Electronic Colloquium on Computational Complexity, Revision 3 of Report No. 137 (2010)

IP = PSPACE using Error Correcting Codes*

Or Meir†

Abstract

The **IP** theorem, which asserts that **IP** = **PSPACE** (Lund et. al., and Shamir, in J. ACM 39(4)), is one of the major achievements of complexity theory. The known proofs of the theorem are based on the arithmetization technique, which transforms a quantified Boolean formula into a related polynomial. The intuition that underlies the use of polynomials is commonly explained by the fact that polynomials constitute good error correcting codes. However, the known proofs seem tailored to the use of polynomials, and do not generalize to arbitrary error correcting codes.

In this work, we show that the **IP** theorem can be proved by using general error correcting codes. We believe that this establishes a rigorous basis for the aforementioned intuition, and sheds further light on the **IP** theorem.

1 Introduction

The **IP** theorem [LFKN92, Sha92] asserts that **IP** = **PSPACE**, or in other words, that any set in **PSPACE** has an interactive proof. This theorem is fundamental to our understanding of both interactive proofs and polynomial space computations. In addition, it has important applications, such as the existence of program checkers for **PSPACE**-complete sets, and the existence of zero knowledge proofs for every set in **PSPACE**. Indeed, the theorem is one of the major achievements of complexity theory. We note that an additional proof of the **IP** theorem has been suggested by [She92], and also that the work of [GKR08] implicitly gives an alternative proof of the **IP** theorem.

The known proofs of the **IP** theorem go roughly along the following lines: Suppose that we are given a claim that can be verified in polynomial space, and we are required to design an interactive protocol for verifying the claim. We begin by expressing the claim as a quantified Boolean formula, using the **PSPACE**-completeness of the **TQBF** problem. Then, we "arithmetize" the formula, transforming it into a claim about the value of a particular arithmetic expression. Finally, we use the celebrated sum-check protocol in order to verify the value of the arithmetic expression. One key point is that the sum-check protocol employs the fact that certain restrictions of the arithmetic expression are low-degree polynomials.

While the arithmetization technique used in the proof turned out to be extremely useful, it seems somewhat odd that one has to go through algebra in order to prove the theorem, since the theorem itself says nothing about algebra. The intuition behind the use of algebra in the proof is usually explained by the fact that low-degree polynomials constitute good error correcting codes.

In order to demonstrate this intuition, let us consider the special case of proving that co**NP** $\subseteq$ **IP**, which amounts to designing a protocol for verifying that a given Boolean formula has no satisfying

*This research was supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100 Israel. Email: or.meir@weizmann.ac.il.

ISSN 1433-8092

assignments. In this case, the main difficulty that the verifier faces is that it has to distinguish between a formula that has no satisfying assignments and a formula that has only one satisfying assignment. If we consider the truth tables of those formulas, then the verifier has to distinguish two exponential-length strings that differ only on at one coordinate, which seems difficult to do in polynomial time. However, if the verifier could access an encodings of the truth tables via an error correcting code, then its task would have been easy: An error correcting code has the property that any two distinct strings are encoded by strings that differ on many coordinates, even if the original strings were very close to each other. Therefore, if the verifier could access an error-correcting encoding of the truth table of the formula, it could just pick a random coordinate of the encoding and check that it matches the encoding of the all-zeroes truth table.

The role of algebra in the proof of the **IP** theorem is now explained as follows: The arithmetization technique transforms the formula into a low-degree polynomial. Since low-degree polynomials are good error correcting codes, the arithmetization should have a similar effect to that of encoding the truth table of the formula via an error correcting code. Morally, this should help the verifier in distinguishing between satisfiable and unsatisfiable formulas.

While the above intuition is very appealing, it is not clear what is the relation between this intuition and the actual proof, and whether the actual proof indeed implements this intuition. In particular, the polynomial that is obtained from the arithmetization of a formula *is not the encoding of the formula's truth table by the corresponding polynomial code*[1], but rather an arbitrary polynomial that agrees with the formula on the Boolean hypercube. Furthermore, the known proofs of the **IP** theorem use algebraic manipulations that can not be applied to general error correcting codes. Those considerations give raise to the natural question of whether the foregoing intuition is correct or not. In other words, we would like to know whether the error correcting properties of polynomials are indeed the crux of the proof of the **IP** theorem, or are there other properties of polynomials that are essential to the proof.

In this work, we show that the **IP** theorem can actually be proved by using only error correcting codes, while making no reference to polynomials. We believe that this establishes a rigorous basis for the aforementioned intuition. While our proof is somewhat more complicated than the previous proofs of the **IP** theorem, we believe that it is valuable as it explains the role of error correcting codes in the **IP** theorem.

Our techniques. Our work relies heavily on the notion of tensor product of codes, which is a classical operation on codes. The tensor product operation generalizes the process of moving from univariate polynomials to multivariate polynomials, in the sense that if we view univariate polynomials as error correcting codes, then multivariate polynomials are obtained by applying the tensor product operation to univariate polynomials. We refer to error correcting codes that are obtained via the tensor product operation as "tensor codes".

Our first main observation is the following. Recall that in the proof of the **IP** theorem, the sum-check protocol is applied to multivariate polynomials. We show that the sum-check protocol can in fact be applied to any tensor code. Specifically, we note that tensor codes have the following property: A codeword c of a tensor code can be viewed as a function from some hypercube $[\ell]^m$ to a finite field $\mathbb{F}$, such that if a function $f : [\ell] \to \mathbb{F}$ is defined by an expression of the form

$$f(x_i) = \sum_{x_{i+1}} \cdots \sum_{x_m} c\left(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_m\right)$$

[1]In other words, the polynomial generated by the arithmetization is not the low-degree extension of the truth table. To see this, note that the arithmetization of an unsatisfiable formula may produce a non-zero polynomial.

then f is a codeword of some other error correcting code. We observe that this is the only property that is required for the sum-check protocol to work, and therefore the protocol can be used with any tensor code. In other words, the essential property of multivariate polynomials that is used in the sum-check protocol is the fact that multivariate polynomials are tensor codes.

Our next step is to use the foregoing observation to prove that co**NP** $\subseteq$ **IP** without using polynomials. To this end, we replace the multivariate polynomials used in the proof with general tensor codes. In particular, we replace the polynomial that is obtained from the arithmetization with a tensor codeword that agrees with the formula on the Boolean hypercube. We perform this replacement by generalizing the arithmetization technique to work with general error correcting codes instead of polynomials. This generalization is done by constructing "multiplication codes", which are error correcting codes that emulate polynomial multiplication, and may be of independent interest.

