \in \mathrm{F}^k$ is just the distance between the corresponding restrictions of g and $\widetilde{g}$. We conclude that for every $\vec{b} \in \mathrm{F}^k$, the distance

$$\Delta(g|_{\vec{x}^1=\vec{b}}, \widetilde{g}|_{\vec{x}^1=\vec{b}}) \leq \delta \cdot \beta.$$

Thus part 1 has been proved.

For part 2 we use the hypothesis that $\delta < 0.2$. As already proved, $\Delta(g|_{\vec{x}^1=\vec{b}}, \widetilde{g}|_{\vec{x}^1=\vec{b}}) \leq \delta \cdot \beta < 1/4$ for every row $\vec{b}$. Since $\widetilde{g}|_{\vec{x}^1=\vec{b}}$ is a degree-h polynomial, this also means that $g|_{\vec{x}^1=\vec{b}}$ is $\delta \cdot \beta$-close for every row $\vec{b}$ and $\widetilde{g}|_{\vec{x}^1=\vec{b}}$ is the unique degree-h polynomial closest to it (the uniqueness follows from the fact that $\beta \cdot \delta < 1/4$ and Definition 8). In other words, for every $\vec{b}$,

$$\begin{aligned}
\Delta_h(g|_{\vec{x}^1=\vec{b}}) &= \Delta(g|_{\vec{x}^1=\vec{b}}, \widetilde{g}|_{\vec{x}^1=\vec{b}}) \\
&= \text{fraction of points in row } \vec{b} \text{ where } g \text{ and } \widetilde{g} \text{ disagree} \\
&= \text{fraction of points in row } \vec{b} \text{ that have } *\text{'s.}
\end{aligned}$$

Now let ρ be the fraction of rows such that $\Delta_h(g|_{\vec{x}^1=\vec{b}}) < \gamma\delta$. Only a δ fraction of the entries in the matrix are $*$'s, and by part 1 the fraction of entries in each row is at most $\delta \cdot \beta$. So we have:

$$\begin{aligned}
\delta = \Delta(g, \widetilde{g}) &= \frac{1}{|\mathrm{F}|^k} \sum_{\vec{b} \in \mathrm{F}^k} \Delta(g|_{\vec{x}^1=\vec{b}}, \widetilde{g}\,|_{\vec{x}^1=\vec{b}}) \\
&\leq \rho \cdot \gamma\delta + (1-\rho) \cdot \delta\beta.
\end{aligned}$$

It follows that

$$\rho \leq \frac{\beta - 1}{\beta - \gamma} = \frac{\sqrt{\frac{hk}{|\mathrm{F}|}}}{1 + \sqrt{\frac{hk}{|\mathrm{F}|}} - \frac{hk}{|\mathrm{F}|}} < \sqrt{\frac{hk}{|\mathrm{F}|}} = 1 - \gamma.$$

Hence part 2 has been proved.

□

We state the following Corollary for sake of completeness, since it was used in the proof of Lemma 13. It is just a special case — namely, when $k = 1$ — of part 2 of the previous lemma.

Corollary 15 *Let a function $g\colon F^m \to F$ be 1-nice. In other words, the restriction of g on every dimension-1 line is a degree-h univariate polynomial. Let $\Delta_h(g) = \delta \leq 0.2$.*

Then the fraction of $a_1 \in F$ such that the restriction $g|_{x_1=a_1}$ is not $\gamma \cdot \delta$-close is least γ, where $\gamma = 1 - \sqrt{\frac{h}{|F|}}$.

5.2 1-nice Functions whose restrictions are δ-close

The main lemma in this section, Lemma 16, concerns 1-nice functions. It is a strong contrapositive to part 1 of Lemma 14, in the following sense. Lemma 14 shows that if a 1-nice function is δ-close, then restricting its first argument gives a function that is $\delta(1 + o(1))$-close. Lemma 16 will show that if "many" (actually, just a "few") restrictions of a 1-nice function are 0.1-close, then the function is 0.12-close.

Lemma 16 *Let $f\colon F^m \to F$ be a 1-nice function and suppose there are points $a_1, \ldots, a_{10h} \in F$ such that the restriction $f|_{x_1=a_i}$ is 0.1-close for $i = 1, ..., 10h$. If $|F| = \Omega(h^3m^2)$, then f is 1/9-close.*

Note: This lemma has subsequently found important applications in the work of [ALM+92]. The relevant case there is $m = 2$, for which simpler proofs have been found [Sud92, PS94]. Currently, our proof is the only one known for general m.

The mathematical facts used in our proof will include some very basic linear algebra, specifically, how to solve systems of linear equations. All assumed facts appear in the appendix.

In the rest of this section, f, m, h are the same as those in the hypothesis of Lemma 16.

For clarity, view the values of f as lying in a matrix with $|\mathrm{F}|$ columns and $|\mathrm{F}|^{m-1}$ rows. (For $\vec{b} \in \mathrm{F}^{m-1}$ and $a \in \mathrm{F}$, $f(a, \vec{b})$ is at the intersection of the column given by a and the row given by $\vec{b}$.)

$$\begin{array}{cc} & \begin{array}{ccccc} a_1 & a_2 & \ldots & a_{10h} & \ldots \end{array} \\ \vec{b} \in \mathrm{F}^{m-1} \rightarrow & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \ldots \\ \cdot & \cdot & \cdot & \cdot & \ldots \\ \cdot & \cdot & \cdot & \cdot & \ldots \\ \cdot & \cdot & \cdot & \cdot & \ldots \end{pmatrix} \end{array}$$

According to the hypothesis:

1. Function f is 1-nice. Hence in each row, a univariate polynomial of degree h describes all $|\mathrm{F}|$ entries. For row $\vec{b}$, denote this polynomial by $g_{\vec{b}}(x)$, and call it *the row polynomial for $\vec{b}$*.

2. The restriction of f to the columns $a_1, \ldots, a_{10h}$ is 0.1-close. For notational ease, we denote the restriction of f to column a_i by f_i (note that our usual notation would be $f|_{x_1=a_i}$) and by $\widetilde{f}_i$ the degree-h polynomial f_i is close to.

The proof will rely on an "extrapolation" argument, and will use (except in the proof of Corollary 18) only the columns $a_1, \ldots, a_{10h}$. For purposes of counting, put a $\star$ symbol in the entry $(a_i, \vec{b})$ if $f_i(a_i, \vec{b}) \neq \widetilde{f}_i(\vec{b})$. Note that since the row polynomial $g_{\vec{b}}$ always describes f, a $\star$ in row $\vec{b}$ also denotes a place where the row polynomial $g_{\vec{b}}$ disagrees with $\widetilde{f}_i$. *A priori*, the $\star$'s might be arbitrarily distributed thoughout the matrix, subject only to the restriction that no more than 0.1 fraction of the entries in a column be $\star$'s. The following lemma shows that the distribution of $\star$'s is much more restricted.