Finally, we consider the proof of the full **IP** theorem, i.e, **IP** = **PSPACE**. To this end, we devise a protocol for verifying the validity of a quantified Boolean formula. In the known proofs of the **IP** theorem, when considering quantified Boolean formulas we encounter the following obstacle: The arithmetization of a quantified formula results in an arithmetic expression that contains polynomials of very high degree, and not low degree as required by the sum-check protocol. This issue translates in our proof to certain limitations of the aforementioned multiplication codes.

Recall that the proofs of the **IP** theorem by [Sha92, She92] resolve the foregoing issue by performing algebraic manipulations on the arithmetic expression to ensure that the involved polynomials are of low degree. Obviously, such a solution can not applied in our setting. Instead, we build on an idea from [GKR08], which shows that one can use the sum-check protocol to reduce the degree of the polynomials. While their technique still uses the algebraic structure of polynomials, we show that this technique can be adapted to our setting, allowing us to show that **IP** = **PSPACE**.

The adaptation of [GKR08] is done by generalizing the sum-check protocol, and observing that it can be used to reduce the task of evaluating a coordinate of a tensor codeword to the task of evaluating a coordinate of another tensor codeword. This generalization may be of independent interest.

The organization of this paper. In Section 2, we review the basic notions of error correcting codes and define the notation that we use. In Section 3, we review the notion of tensor product codes, and introduce the notion of multiplication codes. In Section 4, we revisit the sum-check protocol and generalize it. In Section 5, we prove that co**NP** $\subseteq$ **IP**, and along the way present our generalization of the arithmetization technique. Finally, in Section 6, we prove the full **IP** theorem.

Remark regarding algebrization. Recall that the **IP** theorem is a classical example for a non-relativizing result. Recently, [AW08] suggested a framework called "algebrization" as a generalization of the notion of relativization, and showed that the **IP** theorem relativizes in this framework, or in other words, the **IP** theorem "algebrizes". We note that while our proof of the **IP** theorem does not seem to algebrize, one can generalize the algebrization framework to include our proof as well. Some details are given in a remark at the end of Section 5.

2 Preliminaries

For any $n \in \mathbb{N}$ we denote $[n] \stackrel{\text{def}}{=} \{0, 1 \ldots, n-1\}$ - note that this is a *non-standard* notation. Similarly, if x is a string of length n over any alphabet, we denote its set of coordinates by $[n]$, and in particular, *the first coordinate will be denoted* 0.

Throughout the paper, we will refer to algorithms that take as input a finite field $\mathbb{F}$. We assume that the finite field $\mathbb{F}$ is represented, say, by a list of its elements and the corresponding addition and multiplication tables.

For any two strings x, y of equal length n and over any alphabet, the relative Hamming distance between x and y is the fraction of coordinates on which x and y differ, and is denoted by $\delta(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|/n$.

All the error correcting codes that we consider in this paper are *linear codes*, to be defined next. Let $\mathbb{F}$ be a finite field, and let $k, \ell \in \mathbb{N}$. A (linear) code C is a linear one-to-one function from $\mathbb{F}^k$ to $\mathbb{F}^\ell$, where k and ℓ are called the code's message length and block length, respectively. We will sometimes identify C with its image $C(\mathbb{F}^k)$. Specifically, we will write $c \in C$ to indicate the fact that there exists $x \in \mathbb{F}^k$ such that $c = C(x)$. In such case, we also say that c is a codeword of C. The relative distance of a code C is the minimal relative Hamming distance between two different codewords of C, and is denoted by $\delta_C \stackrel{\text{def}}{=} \min_{c_1 \neq c_2 \in C} \{\delta(c_1, c_2)\}$.

Due to the linearity of C, there exists an $n \times k$ matrix G, called the generator matrix of C, such that for every $x \in \mathbb{F}^k$ it holds that $C(x) = G \cdot x$. Observe that given the generator matrix of C one can encode messages by C as well as verify that a string in $\mathbb{F}^\ell$ is a codeword of C in time that is polynomial in ℓ. Moreover, observe that the code C always encodes the all-zeroes vector in $\mathbb{F}^k$ to the all-zeroes vector in $\mathbb{F}^\ell$.

We say that C is systematic if the first k symbols of a codeword contain the encoded message, that is, if for every $x \in \mathbb{F}^k$ it holds that $(C(x))_{|[k]} = x$. By applying Gaussian elimination to the generator matrix of C, we may assume, without loss of generality, that C is systematic.

The following fact asserts the existence of (rather weak) linear codes. Such codes are all we need for this paper.

Fact 2.1. *The exists an algorithm that when given as input $k \in \mathbb{N}$ and $\delta \in (0,1)$ and a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq \text{poly}(1/(1-\delta))$, runs in time that is polynomial in k, $\log|\mathbb{F}|$, and $1/(1-\delta)$, and outputs the generator matrix of a linear code C over $\mathbb{F}$ that has message length k, block length $\ell \stackrel{\text{def}}{=} k/\text{poly}(1-\delta)$, and relative distance at least δ.*

Fact 2.1 can be proved via a variety of techniques from coding theory, where many of them do not use polynomials (see, e.g., [Var57, ABN+92, GI05][2]).

3 Tensor Product Codes and Multiplication Codes

In this section we review the notion of tensor product of codes (in Section 3.1) and introduce the notion of multiplication codes (in Section 3.2). We note that while the tensor product is a standard operation in coding theory, and a reader who is familiar with it may skip Section 3.1, with the exception of Propositions 3.7 and 3.8 which are non-standard. On the other hand, the notion of multiplication codes is a non-standard notion that we define for this work (though it may be seen as a variant of the notion of error correcting pairs, see [Köt92, Pel92, Sud01, Lect. 11 (1.4)]).

3.1 Tensor Product of Codes

In this section we define the tensor product operation on codes and present some of its properties. See [MS88] and [Sud01, Lect. 6 (2.4)] for the basics of this subject.

[2]We note that the work of [GI05] does make use of polynomials, but this use of polynomials can be avoided at the expense of having somewhat worse parameters, which we can still afford in this work. Also, we note that the work of [ABN+92] requires $|\mathbb{F}| \geq \exp(1/(1-\delta))$, but this limitation can be waived by means of concatenation.

Definition 3.1. Let $R : \mathbb{F}^{k_R} \to \mathbb{F}^{\ell_R}$, $C : \mathbb{F}^{k_C} \to \mathbb{F}^{\ell_C}$ be codes. The tensor product code $R \otimes C$ is a code of message length $k_R \cdot k_C$ and block length $\ell_R \cdot \ell_C$ that encodes a message $x \in \mathbb{F}^{k_R \cdot k_C}$ as follows: In order to encode x, we first view x as a $k_C \times k_R$ matrix, and encode each of its rows via the code R, resulting in a $k_C \cdot \ell_R$ matrix x'. Then, we encode each of th columns of x' via the code C. The resulting $\ell_C \times \ell_R$ matrix is defined to be the encoding of x via $R \otimes C$.

The following fact lists some of the basic and standard properties of the tensor product operation.

Fact 3.2. *Let $R : \mathbb{F}^{k_R} \to \mathbb{F}^{\ell_R}$, $C : \mathbb{F}^{k_C} \to \mathbb{F}^{\ell_C}$ be codes. We have the following:*

1. *An $\ell_C \times \ell_R$ matrix x over $\mathbb{F}$ is a codeword of $R \otimes C$ if and only if all the rows of x are codewords of R and all the columns of x are codewords of C.*
2. *Let δ_R and δ_C be the relative distances of R and C respectively. Then, the code $R \otimes C$ has relative distance $\delta_R \cdot \delta_C$.*
3. *The tensor product operation is associative. That is, if $D : \mathbb{F}^{k_D} \to \mathbb{F}^{\ell_D}$ is a code then $(R \otimes C) \otimes D = R \otimes (C \otimes D)$.*

The following standard feature of tensor codes will be very useful.

Fact 3.3. *Let R and C be as before and let $r \in R$ and $c \in C$. Define the tensor product $r \otimes c$ of r and c as the $\ell_C \times \ell_R$ matrix defined by $(r \otimes c)_{i,j} = c_i \cdot r_j$. Then, $r \otimes c$ is a codeword of $R \otimes C$.*

Proof Observe that each row of $r \otimes c$ is equal to r multiplied by a scalar, and therefore it is a codeword of R. Similarly, each column of $r \otimes c$ is a codeword of C. By Item 1 of Fact 3.2, it follows that $r \otimes c \in R \otimes C$, as required. ■

The associativity of the tensor product operation allows us to use notation such as $C \otimes C \otimes C$, and more generally:

Notation 3.4. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code. For every $m \in \mathbb{N}$ we denote by $C^m : \mathbb{F}^{k^m} \to \mathbb{F}^{\ell^m}$ the code $\underbrace{C \otimes C \otimes \ldots \otimes C}_{m}$. Formally, $C^m = C^{m-1} \otimes C$.

Notation 3.5. When referring to the code C^m and its codewords, we will often identify the sets of coordinates $[k^m]$ and $[\ell^m]$ with the hypercubes $[k]^m$ and $[\ell]^m$ respectively. Using the latter identification, one can view a string $x \in \mathbb{F}^{k^m}$ as a function $x : [k]^m \to \mathbb{F}$, and view strings in $\mathbb{F}^{\ell^m}$ similarly. With a slight abuse of notation, we say that C^m is systematic if for every codeword $c \in C^m$, the restriction of c to $[k]^m$ equals the message encoded by c^m. It is easy to see that if C is systematic (in the usual sense), then C^m is systematic as well.

Using Fact 3.2, one can prove by induction the following.

Fact 3.6. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code. Then, the codewords of C^m are precisely all the functions $f : [\ell]^m \to \mathbb{F}$ such that the restriction of f to any axis-parallel line of the hypercube is a codeword of C. That is, a function $f : [\ell]^m \to \mathbb{F}$ is a codeword of C^m if and only if for every $1 \leq t \leq m$ and $i_1, \ldots, i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that the function $f(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ is a codeword of C.*

Less standard features. We turn to prove two less standard features of the tensor product operation that will be useful in Section 4. The following claim expresses a coordinate of a tensor codeword using an expression of a "sum-check" form. We will use this claim later to show that one can use the sum-check protocol to evaluate the coordinates of a tensor codeword.

Claim 3.7. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, and let $m \in \mathbb{N}$. Then, for every coordinate $(i_1, \ldots, i_m) \in [\ell]^m$ there exist scalars $\alpha_{t,j} \in \mathbb{F}$ (for every $1 \leq t \leq m$ and $j \in [k]$) such that for every codeword $c \in C^m$ it holds that*

$$c(i_1, \ldots, i_m) = \sum_{j_1 \in [k]} \alpha_{1,j_1} \cdot \sum_{j_2 \in [k]} \alpha_{2,j_2} \cdot \ldots \sum_{j_m \in [k]} \alpha_{m,j_m} \cdot c(j_1, \ldots, j_m)$$

Furthermore, the coefficients $\alpha_{t,j}$ can be computed in polynomial time from the tuple $(i_1, \ldots, i_m)$ and the generator matrix of C.

Proof By induction on m. Suppose that $m = 1$. In this case, c is a codeword of C. Let $i_1 \in [\ell]$. Since C is a linear function, it holds that $c(i_1)$ is a linear combination of the elements of the message encoded by c. Since C is systematic, it holds that $c(0), \ldots, c(k-1)$ are equal to the message encoded by c. Thus, we get that $c(i_1)$ is a linear combination of $c(0), \ldots, c(k-1)$, as required. Furthermore, the corresponding coefficients $\alpha_{1,j}$ are simply the corresponding row in the generator matrix of C.

We now assume that the claim holds for some $m \in \mathbb{N}$, and prove it for $m+1$. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, let $c \in C^{m+1}$, and let $(i_1, \ldots, i_{m+1}) \in [\ell]^{m+1}$ be a coordinate of c. We first observe that by Fact 3.6, it holds that $c(\cdot, i_2, \ldots, i_{m+1})$ is a codeword of C. Thus, by the same considerations as in the case of $m = 1$, it follows that there exist coefficients $\alpha_{1,j_1} \in \mathbb{F}$ for $j_1 \in [k]$ such that

$$c(i_1, \ldots, i_{m+1}) = \sum_{j_1 \in [k]} \alpha_{1,j_1} \cdot c(j_1, i_2, \ldots, i_{m+1})$$

Next, observe that Fact 3.6 implies that for every j_1, it holds that $c(j_1, \underbrace{\cdot, \ldots, \cdot}_{m})$ is a codeword of C^m. The induction hypothesis now implies that there exist coefficients $\alpha_{t,j} \in \mathbb{F}$ (for every $2 \leq t \leq m+1$ and $j \in [k]$) such that for every $j_1 \in [k]$ it holds that

$$c(j_1, i_2, \ldots, i_{m+1}) = \sum_{j_2 \in [k]} \alpha_{2,j_2} \cdot \ldots \sum_{j_{m+1} \in [k]} \alpha_{m+1,j_{m+1}} \cdot c(j_1, \ldots, j_{m+1})$$

Note that the latter coefficients $\alpha_{t,j}$ do not depend on j_1. It follows that

$$c(i_1, \ldots, i_{m+1}) = \sum_{j_1 \in [k]} \alpha_{1,j_1} \cdot \sum_{j_2 \in [k]} \alpha_{2,j_2} \cdot \ldots \sum_{j_{m+1} \in [k]} \alpha_{m+1,j_{m+1}} \cdot c(j_1, \ldots, j_{m+1})$$

as required. Furthermore, it is easy to see that the coefficients $\alpha_{t,j}$ can indeed be computed in polynomial time. ■

The following claim says that the intermediate sum that occurs in a single step of the sum-check protocol is a codeword of C. This is the key property used in each single step of the sum-check protocol.