Lemma 17 *In the matrix described in the previous paragraph, there is a submatrix consisting of 0.4 fraction of the rows and $h+1$ columns (out of $\{a_1, \ldots, a_{10h}\}$) that contains no $\star$'s.*

The proof of the lemma, given at the end of this section, involves finding a "clean" description (i.e., as roots of a low-degree polynomial) of the set of points with $\star$'s.

The following corollary of Lemma 17 not only shows that f is 1/9-close (thus proving Lemma 16), but also provides an explicit formula for the degree-h polynomial closest to f.

Corollary 18 *Let $\{a_1, a_2, \ldots, a_{h+1}\}$ be the set of columns appearing in the submatrix guaranteed by Lemma 17. Then the following polynomial $\phi \in F_h[x_1, \ldots, x_m]$ satisfies $\delta(f, \phi) \leq 1/9$.*

$$\phi(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{h+1} L_{a_i}(x_1) \cdot \widetilde{f}_i(x_2, \ldots, x_m),$$

where $L_{a_i}(x_1)$ is the degree-h univariate polynomial that is 1 *at a_i and* 0 *at $\{a_1, \ldots, a_{h+1}\} \setminus \{a_i\}$.*

Proof: By definition of ϕ, its restriction to the columns $\{a_1, \ldots, a_{h+1}\}$ coincides with the corresponding $\widetilde{f}_i$ in the column. First we show that the same is true for its restrictions to columns $\{a_{h+2}, \ldots, a_{10h}\}$. Consider a row $\vec{b}$ that is part of the submatrix that is free of $\star$'s. The row polynomial $g_{\vec{b}}$ correctly describes the values of $\widetilde{f}_1, \ldots, \widetilde{f}_{h+1}$, and hence of ϕ. It follows that the restriction of ϕ to the row — by construction, a univariate degree h polynomial — is identical to the row polynomial, and hence describes values of f on the entire row. Since the previous observation is true for every row of the sub-matrix, $\Delta(f, \phi) \leq 1 - 0.4 = 0.6$, and furthermore, $\Delta(f_i, \phi|_{x_1=a_i}) \leq 0.6$ for $i = 1, \ldots, 10h$. By hypothesis, $\Delta(f_i, \widetilde{f}_i) \leq 0.1$, so the triangle inequality implies that $\Delta(\widetilde{f}_i, \phi|_{x_1=a_i}) \leq 0.6 + 0.1 \leq 0.7$ But since $\phi|_{x_1=a_i}$ and $\widetilde{f}_i$ are both degree-h polynomials, $\Delta(\widetilde{f}_i, \phi|_{x_1=a_i})$ is either 0 or least $1 - mh/|\mathrm{F}| = 1 - o(1)$. Hence we conclude that $\Delta(\widetilde{f}_i, \phi|_{x_1=a_i}) = 0$, and $\phi|_{x_1=a_i} = \widetilde{f}_i$ for $i = h+2, \ldots, 10h$, as claimed.

Now we show that $\Delta(f, \phi) < 1/9$. Note that this statement refers to the average distance between f and ϕ on *all* the columns. In contrast, the only columns we have been talking about (and in which we placed $\star$'s) are $a_1, \ldots, a_{10h}$.

For each column a_i, where $i \in [1, 10h]$, since ϕ coincides with $\widetilde{f}_i$ in column a_i, an entry with a $\star$ corresponds exactly to a disagreement between the row polynomial $g_{\vec{b}}$ and ϕ. Furthermore, the restriction of ϕ to a row is also a univariate polynomial of degree h, so either it is different from the row polynomial (in which case there are at least $9h$ disagreements in the row among the first $10h$ columns), or is the same as the row polynomial (in which case there are no disagreements and no $\star$'s in the row). Since the fraction of $\star$'s in each column is at most 0.1, it follows by averaging that no more than 1/9 fraction of the rows have more than $9h$ $\star$'s. So ϕ describes 8/9 of the rows perfectly, and $\Delta(f, \phi) \leq 1/9$ □

From now on, our goal is prove Lemma 17. The concept of *well-described* sets of pairs will be relevant.

Definition 19 A set of (point, value) pairs, $\{(a_1, p_1), \ldots, (a_{10h}, p_{10h})\}$, where each $a_i, p_i \in \mathrm{F}$, is *well-described in* $\mathrm{F}_h[x]$ if there is a degree-h univariate polynomial $s \in \mathrm{F}_h[x]$ such that

$$s(a_i) = p_i \quad \text{for at least } 8h \text{ values of } i.$$

We say that such a polynomial s *well-describes* the pairs.

Note that the above polynomial s must be unique, since any other polynomial s' with the same properties agrees with it in at least $6h$ points, and so equals s.

To see the relevance of well-described sets to Lemma 17, recall that in the matrix (with $10h$ columns) defined in connection with the lemma, the fraction of $\star$'s in each column is at most 0.1. So the overall fraction of entries with $\star$'s is at most 0.1. Hence at most 1/2 of the rows have more than $0.2 \times 10h = 2h$ of these $\star$'s. Call the remaining rows *good*. In a good row $\vec{b}$, the set of pairs $\left\{(a_1, \widetilde{f}_1(\vec{b})), \ldots, (a_{10h}, \widetilde{f}_{10h}(\vec{b}))\right\}$ is well-described by the row polynomial $g_{\vec{b}}(x_1)$. The next lemma makes explicit the algebraic object that underlies well-described sets of pairs: an overconstrained linear system.