Claim 3.8. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code, let $m \in \mathbb{N}$, and let $c \in C^m$. Then, for every sequence of scalars $\alpha_{t,j}$ (for every $2 \leq t \leq m$ and $j \in [\ell]$) it holds that the function $f : [\ell] \to \mathbb{F}$ defined by*

$$f(j_1) = \sum_{j_2 \in [\ell]} \alpha_{2,j_2} \cdot \sum_{j_3 \in [\ell]} \alpha_{3,j_3} \cdot \ldots \sum_{j_m \in [\ell]} \alpha_{m,j_m} \cdot c(j_1, \ldots, j_m)$$

is a codeword of C.

Proof The proof is by induction on m. For $m = 1$ the claim is trivial. We assume that the claim holds for some $m \in \mathbb{N}$, and prove it for $m + 1$. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code, let $c \in C^{m+1}$, and let $\alpha_{t,j}$ be scalars for every $2 \leq t \leq m + 1$ and $j \in [\ell]$. We wish to show that the function $f : [\ell] \to \mathbb{F}$ defined by

$$f(j_1) \stackrel{\text{def}}{=} \sum_{j_2 \in [\ell]} \alpha_{2,j_2} \cdot \ldots \sum_{j_{m+1} \in [\ell]} \alpha_{m+1,j_{m+1}} \cdot c(j_1, \ldots, j_{m+1})$$

is a codeword of C. To this end, let us observe that Fact 3.6 implies that for every $j_{m+1} \in [\ell]$, the function $g_{j_2} : [\ell]^m \to \mathbb{F}$ defined by

$$g_{j_{m+1}}(j_1, , \ldots, j_m) \stackrel{\text{def}}{=} c(j_1, \ldots, j_m, j_{m+1})$$

is a codeword of C^m. Therefore, by the induction hypothesis, the function $h_{j_{m+1}} : [\ell] \to \mathbb{F}$ defined by

$$h_{j_{m+1}}(j_1) \stackrel{\text{def}}{=} \sum_{j_2 \in [\ell]} \alpha_{2,j_2} \cdot \ldots \sum_{j_m \in [\ell]} \alpha_{m,j_m} \cdot g_{j_{m+1}}(j_1, \ldots, j_m)$$

is a codeword of C. Now, observe that we can express f as

$$f(j_1) = \sum_{j_{m+1} \in [\ell]} \alpha_{m+1,j_{m+1}} \cdot h_{j_{m+1}}(j_1)$$

In other words, it holds that f is a linear combination of codewords of C. By the linearity of C, it follows that f is a codeword of C. ■

3.2 Multiplication codes

The arithmetization technique, which transforms a Boolean formula into a low-degree polynomial, uses two basic properties of polynomials: The first property is that low-degree polynomials form a linear subspace. The second property is that the product of two low-degree polynomials is a low-degree polynomial (provided that the field is sufficiently large compared to the degree). Therefore, in order to generalize the arithmetization technique to use general error correcting codes, we would like to have error correcting codes with similar properties. The first property is attained by every linear code. The challenge is to obtain codes emulating the second "multiplication" property. To this end, we use the following notation.

Notation 3.9. Let $\mathbb{F}$ be a finite field, let $\ell \in \mathbb{N}$, and let $u, v \in \mathbb{F}^\ell$. Then, we denote by $u \cdot v$ the string in $\mathbb{F}^\ell$ defined by

$$(u \cdot v)_i = u_i \cdot v_i$$

We can now phrase the multiplication property of polynomials as follows. If c_1 and c_2 are codewords of polynomial codes (of sufficiently low degree), then $c_1 \cdot c_2$ is a codeword of a another polynomial code (of a higher degree). The following proposition shows that one can construct codes with such property without using polynomials.

Proposition 3.10. *For every $k \in \mathbb{N}$, $\delta \in (0, 1)$ and a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq \text{poly}\,(1/\,(1 - \delta))$, there exists a triplet (C_A, C_B, C_M) of systematic linear codes over $\mathbb{F}$ that have the following properties:*

1. ***Multiplication:*** *For every $c_A \in C_A$ and $c_B \in C_B$ it holds that $c_A \cdot c_B \in C_M$.*

2. *C_A and C_B have message length k, and C_M has message length k^2.*

3. *C_A, C_B, and C_M all have block length $\ell \stackrel{\text{def}}{=} k^2/\text{poly}\,(1-\delta)$, and relative distance δ.*

Furthermore, the exists an algorithm that when given as input k, δ, and $\mathbb{F}$, runs in time that is polynomial in k, $\log|\mathbb{F}|$, and $1/(1-\delta)$, and outputs the generating matrices of C_A, C_B and C_M.

Remark 3.11. Again, it is trivial to construct codes as in Proposition 3.10 using polynomials. Indeed, taking C_A, C_B, and C_M to be Reed-Solomon codes of appropriate degree would yield codes with the same multiplication property and with better parameters. The novelty of our proof of Proposition 3.10 is that the construction of the codes is based on generic codes, and not on polynomial codes. Specifically, we will only use the tensor product operation.

Proof The algorithm begins by invoking the algorithm of Fact 2.1 on input k, $\sqrt{\delta}$, and $\mathbb{F}$. This results in a code C with message length k, relative distance $\sqrt{\delta}$, and block length[3] $\ell_C = k/\text{poly}\left(1-\sqrt{\delta}\right) \leq k/\text{poly}\,(1-\delta)$. Next, the algorithm sets $\ell = \ell_C^2$ and constructs the generating matrices of the codes C_A, C_B, and C_M that are defined as follows:

1. The codewords of C_A are precisely all the $\ell_C \times \ell_C$ matrices c_A such that all the rows of c_A are identical and are equal to some codeword of C.