Lemma 19 (Berlekamp-Welch [BW]) *Let $(a_1, p_1), \ldots, (a_{10h}, p_{10h})$ be well-described in $F_h[x]$. Then*

1. *There is a polynomial $c \in F_{3h}[x]$ and a nonzero polynomial $e \in F_{2h}[x]$, such that*

$$c(a_i) = e(a_i) \cdot p_i \qquad \text{for } i = 1, 2, \ldots, 10h. \tag{36}$$

In other words, the following system of equations has a nontrivial solution for c_0, $\ldots, c_{3h}$, $e_0, \ldots, e_{2h}$ (which are intended as the coefficients of c, e respectively.

$$\begin{aligned}
c_0 + c_1 a_1 + \ldots + c_{3h} a_1^{3h} &= (e_0 + e_1 a_1 + \cdots + e_{2h} a_1^{2h}) \cdot p_1 \\
c_0 + c_1 a_2 + \ldots + c_{3h} a_2^{3h} &= (e_0 + e_1 a_2 + \cdots + e_{2h} a_2^{2h}) \cdot p_2 \\
&\vdots \\
c_0 + c_1 a_{10h} + \ldots + c_{3h} a_{10h}^{3h} &= (e_0 + e_1 a_{10h} + \cdots + e_{2h} a_{10h}^{2h}) \cdot p_{10h}
\end{aligned}$$

2. *For any polynomials c, e that satisfy the condition in (36), e divides c as a polynomial and the rational function c/e is the degree-h univariate polynomial that well-describes $(a_1, p_1), \ldots, (a_{10h}, p_{10h})$.*

Proof: Let s be the polynomial that well-describes the set.

Let e be a non-zero univariate polynomial of degree $2h$ that is 0 on the set of points $\{a_i : s(a_i) \neq p_i\}$. (If this set is empty, we let $e \equiv 1$, the unit polynomial.) Then we have

$$e(a_i) \cdot s(a_i) = p_i \cdot e(a_i) \quad \forall i \in \{1, \ldots, 10h\}. \tag{37}$$

Part 1 now follows, by defining $c(x) = s(x) \cdot e(x)$. The claim about the existence of a nontrivial solution to the given linear system also follows, since the coefficients of e and c defined above satisfy the system.

As for part 2, let c, e be any polynomials that satisfy Equation (36). Note that $s(x)e(x) - c(x)$ is zero at each a_i where $s(a_i) = p_i$. Since $s(x)$ well-describes $(a_1, p_1), \ldots, (a_{10h}, p_{10h})$, the polynomial $s(x)e(x) - c(x)$ must have at least $8h$ roots. But it has degree at most $3h$. Hence $s(x)e(x) - c(x) \equiv 0$. Therefore $c(x)/e(x) = s(x)$. $\square$

Note: The linear system in the statement of Lemma 19 is homogeneous and overconstrained: it has $10h$ constraints and only $(2h+1) + (3h+1)$ variables. Represent the system in standard form as $A \cdot \overline{y} = \overline{0}$, where A is a $(5h+2) \times (10h)$ matrix and $\overline{y}$ is the vector of variables $(c_0, c_1, \ldots, c_{3h}, e_0, \ldots, e_{2h})$. The system has a nontrivial solution. Hence Cramer's rule (Fact 25 in the Appendix) implies that the determinant of every $(5h+2) \times (5h+2)$ submatrix of A is zero. This observation will be useful in Lemma 20.

The following Lemma is an extension of Lemma 19 to the multivariate case we are interested in. We need the following definition.

Definition 20 For integers l, d, the set $\mathrm{F}_{l,d}[x_1, \ldots, x_m]$ contains functions from F^m to F that are polynomials of degree at most d in the first variable x_1 and at most l in the other variables.

Lemma 20 *Let $\{a_1, \ldots, a_{10h}\}$ be a set of points and $s_1, \ldots, s_{10h} \colon F^{m-1} \to F$ be polynomials of degree h such that for at least half of the points $\vec{b} \in F^{m-1}$,*

$$\left\{(a_1, s_1(\vec{b})), (a_2, s_2(\vec{b})), \ldots, (a_{10h}, s_{10h}(\vec{b}))\right\} \quad \text{is well-described in } F_h[x].$$

Then there is a polynomial $c \in F_{3h,6h^2}[x_1, \ldots, x_m]$ and a nonzero polynomial $e \in F_{2h,6h^2}[x_1, \ldots, x_m]$ such that

$$c(a_i, \vec{b}) = s_i(\vec{b}) \cdot e(a_i, \vec{b}) \quad \forall \vec{b} \in F^{m-1} \text{ and } i = 1, \ldots, 10h. \tag{38}$$

Note: It is proved that the desired property of c, e holds for all $\vec{b} \in \mathrm{F}^{m-1}$, even though the hypothesis mentioned a property true only for half of the $\vec{b}$'s.

Proof: (In the following, $\vec{y}$ denotes a point in F^{m-1}. A polynomial in $\vec{y}$ is a polynomial in $m-1$ variables.)

For $\vec{y} \in \mathrm{F}^m$, consider the following linear system in the variables $U_0(\vec{y}), \ldots, U_{2h}(\vec{y})$ and $V_0(\vec{y}), \ldots, V_{3h}(\vec{y})$

$$\begin{aligned}
V_0(\vec{y}) + V_1(\vec{y})a_1 + \ldots + V_{3h}(\vec{y})a_1^{3h} &= s_1(\vec{y}) \cdot (U_0(\vec{y}) + U_1(\vec{y})a_1 + \cdots + U_{2h}(\vec{y})a_1^{2h}) \\
V_0(\vec{y}) + V_1(\vec{y})a_2 + \ldots + V_{3h}(\vec{y})a_2^{3h} &= s_2(\vec{y}) \cdot (U_0(\vec{y}) + U_1(\vec{y})a_2 + \cdots + U_{2h}(\vec{y})a_2^{2h}) \\
&\vdots \\
V_0(\vec{y}) + V_1(\vec{y})a_{10h} + \ldots + V_{3h}(\vec{y})a_{10h}^{3h} &= s_{10h}(\vec{y}) \cdot (U_0(\vec{y}) + U_1(\vec{y})a_{10h} + \cdots + U_{2h}(\vec{y})a_{10h}^{2h})
\end{aligned}$$

Suppose we represent the system in standard form as $A \cdot \overline{z} = 0$, where A is a $(5h+2) \times 10h$ matrix of coeficients

$$A = \begin{pmatrix}
1 & a_1 & \ldots & a_1^{3h} & -s_1(\vec{y}) & -a_1 \cdot s_1(\vec{y}) & \ldots & -a_1^{2h} \cdot s_1(\vec{y}) \\
1 & a_2 & \ldots & a_2^{3h} & -s_1(\vec{y}) & -a_2 \cdot s_1(\vec{y}) & \ldots & -a_2^{2h} \cdot s_1(\vec{y}) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1 & a_{10h} & \ldots & a_{10h}^{3h} & -s_1(\vec{y}) & -a_{10h} \cdot s_1(\vec{y}) & \ldots & -a_{10h}^{2h} \cdot s_1(\vec{y})
\end{pmatrix}$$

and $\overline{z}$ is the vector of variables $(V_0(\vec{y}), \ldots, V_{3h}(\vec{y}), U_0(\vec{y}), U_{2h}(\vec{y}))$.