2. C_B is defined similarly to C_A, but with columns instead of rows.

3. The code C_M is the code C^2.

It is easy to see that C_A, C_B, and C_M have the required parameters and that their generating matrices can be constructed in polynomial time. It remains to show that for every $c_A \in C_A$ and $c_B \in C_B$ it holds that $c_A \cdot c_B \in C_M$. To this end, recall that c_A is an $\ell_C \times \ell_C$ matrix all of whose rows are equal to some codeword c_r of C, whereas c_B is an $\ell_C \times \ell_C$ matrix all of whose columns are equal to some codeword c_c of C. Finally, observe that $c_A \cdot c_B = c_r \otimes c_c$, so it follows by Fact 3.3 that $c_A \cdot c_B \in C^2 = C_M$, as required. ■

It is important to note that the multiplication of the codes of Proposition 3.10 is much more limited than the multiplication of polynomial codes. Specifically, the multiplication of polynomials can be applied many times. That is, if $c_1, \ldots, c_t$ are codewords of polynomial codes, then $c_1 \cdot \ldots \cdot c_t$ is also a codeword of a polynomial code, as long as the degrees of $c_1, \ldots, c_t$ are sufficiently small compared to t and $|\mathbb{F}|$. On the other hand, Proposition 3.10 only allows the multiplication of two codewords. This limitation is the reason that our emulation of the arithmetization technique in Section 5 is somewhat more complicated than the standard arithmetization.

Remark 3.12. It is possible to generalize the construction of Proposition 3.10 to allow multiplication of more codewords. However, the generalized construction yields codes with block length that is exponential in the number of multiplications allowed, and therefore we can only afford a constant number of multiplications.

[3]The inequality can be seen by defining $\alpha \stackrel{\text{def}}{=} 1-\delta$, noting that $\sqrt{\delta} = \sqrt{1-\alpha} \leq \sqrt{(1-\alpha/2)^2} = 1-\alpha/2$, and then observing that the latter yields $1-\sqrt{\delta} \geq 1-(1-\alpha/2) = (1-\delta)/2$.

The tensor product of multiplication codes. The following proposition shows that applying the tensor product operation preserves the multiplication property of codes.

Proposition 3.13. *Let C_1, C_2, and C_3 be codes of the same block length such that for every two codewords $\tilde{c}_1 \in C_1$ and $\tilde{c}_2 \in C_2$ it holds that $\tilde{c}_1 \cdot \tilde{c}_2 \in C_3$. Then, for every $m \in \mathbb{N}$, and for every $c_1 \in C_1^m$, $c_2 \in C_2^m$, it holds that $c_1 \cdot c_2 \in C_3^m$.*

Proof Let $\ell \in \mathbb{N}$ be the block length of C_1, C_2, and C_3, and fix $m \in \mathbb{N}$. Let $c_1 \in C_1^m$, $c_2 \in C_2^m$ be codewords. We view c_1, c_2 and $c_1 \cdot c_2$ as functions from $[\ell]^m$ to $\mathbb{F}$. By Fact 3.6, for every $1 \leq t \leq m$ and $i_1, \ldots i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that $c_1(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ and $c_2(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ are codewords of C_1 and C_2 respectively. The multiplication property of C_1, C_2, and C_3 now implies that for every $1 \leq t \leq m$ and $i_1, \ldots i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that

$$(c_1 \cdot c_2)(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$$

is a codeword of C_3. By applying Fact 3.6 in the opposite direction, the latter claim implies that $c_1 \cdot c_2$ is a codeword of C_3^m, as required. ■

4 The Sum-Check Protocol Revisited

In this section, we show a generalization of the sum-check protocol, which views the sum-check protocol as a protocol for reducing the evaluation of one tensor codeword to the evaluation of another tensor codeword. We will use this generalization both in the proof of co**NP** $\subseteq$ **IP** and of **IP** = **PSPACE**. In order to explain this idea and explain why it is useful, we need the following definition of "consistency", which generalizes the notion of the encoding of a message.

Definition 4.1. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, let $k', m \in \mathbb{N}$ be such that $k' \leq k$. We say that a codeword $c : [\ell]^m \to \mathbb{F}$ of C^m is consistent with a function $f : [k']^m \to \mathbb{F}$ if c agrees with f on $[k']^m$.

Let ϕ be a (possibly quantified) Boolean formula over m variables (if ϕ is quantified, then m is the number of free variables). We say that c is consistent with ϕ if c is consistent with the truth table of ϕ, viewed as a function from $\{0,1\}^m$ to $\mathbb{F}$.

Remark 4.2. Observe that every codeword is consistent with the message it encodes. In particular, if $k' = k$ in the above definition, then c is consistent with f if and only if c is the encoding of f. On the other hand, if $k' < k$, then there may be many codewords of C^m that are consistent with f.

As an example for the notion of consistency, note that the arithmetization of a Boolean formula ϕ yields a multivariate polynomial that is consistent with ϕ. Observe, however, that the latter polynomial is not *the encoding of the truth table of* ϕ via a Reed-Muller code; that is, this polynomial is *not the low-degree extension of the truth table of* ϕ. Thus, the arithmetization also provides an example for the difference between the *encoding of* a truth table and a codeword that is *consistent with* the truth table.

Now, our generalization of the sum-check protocol says roughly the following: Let c be a codeword of a code C^m, and let d be a codeword of a code D^m that is consistent with the message encoded by c. Then, the sum-check protocol reduces the task of verifying a claim of the form $c(\bar{i}) = u$ to the task of verifying a claim of the form $d(\bar{r}) = v$ (where $\bar{i}$ and $\bar{r}$ are coordinates of c and d respectively, and $u, v \in \mathbb{F}$).

Such a reduction is useful, for example, when the verifier can compute $d(\bar{r})$ easily, but can not compute $c(\bar{i})$ efficiently without the help of the prover. As a concrete example, consider the case

where c is the low-degree extension of the truth table of a formula ϕ and d is the polynomial obtained from the arithmetization of ϕ. Then, the sum-check protocol reduces the (hard) task of evaluating the low-degree extension (i.e., computing $c(\bar{i})$) to the (easy) task of evaluating the arithmetization polynomial (i.e. computing $d(\bar{r})$). A related example is the original proof of co**NP** $\subseteq$ **IP**, where the sum-check protocol is used to reduce the evaluation of an exponential sum (which is hard for the verifier) to the evaluation of the polynomial obtained from the arithmetization (which is easy for the verifier).

We first give an informal statement of the reduction, and then give the formal statement.