There are two ways to view this system. The first is as a collection of $|\mathrm{F}|^{m-1}$ systems, one for each $\vec{y} \in \mathrm{F}^{m-1}$. In this viewpoint, whenever $\vec{y}$ is one of the $\geq 1/2$ fraction of points for which the given set of pairs is well-described, Lemma 19 implies that the associated system has nontrivial solutions. As noted in the comment after Lemma 19, it follows that for half the $\vec{y} \in \mathrm{F}^{m-1}$, the determinant of every $(5h+2) \times (5h+2)$ submatrix of A is 0.

However, the above viewpoint ignores the fact that each entry of A is the value of some degree-h polynomial in $\vec{y}$. Thus the solutions to the collection of $|\mathrm{F}|^{m-1}$ systems are very likely connected to each other. The next viewpoint takes this into account.

We will adopt the viewpoint that A is a matrix over the ring of multivariate polynomials. We claim that from this viewpoint, the system has a nontrivial solution. Specifically, we construct a solution $(V_0(\vec{y}), \ldots, V_{3h}(\vec{y}), U_0(\vec{y}), V_{2h}(\vec{y}))$ in which each V_i, U_i is also a polynomial (of some approriate degree) in $\vec{y}$. To do this, it suffices (see Fact 25) to show that the determinant of every $(5h+2) \times (5h+2)$ submatrix of A is the zero polynomial in $\vec{y}$. Let B be the submatrix. Since each entry of B is a degree-h polynomial in $\vec{y}$, and $det(B)$ is itself a degree-$(5h+2)$ polynomial in the entries of B (see Fact 24), we conclude that $det(B)$ is a polynomial of degree $(5h+2)h$ in $\vec{y}$. As already noted (in the first "viewpoint"), this determinant is zero for half the values of $\vec{y}$ (i.e., for each value of $\vec{y}$ for which the above system has a solution). But a degree-$(5h^2+2h)$ polynomial that is zero at $1/2 > m(5h^2+2h)/|\mathrm{F}|$ fraction of points is identically zero. So $det(B)$ is identically zero.

Now we solve the system (over the ring of polynomials) by using Cramer's rule. Fact 25 implies that the solutions $U_i(\vec{y}), V_i(\vec{y})$ thus obtained are polynomials of degree at most $5h+1$ in the entries of A, in other words, are polynomials of degree at most $(5h+1) \cdot h$ in $\vec{y}$.

Now, since $(5h+1)h \leq 6h^2$ for $h \geq 1$, we conclude that the polynomials c and e defined by

$$c(x, \vec{y}) = \sum_{i=0}^{3h} V_i(\vec{y})x^i \quad \forall x \in \mathrm{F}, \vec{y} \in \mathrm{F}^{m-1} \tag{39}$$

and

$$e(x, \vec{y}) = \sum_{i=0}^{2h} U_i(\vec{y})x^i \quad \forall x \in \mathrm{F}, \vec{y} \in \mathrm{F}^{m-1} \tag{40}$$

are in $\mathrm{F}_{3h,6h^2}[x_1, \ldots, x_m]$ and $\mathrm{F}_{2h,6h^2}[x_1, \ldots, x_m]$ respectively, and fit the requirements of the Lemma.

□

Using Lemma 20, we now formally prove Lemma 17.

Proof:(Lemma 17) Call a row $\vec{b} \in \mathrm{F}^{m-1}$ *good* if it has fewer than $2h$ $\star$'s; in other words, the row polynomial $g_{\vec{b}}$ well-describes the set $\left\{(a_1, \widetilde{f_1}(\vec{b})), \ldots, (a_{10h}, \widetilde{f_{10h}}(\vec{b}))\right\}$. A simple averaging shows that good rows constitute at least half of all rows. Hence the polynomials $\left\{\widetilde{f_1}, \ldots, \widetilde{f_{10h}}\right\}$ satisfy the conditions of Lemma 20, and we conclude that there exists a polynomial $c \in \mathrm{F}_{3h,6h^2}[x_1, \ldots, x_m]$ and a nonzero polynomial $e \in \mathrm{F}_{2h,6h^2}[x_1, \ldots, x_m]$ such that

$$c(a_i, \vec{b}) = \widetilde{f_i}(\vec{b}) \cdot e(a_i, \vec{b}) \qquad \forall \ \ \vec{b} \in \mathrm{F}^{m-1} \ \text{ and } \ i = 1, \ldots, 10h. \tag{41}$$

Note that the restrictions of c and e to any row $\vec{b}$ are univariate polynomials in x_1 of degree $3h$ and $2h$ respectively. Denote these polynomials by $c_{\vec{b}}$ and $e_{\vec{b}}$.
Claim 1: If row $\vec{b}$ is good, then $e_{\vec{b}}$ divides $c_{\vec{b}}$ and furthermore, $c_{\vec{b}}/e_{\vec{b}} = g_{\vec{b}}$.
Proof of Claim 1: If row $\vec{b}$ is good, the set $\left\{(a_1, \widetilde{f_1}(\vec{b})), \ldots, (a_{10h}, \widetilde{f_{10h}}(\vec{b}))\right\}$ is well-described in $\mathrm{F}_h[x]$ (namely, by the row-polynomial $g_{\vec{b}}$). By our construction,

$$c_{\vec{b}}(a_i) = \widetilde{f_i}(\vec{b})\, e_{\vec{b}}(a_i) \quad \forall\, i = 1, \ldots, 10h.$$

So part 2 of Lemma 19 implies that $e_{\vec{b}}$ divides $c_{\vec{b}}$ and furthermore, $c_{\vec{b}}/e_{\vec{b}}$ well-describes the above set of pairs. Since the well-describing polynomial is unique (as noted after Definition 19), it follows that $c_{\vec{b}}/e_{\vec{b}} = g_{\vec{b}}$. Thus Claim 1 has been proved.

Since e is a nonzero polynomial of degree at most $2h$ in x_1, it cannot be identically zero on more than $2h$ of the columns. Assume w.l.o.g. that $\{a_1, \ldots, a_{8h}\}$ are columns where it is not identically zero. Call a row *nice* if the restriction of e to the columns $a_1, \ldots, a_{8h}$ is not zero in this row.
Claim 2: If row $\vec{b}$ is nice, then $c_{\vec{b}}/e_{\vec{b}}$ agrees with $\widetilde{f_i}$ in for $i = 1, \ldots, 8h$.
Proof of Claim 2: By inspecting Equation (41), we see that whenever $e \neq 0$ at a point in a column a_i, then c/e agrees with $\widetilde{f_i}$ at that point. Hence Claim 2 has been proved.