Theorem 4.3 (The sum-check protocol, informal)**.** *Let C and D be codes, and let $c \in C^m$ and $d \in D^m$ be codewords such that d is consistent with the message encoded by c. Then, there exists an interactive protocol that takes as input a claim of the form "$c\left(\bar{i}\right) = u$" and behaves as follows:*

- ***Completeness:*** *If the claim is correct (i.e., $c\left(\bar{i}\right) = u$), then the protocol outputs a* correct *claim of the form "$d(\bar{r}) = v$".*
- ***Soundness****: If the claim is incorrect (i.e., $c\left(\bar{i}\right) \neq u$), then with high probability the protocol either rejects or outputs an* incorrect *claim of the form "$d(\bar{r}) = v$".*

Theorem 4.4 (The sum-check protocol, formal)**.** *There exists a public coin interactive protocol between an unbounded prover and a polynomial time verifier that behaves as follows:*

- ***Input:*** *The parties enter the protocol with a common input that contains the following:*
  - *A finite field $\mathbb{F}$.*
  - *The generating matrices of systematic codes $C : \mathbb{F}^{k_C} \to \mathbb{F}^{\ell_C}$ and $D : \mathbb{F}^{k_D} \to \mathbb{F}^{\ell_D}$ where $k_D \geq k_C$ and D has relative distance δ_D.*
  - *A pair $\left(\bar{i}, u\right)$, where $\bar{i} \in [\ell_C]^m$ and $u \in \mathbb{F}$.*
- ***Output:*** *At the end of the protocol, the verifier either rejects, or outputs a pair $(\bar{r}, v)$, where $\bar{r} \in [\ell_D]^m$ and $v \in \mathbb{F}$.*

The output satisfies the following condition. For every two codewords $c \in C^m$, $d \in D^m$ such that d is consistent with the message encoded by c, the following holds:

- ***Completeness:*** *If $c\left(\bar{i}\right) = u$, then there exists a strategy for the prover that makes the verifier output with probability 1 a pair $(\bar{r}, v)$ such that $d\left(\bar{r}\right) = v$.*
- ***Soundness:*** *If $c\left(\bar{i}\right) \neq u$, then for every strategy taken by the prover, the probability that the verifier outputs a pair $(\bar{r}, v)$ for which $d\left(\bar{r}\right) = v$ is at most $m \cdot (1 - \delta_D)$.*

Furthermore, the output $\bar{r}$ depends only on the randomness used by the verifier.

Remark 4.5. The statement of Theorem 4.4 may seem confusing, since the codewords c and d are not given to the prover and verifier in any way. In fact, the codewords c and d are chosen by the prover, and may be chosen arbitrarily, subject to d being consistent with the message encoded by c.

However, in this work we will use Theorem 4.4 as a sub-protocol of higher level protocols. In those applications, the prover will be forced to use specific choices of c and d in order to convince the verifier of the high level protocol. In particular, those specific choices of c and d will be determined by the high level protocol.

Proof Let $\mathbb{F}$, C, D, m be as in the theorem. For convenience, throughout the description of the protocol we fix specific choices of the codewords c and d as in the theorem. However, the strategy of the verifier described below does not depend on the specific choice of c and d. Note that the strategy of the prover must depend on the choice of c and d.

We begin with recalling that by Claim 3.7, there exist scalars $\alpha_{t,j} \in \mathbb{F}$ for $1 \leq t \leq m$ and $j \in [k]$ such that for every choice of c it holds that $c\left(\bar{i}\right) = u$ if and only if

$$\sum_{j_1 \in [k_C]} \alpha_{1,j_1} \cdot \sum_{j_2 \in [k_C]} \alpha_{2,j_2} \cdot \ldots \sum_{j_m \in [k_C]} \alpha_{m,j_m} \cdot c(j_1, \ldots, j_m) = u$$

Moreover, the coefficients $\alpha_{t,j}$ can be computed efficiently. We know that c is systematic, and that d is consistent with the message encoded by c, and therefore c and d agree on $[k_C]^m$. Hence, in order to verify that $c\left(\bar{i}\right) = u$, it suffices to verify that

$$\sum_{j_1 \in [k_C]} \alpha_{1,j_1} \cdot \sum_{j_2 \in [k_C]} \alpha_{2,j_2} \cdot \ldots \sum_{j_m \in [k_C]} \alpha_{m,j_m} \cdot d(j_1, \ldots, j_m) = u$$

From this point on, the prover and verifier compute the above exponential sum exactly as in the standard sum-check protocol, except that univariate polynomials are replaced by codewords of D. Details follow.

The verifier and prover engage in an iterative protocol of m iterations. Let $v_0 \stackrel{\text{def}}{=} u$. When the parties enter the t-th iteration, the prover should convince the verifier that the following equality holds for some $r_1, \ldots, r_{t-1} \in [\ell]$ and v_{t-1} that are determined in the previous iterations.

$$\sum_{j_t \in [k_C]} \alpha_{t,i_t} \cdot \ldots \sum_{j_m \in [k_C]} \alpha_{m,i_m} \cdot d(r_1, \ldots, r_{t-1}, j_t, \ldots, j_m) = v_{t-1}$$

To this end, let us consider the function $h : [\ell] \to \mathbb{F}$ defined by

$$h(j_t) = \sum_{j_{t+1} \in [j_C]} \alpha_{t+1,j_{t+1}} \cdot \ldots \sum_{j_m \in [k_C]} \alpha_{m,j_m} \cdot d(r_1, \ldots, r_{t-1}, j_t, \ldots, j_m)$$

Observe that by Claim 3.8 the function h is a codeword of D. This follows by applying Claim 3.8 to the function $d(r_1, \ldots, r_{t-1}, \cdot, \cdot, \ldots, \cdot)$, while recalling that this function is a codeword of D^{m-t+1} by Fact 3.6.

The verifier expects an honest prover to send the function h (represented by its truth table). Let $h' : [\ell] \to \mathbb{F}$ be the function sent by the prover. The verifier checks that h' is a codeword of D, and that $\sum_{j_t \in [k_c]} a_{t,i_t} \cdot h'(j_t) = v_{t-1}$, and rejects if any of the checks fails. Next, the verifier chooses $r_t \in [\ell]$ uniformly at random and sends it to the prover. The parties now enter the $(t+1)$-th iteration of the protocol with $v_t \stackrel{\text{def}}{=} h'(r_t)$. Finally, at the end of the protocol, the verifier outputs the pair $(\overline{r}, v)$ where $\overline{r} \stackrel{\text{def}}{=} (r_1, \ldots, r_m)$ and $v \stackrel{\text{def}}{=} v_m$.