Combining Claims 1 and 2, we conclude that in a row $\vec{b}$ that is both nice and good, $c_{\vec{b}}/e_{\vec{b}}$ agrees with $\widetilde{f_i}$ for $i = 1, \ldots, 8h$ and $c_{\vec{b}}/e_{\vec{b}} = g_{\vec{b}}$. Thus the row polynomial $g_{\vec{b}}$ agrees with $\widetilde{f_i}$ for $i = 1, \ldots, 8h$, and so there are no $\star$'s in the first $8h$ entries of this row.

But how many rows are both nice and good? The fraction of good rows is at least $1/2$, as already noted. We claim that the fraction of nice rows is at least $1 - 48mh^3/\left|\mathrm{F}\right|^3$. The reason is that the restriction of e to any of the columns $\{a_1, \ldots, a_{8h}\}$ is a nonzero polynomial in $\mathrm{F}_{6h^2}[x_2, \ldots, x_m]$, and so is zero in this column at no more than $6(m-1)h^2/\left|\mathrm{F}\right|)$ fraction of the rows. So the fraction of rows in which e has zeroes in any one of these $8h$ columns is at most $\frac{8h \cdot 6h^2 m}{|\mathrm{F}|}$.

Since at least $1 - 48mh^3/\left|\mathrm{F}\right|^3$ rows are nice and at least $1/2$ of the rows are good, at least $1/2 - 48mh^3/\left|\mathrm{F}\right|^3$ of the rows are both. Since $48mh^3/\left|\mathrm{F}\right|^3 < 0.1$, we conclude that at least 0.4 fraction of the rows are both nice and good. Thus there are no $\star$'s in the submatrix consisting of the first $8h$ columns and all (the 0.4 fraction of) rows that are both good and nice. This proves Lemma 17. (Note that our matrix has $8h$ columns, which is larger than the $h + 1$ we needed to prove.) $\square$

6 Discussion

As mentioned above, our characterization of NP in terms of PCP is non-relativizing. This opens up the tantalizing possibility of separating the class NP from other classes (e.g., EXPTIME) using relativizing techniques (e.g., diagonalization). Of course, similar tantalizing possibilities were raised by recent results like IP = PSPACE or MIP = NEXPTIME, which are also nonrelativizing. (We note that in fact, IP = PSPACE is false with probability 1 with respect to a random oracle [CC+94].)

Note in this connection that the characterization NP $=$ PCP$(\log n, \log^{0.5+\epsilon} n)$ can be strengthened further by using an idea of [BFLS91]. Namely, if we change the model a little and ask that the input be provided in an encoded form (using a specific error-correcting code) to the verifier, then our $(\log n, \log^{0.5+\epsilon} n)$-restricted verifier runs in polylogarithmic time. This stronger characterization might be useful in any attempts to separate complexity classes using our new characterization of NP.

Another application of our ideas is to mechanical checking of formal mathematical proofs. Babai et al. [BFLS91] observed earlier that since the language

$$\{(T, 1^n) : T \text{ is a theorem of Peano arithmetic with a proof of size} \leq n.\}$$

is NP-complete, their main theorem implies that proofs for mathematical statements can be checked by reading only poly(*logn*) bits in them. (Note that above language is also NP-complete if we use instead of Peano arithmetic, any of the usual axiomatic systems for mathematics.) Our main result implies that these proofs can be checked by reading a sublogarithmic number of bits. We suspect that this result is largely of theoretical (as opposed to practical) interest.

As mentioned in the introduction, the initial motivation for our work was to show the NP-hardness of approximating clique, thereby solving an open problem of [FGL+91] (namely, how to change "almost"NP-hard in their main result to NP-hard). Indeed, by showing that NP = PCP$(\log n, \log n)$, and applying the reduction to Clique approximation [FGL+91], we get that approximating Clique to within a constant factor is NP-hard.

We were surprised to find out that our technique enables us to show an even better chracterization of NP in terms of PCP, with a sub-logarithmic number of queries. Thus there appeared to be no reason why the number of queries couldn't be decreased further. So, in an earlier draft of this paper, we posed this as an open problem.

Soon, the open problem acquired even more importance, since it was realized that decreasing the number of queries would have important consequences for *MAX-SNP* problems.

The class MAX-SNP was introduced by Papadimitriou and Yannakakis [PY91], as a framework for classifying approximation problems according to their difficulty. The authors defined a notion of *MAX-SNP-completeness*: a MAX-SNP-complete problem has the property that it has a *polynomial-time approximation scheme* iff every MAX-SNP problem does. (A *polynomial-time approximation scheme* for a problem is a family of polynomial-time algorithms, such that for every fixed $\epsilon > 0$, some algorithm from this family approximates the problem within a factor $1 + \epsilon$.)

Soon after the circulation of the draft of this paper, Mario Szegedy and Madhu Sudan independently discovered a reduction from a weaker subclass of PCP (namely, the one where the decision time of the verifier is small) to MAX-3SAT, a MAX-SNP-complete problem. Our proof of Theorem 2 implies that NP is contained in this weaker subclass. Hence their reduction — a precursor of the reduction that later appears in [ALM+92] — shows the hardness of approximating MAX-3SAT within a factor $(1 + \epsilon)$, where ϵ depends upon the decision time of the verifier in our result. Further, this ϵ becomes a constant if the decision time could be reduced to a constant (equivalently, when the number of query bits is constant). This reduction was reported in [AMS+92], alongwith the result that approximating any MAX-SNP-complete problem to within a factor of $1 - \frac{1}{(\log \log n)^{O(1)}}$ is NP-hard.

Shortly afterwards, it was shown in [ALM+92] that NP = PCP$(\log n, 1)$. Note that this characterization of NP in terms of PCP is optimal up to constant factors if P $\neq$ NP (see our remarks following Theorem 1).

The techniques of [ALM$^+$92] are similar to ours and rely heavily on our Composition Lemma (Lemma 5). Their first step is to construct a new verifier that improves our basic verifier of Section 4.2, in that it reads only $O(1)$ entries from the table provided in the proof, and has polylogarithmic decision time. One crucial component in constructing such a verifier is an efficient low-degree test based upon our main technical lemma (Lemma 16) and the work of Rubinfeld and Sudan [RS92]. Another important idea is that of *parallelization*, introduced in the work of Lapidot and Shamir [LS91] and Feige and Lovász [FL92]. The second step in [ALM$^+$92] is to compose the verifier of the first step with itself. This gives a verifier that reads $O(1)$ entries from the proof, but now the entries are of size $O(\log\log n)$. The third step is to compose the verifier of the second step with a new verifier, whose construction used some ideas in [BLR90]. This new verifier is inefficient in the number of random bits it uses (this number is polynomial in the proof size), but it queries only $O(1)$ bits from the proof. The verifier obtained through the above composition is $(\log n, 1)$-restricted.