The completeness of the protocol is clear, and analysis of the soundness works exactly as the standard sum-check protocol. In particular, if the parties enter an iteration with a false claim, then one of the following two cases must hold:

- the verifier rejects, since h' does not pass the checks, or,
- h' is a codeword of D but is not equal to h, in which case it holds that $h'(r_t) \neq h(r_t)$ with probability at least δ_D.

Thus, the probability that the parties enter the next iteration with a true claim is at most $1 - \delta_D$.

The "furthermore" part of the theorem, which says that $\overline{r}$ depends only on the randomness used by the verifier, follows immediately from the description of the protocol. ■

5 A Proof of coNP $\subseteq$ IP

In this section we prove that co**NP** $\subseteq$ **IP** using tensor codes. We begin with an overview of the proof, and then provide the full details.

5.1 Proof overview

In order to prove that co**NP** $\subseteq$ **IP**, it suffices to design a protocol for verifying that a Boolean formula is unsatisfiable. For every Boolean formula ϕ, let us denote by h_ϕ the encoding of the truth table of ϕ via some tensor code of relative distance at least $\frac{1}{2}$. Observe that if ϕ is unsatisfiable then h_ϕ is the all-zeroes codeword, since the encoding of the all-zeroes message via a linear code is always the all-zeroes codeword. On the other hand, if ϕ is satisfiable then h_ϕ is non-zero on at least $\frac{1}{2}$ fraction of its coordinates.

A toy problem. We begin by making the unjustified assumption that for every formula ϕ and coordinate $\bar{i}$ of h_ϕ we can compute $h_\phi(\bar{i})$ efficiently. Under this assumption, it is easy to show that co**NP** $\subseteq$ **RP**. This is true since we can use the following randomized algorithm for checking the unsatisfiability of a formula: When given as input a formula ϕ, the algorithm chooses a coordinate $\bar{i}$ uniformly at random, and accepts if and only if $h_\phi(\bar{i}) = 0$.

Of course, the above assumption seems unjustified. The point is that, while we may not be able to compute $h_\phi(\bar{i})$ efficiently, we can devise an interactive protocol that allows an efficient verifier to verify the value of $h_\phi(\bar{i})$. By using this protocol inside the aforementioned "algorithm", we obtain a protocol for verifying the unsatisfiability of a Boolean formula. It remains to show how to construct a protocol for verifying the value of $h_\phi(\bar{i})$.

Proof via Arithmetization. We now show a protocol for verifying the value of $h_\phi(\bar{i})$ that uses arithmetization, and we will later show how to avoid the use of arithmetization. Let ϕ a Boolean formula over n variables, and let p_ϕ the polynomial that is obtained from the arithmetization of ϕ. We observe that p_ϕ is *consistent* with the formula ϕ, and it can be shown that p_ϕ is a codeword of some tensor code[4]. Therefore, we can use the sum-check protocol of Theorem 4.4 to reduce the task of verifying a claim of the form $h_\phi(\bar{i}) = u$ to the task of verifying a claim of the form $p_\phi(\bar{r}) = v$. Finally, observe that the verifier can compute the value $p_\phi(\bar{r})$ by itself, and thus verify that $p_\phi(\bar{r}) = v$. This concludes the description of the protocol.

Proof via Error Correcting Codes. In order to remove the use of arithmetization in the foregoing protocol, we examine the properties of the polynomial p_ϕ on which we relied, and construct a codeword $c_{M,\phi}$ that has the same properties without using polynomials. Specifically, the codeword $c_{M,\phi}$ will possess the following properties:

1. $c_{M,\phi}$ is a codeword of some tensor code.
2. $c_{M,\phi}$ is consistent with ϕ.
3. For every coordinate $\bar{j}$, the value $c_{M,\phi}(\bar{j})$ can be computed in polynomial time.

[4]To see it, assume that p_ϕ is an n-variate polynomial whose individual degrees are bounded by some number d. It turns out that the family of such polynomials is a tensor code. Specifically, if we let RS denote the Reed-Solomon code of univariate polynomials of degree at most d, then it is well-known that the aforementioned family of polynomials is exactly RS^n.

It can be observed that those properties are the only properties of p that we needed. This yields the following protocol: In order to verify a claim of the form $h_\phi(\bar{i}) = u$, the verifier reduces it to a claim of the form $c_{M,\phi}(\overline{r}) = v$ using the sum-check protocol of Theorem 4.4. Then, the verifier computes $c_{M,\phi}(\overline{r})$ by itself, and accepts if and only if $c_{M,\phi}(\overline{r}) = v$.

Specialized formulas and the construction of $c_{M,\phi}$. In general, we do not know how to construct the above codeword $c_{M,\phi}$ for every formula ϕ, but only for "specialized formulas", which will be discussed shortly. To resolve this issue, the protocol begins by transforming ϕ into an equivalent specialized formula ϕ_{sp}, and then proceeds as before while working with ϕ_{sp}. This issue is a direct consequence of the limitation of our multiplication codes that allows only one multiplication, as discussed after Proposition 3.10.

A "specialized formula" is a formula ϕ that can be written as $\phi = \phi_{\text{once}} \wedge \phi_{\text{eq}}$, where ϕ_{once} is a read-once formula, and ϕ_{eq} is a conjunction of equality constraints. The point is that since ϕ_{once} and ϕ_{eq} are very simple, it is easy to evaluate the encoding of their truth tables via any (tensor) code. Now, we let $c_{A,\phi_{\text{once}}}$ and $c_{B,\phi_{\text{eq}}}$be the encodings of ϕ_{once} and ϕ_{eq} via the multiplication codes of Proposition 3.10, and set $c_{M,\phi} = c_{A,\phi_{\text{once}}} \cdot c_{B,\phi_{\text{eq}}}$. It is easy to see that the codeword $c_{M,\phi}$ is consistent with ϕ. Moreover, the codeword $c_{M,\phi}$ is easy to evaluate, since $c_{A,\phi_{\text{once}}}$ and $c_{B,\phi_{\text{eq}}}$ are easy to evaluate.

We stress that while $c_{A,\phi_{\text{once}}}$ and $c_{B,\phi_{\text{eq}}}$ are the *unique encodings* of the truth tables of ϕ_{once} and ϕ_{eq} (via the corresponding codes), the codeword $c_{M,\phi}$ is not the encoding of the truth table of ϕ (via the corresponding code), but merely a codeword that is consistent with ϕ. This is a side-effect of the multiplication operation.

Comparison with the arithmetization technique. We view the construction of the codeword $c_{M,\phi}$ as a generalization of the arithmetization technique, since it produces a codeword that has essentially the same properties of the polynomial p_ϕ, but does it using any tensor code and not necessarily a polynomial code. However, one should note that, while the codeword $c_{M,\phi}$ can be used to replace p_ϕ in the above argument, it may not do so in *every* argument that involves arithmetization (e.g. some of the proofs of the PCP theorem). That is, our technique should be thought as a generalization of the arithmetization technique only in the context of the **IP** theorem.