Since [ALM$^+$92] a sequence of papers [PS, BGLR93, BS94, BGS95, H96] have constructed more and more efficient (in terms of constant factors) $(\log n, 1)$-restricted verifiers for SAT. The latest verifier [H97] needs only 3 query bits[8]. (Some of our ideas from Section 4.4 regarding the "concatenation property" were useful in achieving some of this efficiency.) In a different direction, a recent paper [PS94] improves existing verifiers by reducing the size of the proof required by this verifier to $n^{1+\epsilon}$. (This had not been achieved in [ALM$^+$92] because of the huge field size required by our Lemma 16, which is crucial to [ALM$^+$92]. A centerpiece of [PS94] is a better proof for Lemma 16 — more correctly, of the subcase of the lemma when m, the number of variables, is 2.) In another, even more recent work, Raz and Safra [RS97] prove an important generalization of NP = PCP$(\log n, 1)$ in which the proof is no longer a bit string but a string over an alphabet of size 2^a (i.e., each symbol is represented using a bits). The verifier is allowed to use $O(\log n)$ random bits, read $O(1)$ symbols from the proof, and must reject with probability $2^{-\Theta(a)}$ if the input is not in language L. The paper shows how to construct such verifiers for $a \leq \log^{1-\epsilon} n$. It also generalizes our low degree test in an important way (see also [ASu97]).

Problems other than Clique and MAX-SNP problems have also been shown hard to approximate. These include Chromatic-Number and Set-Cover (Lund and Yannakakis [LY94]; see also [BGLR93, KLS93]) and problems on lattices, codes, and linear systems (Arora, Babai, Stern and Sweedyk [ABSS93]). Independently of these works, other hardness results for a variety of approximation problems were obtained by Bellare [Bel93], Bellare and Rogaway [BR93], and Zuckerman ([Zuc93]). We refer the reader to [AL96] for a survey.

The NP-hardness result for approximating clique has been steadily improved. A consequence of NP = PCP$(\log n, 1)$ is that approximating clique number within a factor n^ϵ is NP-hard, for some fixed $\epsilon > 0$. A sequence of improvements culminating in Hastad [H96] shows that even approximating within a factor $n^{1-\epsilon}$ is hard if NP $\not\subseteq BPP$.

Lastly, we mention that PCP-style characterizations have been provided for other complexity classes as well, such as PSPACE (Condon et al., [CFLS93]) and PH (Kiwi et al., [KLR$^+$94]).

Acknowledgments

We thank Mike Luby for many discussions in the early stages of this work; specifically, his lower bound on the complexity of the protocol as presented in [FGL$^+$91] is what started this work. We also thank Ron Fagin, Oded Goldreich, Noam Nisan, Madhu Sudan, Steven Phillips, Mario Szegedy, Umesh Vazirani and Moshe Vardi for many insightful discussions. Numerous comments of Yuval Shahar and the anonymous referees helped us improve the clarity of our presentation.

[8]The definition of proof-checking is slightly different from ours, and allows a small probability of error when $x \in L$

References

[AKS87] M. Ajtai, J. Komós, and E. Szemeredi. Deterministic simulation in logspace. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 132–140, 1987.

[AGHP92] N. Alon, O. Goldreich, J. Hastad, R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, **3**:3 pp 289–304. 1992.

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 724–733, 1993.

[AL96] S. Arora and C. Lund. Hardness of approximations. Chapter 10 in *Approximation Algorithms for NP-hard Problems*, Dorit Hochbaum, editor. PWS Publishing, 1996.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 13–22, 1992.

[AMS+92] S. Arora, R. Motwani, M. Safra, M. Sudan, and M. Szegedy. PCP and approximation problems. Manuscript, 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1992.

[ASu97] S. Arora and M. Sudan. Improved low degree testing and its applications. In *Proc. 29th Annual ACM Symp. on Theory of Computing*, pages 496–505, 1997

[Bab85] L. Babai. Trading group theory for randomness. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 421–429, 1985.

[Bel93] M. Bellare. Interactive Proofs and approximation: Reductions from two provers in one round. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems*. IEEE Computer Press, 1993. Preliminary version: IBM Research Report RC 17969 (May 1992).

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. (Preliminary version in *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990.)

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 21–31, 1991.

[BGKW88] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi prover interactive proofs: How to remove intractability assumptions. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 113–121, 1988.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient multi-prover interactive proofs with applications to approximation problems. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 113–131, 1993.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan. Free bits and nonapproximability. In *Proc. 27th ACM Symp. on Theory of Computing*, 1995. to appear.

[BH92] R. Boppana and M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 86–97, 1989.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 73–83, 1990.

[BR93] M. Bellare and P. Rogaway. The complexity of approximating non-linear programs. In P.M. Pardalos, editor, *Complexity of Numerical Optimization*. World Scientific, 1993. Preliminary version: IBM Research Report RC 17831 (March 1992).

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 184–193, 1994.

[BW] E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470.

[CC+94] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Hastad, D. Ranjan, and P. Rohatgi. The random oracle hypothesis is false. *Journal of Computer and System Sciences*, **49**:1, 1994.

[CG89] B. Chor and O. Goldreich. On the power of two-point based sampling. *Journal of Complexity*, **5**:96–106, 1989.

[CFLS93] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Random debaters and the hardness of approximating stochastic functions. In *Proc. of the 9th Structure in Complexity Theory Conference*, pages 280–293, 1993. Also available as DIMACS Techreport TR 93-79.

[CL89] A. Condon and R. Ladner. On the complexity of space bounded interactive proofs. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 462–467, 1989.

[Con93] A. Condon. The complexity of the max-word problem and the power of one-way interactive proof systems. *Computational Complexity*, 3:292–305, 1993.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In Richard Karp, editor, *Complexity of Computer Computations*, pages 43–73. AMS, 1974.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1991.

[FL92] U. Feige and L. Lovász. Two-prover one-round proof systems: Their power and their problems. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 733–741, 1992.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Proceedings of the 3rd Conference on Structure in Complexity Theory*, pages 156–161, 1988.

[FS88] L. Fortnow and M. Sipser. Are there interactive protocols for co-np languages? *Information Processing Letters*, 28:249–251, 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *SIAM J. of Computation*, 18:186–208, 1989. (Preliminary version in *Proc. ACM Symposium on Theory of Computing*, 1985.)

[H96] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th IEEE Symposium on Foundations of Computer Science*, pages 627–636, 1996.

[H97] J. Hastad. Some optimal inapproximability results. In *Proc. 29th Annual ACM Symp. on Theory of Computing*, pages 1–10, 1997

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In Miller and Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KLR+94] M. Kiwi, C. Lund, A. Russell, D. Spielman, and R. Sundaram. Interaction and alternation. In *Proceedings of the 9th Structure in Complexity Theory Conference*, 1994.

[KLS93] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems, ISTCS*, pages 250–260. IEEE Computer Society Press, 1993.

[Lev73] L. Levin. Universal'nyĭe perebornyĭe zadachi (universal search problems : in Russian). *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[Lip89] R. Lipton. Efficient checking of computations. In *Proceedings of 6th STACS*, 1989.

[LS91] D. Lapidot and A. Shamir. Fully parallelized multi prover protocols for NEXPTIME. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 13–18, 1991.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. *Siam J. Computing*, **22**:838–856, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[PS] S. Phillips and S. Safra. PCP and tighter bounds for approximating MAX-SNP. Manuscript, April 1992.

[PS94] A. Polishchuk and D. Spielman. Nearly linear size holographic proofs. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 194–203, 1994.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test and a sub-constant error-probability PCP. In *Proc. 29th Annual ACM Symp. on Theory of Computing*, pages 475–484, 1997

[RS92] R. Rubinfeld and M. Sudan. Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 23–32, 1992.

[Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992. Prelim. version in 1990 FOCS, pages 11–15.

[She91] A. Shen. Multilinearity test made easy. Manuscript, 1991.

[Sud92] M. Sudan. *Efficient checking of polynomials and proofs and the hardness of approximation problems.* PhD thesis, U.C. Berkeley, 1992.

[Zuc91] D. Zuckerman. Simulating BPP using a general weak random source. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 79–89, 1991.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *8th Structure in Complexity Theory Conf.*, pages 305–312, 1993.

Appendix

We include the statements/proofs of some simple facts assumed in the paper.

Fact 21 *For every set of k (point, value) pairs $\{(a_i, b_i) : 1 \leq i \leq k\}$, (where $a_i, b_i \in F$ and the a_i's are distinct), there is a unique polynomial $p(x)$ of degree $k-1$ such that*

$$p(a_i) = b_i.$$

Proof: Let

$$L_i(x) = \prod_{j \neq i} \frac{(x - a_j)}{a_i - a_j}$$

be the polynomial that is 1 at a_i and zero at all a_j for $j \neq i$. Then the desired polynomial p is given by

$$p(x) = \sum_{i \leq k} b_i L_i(x).$$

Uniqueness is easy to verify. $\square$

Fact 22 (Schwartz) *An m-variate polynomial of degree d is 0 at no more than md/q fraction of points in F^m, where $q = |F|$.*

Proof: Proved by induction on m. Truth for $m = 1$ is clear, since a univariate degree-d polynomial has at most d roots.

A degree-d polynomial $p(x_1, \ldots, x_m)$ has a representation as

$$\sum_{i=0}^{k} x_1^i \cdot p_i(x_2, \ldots, x_m) \tag{42}$$

where $k \leq d$, each $p_i(x_2, \ldots, x_m)$ is a $(m-1)$-variate polynomial of degree at most d, and $p_k(x_2, \ldots, x_m)$ is a nonzero polynomial.

By the inductive hypothesis, $p_k(x_2, \ldots, x_m) = 0$ for at most $d(m-1)/q$ fraction of values of $(x_2, \ldots, x_m) \in \mathrm{F}^{m-1}$. For any other value of $(x_2, \ldots, x_m)$, the expression in Equation (42) is a degree k polynomial in x_1, and so is zero for at most k values of x_1.

Hence the fraction of values of $(x_1, \ldots, x_m) \in \mathrm{F}^m$ where p is zero is at most $d(m-1)/q + k/q \leq dm/q$. $\Box$

Now we prove a lemma that was used in Section 4.2.1.

Lemma 23 *["Zero-tester" Polynomials, [BFLS91, FGL$^+$91]] Let $F = GF(q)$ and integers m, h satisfy $32mh < q$. Then there exists a family of q^{4m} polynomials $\{R_1, R_2, \ldots\}$ in $F_h[x_1, \ldots, x_{4m}]$ such that if $f : [0,h]^{4m} \to F$ is any function not identically 0, then if R is chosen randomly from this family,*

$$\Pr[\sum_{y \in [0,h]^{4m}} R(y) f(y) = 0] \leq \frac{1}{8}. \tag{43}$$

This family is constructible in $q^{O(m)}$ time.

Proof: In this proof we will use the symbols $0, 1, \ldots, h$ to denote both integers in $\{0, \ldots, h\}$, and field elements. We use boldface to denote the latter usage. Thus for example $\mathbf{0} \in \mathrm{F}$. (Furthermore, $0, 1, \ldots, h$ refer to integers only when they appear in the exponent.)

For now let $t_1, \ldots, t_{4m}$ be formal variables (later we give them values). Consider the following degree h polynomial in $t_1, \ldots, t_{4m}$ with coefficients in F.

$$\sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \prod_{j=1}^{4m} t_j^{i_j}. \tag{44}$$

This polynomial is the zero polynomial iff f is identically $\mathbf{0}$ on $[\mathbf{0}, \mathbf{h}]^{4m}$. Further, if it is not the zero polynomial then by Fact 7 its roots constitute a fraction no more than $4hm/q$ of all points in F^{4m}. Assume that this fraction is less than 1/8.

We construct a family $\{R_{b_1, \ldots, b_{4m}} : b_1, \ldots, b_{4m} \in \mathrm{F}\}$ of q^{4m} polynomials, such that

$$\sum_{(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \in [\mathbf{0}, \mathbf{h}]} R_{b_1, \ldots, b_{4m}}(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) = \mathbf{0} \tag{45}$$

iff $(b_1, \ldots, b_{4m})$ is a root of the polynomial in (44). This will prove the lemma.

Denote by $I_{t_i}(x_i)$ the univariate degree-h polynomial in x_i whose coefficients are polynomials in t_i and whose values at $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{h} \in \mathrm{F}$ are $\mathbf{1}, t_i, \ldots, t_i^h$ respectively (such a polynomial exists and has degree h in t_i; see the proof of Fact 21 in the appendix).

Let g be the following polynomial in variables $x_1, \ldots, x_{4m}, t_1, \ldots, t_{4m}$.

$$g(t_1, \ldots, t_{4m}, x_1, \ldots, x_{4m}) = \prod_{i=1}^{4m} I_{t_i}(x_i).$$

Then for $\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]$ we have

$$g(t_1, \ldots, t_{4m}, \mathbf{i_1}, \ldots, \mathbf{i_{4m}}) = \prod_{j=1}^{4m} I_{t_j}(\mathbf{i_j}) = \prod_{j=1}^{4m} {t_j}^{i_j},$$

and so

$$\sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}})\, g(t_1, \ldots, t_m, \mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}})$$

$$= \sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \prod_{j=1}^{4m} t_j^{i_j}$$

Now for $b_1, \ldots, b_{4m} \in \mathrm{F}$ define $R_{b_1, \ldots, b_{4m}}$ as the polynomial obtained by substituting $t_1 = b_1, \ldots, t_{4m} = b_{4m}$ in g.

$$R_{b_1, \ldots, b_{4m}}(x_1, \ldots, x_{4m}) = g(b_1, \ldots, b_{4m}, x_1, \ldots, x_{4m}).$$

This family of polynomials clearly satisfies the property that (45) holds iff $(b_1, \ldots, b_{4m})$ is a root of the polynomial in (44). Hence the Lemma is proved. $\square$

Remark: An alternative proof of Lemma 10 uses the notion of ϵ-biased random variables [NN93, AGHP92].

Example 3 We give an example to illustrate Lemma 23. We write a polynomial g for checking the sums of functions on $[\mathbf{0}, \mathbf{h}]^p$ for $\mathbf{h} = \mathbf{1}$ and $p = 2$.

$$g(t_1, t_2, x_1, x_2) = (\mathbf{1} + x_1(t_1 - \mathbf{1}))(\mathbf{1} + x_2(t_2 - \mathbf{1})). \tag{46}$$

Hence we have

$$\sum_{x_1, x_2 \in [\mathbf{0}, \mathbf{1}]^2} f(x_1, x_2) g(t_1, t_2, x_1, x_2) = f(\mathbf{0}, \mathbf{0}) + f(\mathbf{1}, \mathbf{0}) t_1 + f(\mathbf{0}, \mathbf{1}) t_2 + f(\mathbf{1}, \mathbf{1}) t_1 t_2. \tag{47}$$

Clearly, the polynomial in (47) is nonzero if any of its four terms is nonzero. $\square$

Fact 24 *Let $A = (a_{ij})$ be an $n \times n$ matrix, where the entries a_{ij} are considered as variables. Then the determinant of A is a polynomial of degree n in the a_{ij}'s.*

Proof: Follows from inspecting the expression for the determinant.

$$\det(A) = \sum_{\sigma \in S_n} (-1)^{\mathrm{sgn}(\sigma)} \cdot \prod_{i \leq n} a_{i\sigma(i)},$$

where S_n is the set of all permutations of $\{1, \ldots, n\}$. $\square$

The following fact is used in the proof of Lemma 20in Section 5.2. The reader may choose to read it by mentally substituting "$\mathrm{F}[x_1, \ldots, x_m]$, the set of polynomials over field F in the formal variables $x_1, \ldots, x_m$" in place of "integral domain R."

Fact 25 (Cramer's Rule) *Let A be an $m \times n$ matrix whose entries come from an integral domain R, and $m > n$. Let $A \cdot z = 0$ be a system of m equations in n variables (note: it is an overconstrained homogeneous system).*

1. *The system has a non-trivial solution iff all $n \times n$ submatrices of A have determinant 0.*

2. If the system has a nontrivial solution, then it has one of the type $(z_1 = t_1, \ldots, z_n = t_n)$ where each t_i is a sum of determinants of submatrices of A.

Proof: (Part 1) If some $n \times n$ submatrix has nonzero determinant, then the coefficient vectors are independent and so cannot have a nontrivial combination summing to 0. Conversely, if a nontrivial combination of the coefficient vectors exists, then they are dependent and therefore every $n \times n$ submatrix must have zero determinant.

(Part 2) Our proof mimics the usual method of solving equations over fields. We need to be careful, however, since this method involves matrix inversion, which involves division. Division is not a well-defined operation over an integral domain. Therefore we will use the fact that for a square matrix M,

$$\det(M) \cdot M^{-1} = adj(M),$$

where $adj(M)$ is a square matrix whose each entry is a determinant of some submatrix of M. In other words, $adj(M)$ is well-defined over an integral domain.

We solve the system $A \cdot z = 0$ as follows. Let B be the largest nonsingular square submatrix of A. Suppose B is $(n-l) \times (n-l)$ for $l \geq n-m-1$. W.l.o.g. assume that the first l columns are not in B. Hence A looks like

$$A = \begin{pmatrix} C & B \\ E & D \end{pmatrix}$$

where we assume w.l.o.g. that $C \neq 0$. Let $u = (z_1, \ldots, z_l)$ and $y = (z_{l+1}, \ldots, z_n)$. Then for every value of $u \in R^l$, the system $B \cdot y = -C \cdot u$ can be solved for y.

We take any vector $\hat{u} \in R^l$ such that $C \cdot \hat{u} \neq 0$, and solve the following system for y.

$$B \cdot y = -C \cdot (det(B) \cdot \hat{u}).$$

Let $\hat{y} = -det(B) \cdot B^{-1}(C \cdot \hat{u})$ be a solution. Note that $\hat{y}$ is well-defined since $det(B) \cdot B^{-1}$ involves no division. Hence our solution is of the form $z = (det(B) \cdot \hat{u},\ \ \hat{y})$. Every coordinate of $det(B) \cdot \hat{u}$ is a multiple of $det(B)$ and every coordinate of $\hat{y}$ is a sum of entries of $adj(B)$. But each entry of $adj(B)$ is a determinant of some submatrix of B. Hence the claim is proved. □

Finally, we state the result in [FGL$^+$91] that was used in the proof of Corollary 3. First we need to define the *soundness* of a verifier. A verifier checks membership proofs for a language with soundness p if for every input that is in the language, the verifier accepts some proof with probability 1, and for every input that is not in the language, the verifier rejects every proof with probability $1 - 1/p$.

Theorem 26 ([FGL$^+$91]) *For integer-valued functions r, q, p, suppose there is a $(r(n), q(n))$-restricted verifier that checks membership proofs for SAT with soundness $p(n)$.*

Then for every integer n, there is a reduction from SAT instances of size n to graphs of size $2^{O(r(n))}q(n)$, such that the clique number of the graph when the SAT instance is satisfiable is a factor $p(n)$ bigger than the clique number when the instance is not satisfiable.

□