Moreover, our construction of the codeword $c_{M,\phi}$ can only be applied to specialized formulas, while the arithmetization technique can be applied to any formula.

5.2 Full proof

We turn to describe the full details of the proof. We begin by defining the notion of "specialized formula" mentioned above.

Definition 5.1. A specialized formula is a formula ϕ that can be written as $\phi = \phi_{\text{once}} \wedge \phi_{\text{eq}}$, where

1. ϕ_{once} is a read-once formula, i.e., every variable occurs in ϕ_{once} exactly once, and
2. ϕ_{eq} is a conjunction of equality constraints over the variables of ϕ.

We can now state the "arithmetization generalization" discussed above, that is, the construction of the codeword $c_{M,\phi}$.

Lemma 5.2. *Let $\mathbb{F}$ be a finite field, and let $C_M : \mathbb{F}^4 \to \mathbb{F}^\ell$ be the multiplication code generated by Proposition 3.10 for $k = 2$ and any relative distance δ. Then, there exists a polynomial time algorithm that behaves as follows:*

- ***Input:*** *The algorithm is given as input a* specialized *Boolean formula ϕ over n variables, the generator matrix of C_M, and a coordinate $\bar{i} \in [\ell]^n$.*
- ***Output:*** *The algorithm outputs $c_{M,\phi}(\bar{i})$, where $c_{M,\phi}$ is a fixed codeword of $(C_M)^n$ that is consistent with ϕ and is determined by ϕ.*

We prove Lemma 5.2 in Section 5.2.1, but first, we show how to prove that co**NP** $\subseteq$ **IP** based on Lemma 5.2. To this end, we use the following standard fact that says that every formula can be transformed into an "equivalent" specialized formula.

Fact 5.3 (). *Let ϕ be a Boolean formula over n variables, and let m be the total number of occurrences of variables in ϕ. Then, there exists a specialized formula ϕ_{sp} over m variables that is satisfiable if and only if ϕ is satisfiable. Furthermore, ϕ_{sp} can be computed in polynomial time from ϕ. We refer to ϕ_{sp} as the the specialized version of ϕ.*

Proof ϕ_{sp} is obtained from ϕ by applying the standard transformation for making each variable appear at most three times. That is, ϕ_{sp} is constructed by

1. Replacing each occurrence of a variable in ϕ with a new variable, which may be thought as a "copy" of the original variable.
2. Adding equality constraints for each pair of variables that are copies of the same variable in ϕ.

It is easy to see that ϕ_{sp} satisfies the requirements. ■

Theorem 5.4. co**NP** $\subseteq$ **IP**

Proof We design a protocol for verifying the unsatisfiability of a Boolean formula. Let ϕ be a Boolean formula over n variables and m occurrences, and let ϕ_{sp} be its specialized version constructed by Fact 5.3. It suffices to design a protocol for verifying the satisfiability of ϕ_{sp}.

Let $\mathbb{F}$ be a finite field of size at least $4m$, let C_M be the code generated by Proposition 3.10 for $k = 2$ and relative distance $\delta = 1 - 1/2m$, and let $c_M = c_{M,\phi_{\text{sp}}}$ be the codeword whose existence is guaranteed by Lemma 5.2. Let $H : \mathbb{F}^2 \to \mathbb{F}^{\ell_H}$ be any systematic linear code of distance at least $1 - 1/2m$ (for example, one may use the $|\mathbb{F}|$-ary Hadamard code), and let $h = h_{\phi_{\text{sp}}}$ be the encoding of the truth table of ϕ_{sp} via H^m. Note that h is the *(unique) encoding* of the truth table of ϕ_{sp} via H^m, but may be hard to evaluate, while c_M is merely a codeword of $(C_M)^m$ that is *consistent* with ϕ_{sp}, but is easy to evaluate.

Observe that if ϕ_{sp} is unsatisfiable then h is the all-zeroes function, while if ϕ_{sp} is satisfiable then at least $\left(1 - \frac{1}{2m}\right)^m \geq \frac{1}{2}$ fraction of the entries of h are non-zero. Thus, it suffices to check that a random coordinate of h is non-zero.

At the beginning of the protocol, the verifier chooses a uniformly distributed tuple $\bar{i} \in [\ell_H]^m$, and sends it to the prover. The prover should prove to the verifier that $h(\bar{i}) = 0$. To this end, the prover and the verifier engage in the sum-check protocol of Theorem 4.4 with $C = H$, $D = C_M$, $c = h$, $d = c_M$, $\bar{i} = \bar{i}$, and $u = 0$. If the verifier does not reject at this stage, then the sum-check protocol outputs a pair $(\bar{r}, v)$ that is expected to satisfy $c_M(\bar{r}) = v$. Finally, the verifier uses the algorithm of Lemma 5.2 to compute $c_M(\bar{r})$, accepts if $c_M(\bar{r}) = v$, and rejects otherwise.

For the completeness of the protocol, note that if ϕ_{sp} is unsatisfiable, then $h(\bar{i}) = 0$. Therefore, by the completeness of the sum-check protocol of of Theorem 4.4, there exists a prover strategy that guarantees that the verifier does not reject and outputs a pair $(\bar{r}, v)$ such that $c_M(\bar{r}) = v$. It is easy to see that if the prover uses this strategy, the verifier will always accept.

For the soundness of the protocol, observe that if ϕ_{sp} is satisfiable, then $h(\bar{i}) \neq 0$ with probability at least $\frac{1}{2}$. Conditioned on $h(\bar{i}) \neq 0$, the soundness of Theorem 4.4 guarantees that with probability at least $m \cdot (1 - \delta_{C_M}) \geq \frac{1}{2}$, the verifier either rejects or outputs a pair $(\bar{r}, v)$ such that $c_M(\bar{r}) \neq v$, in which case the verifier rejects in the next step. It follows that if ϕ is satisfiable, then the verifier rejects with probability at least $\frac{1}{4}$, which suffices for our purposes. ■

5.2.1 Proof of Lemma 5.2

We turn to proving Lemma 5.2. Let ϕ be a specialized Boolean formula over n variables, and let (C_A, C_B, C_M) be the multiplication code generated by Proposition 3.10 for $k = 2$ and any relative distance δ. We seek to construct a codeword $c_M = c_{M,\phi}$ of $(C_M)^n$ that is consistent with ϕ, and such that the value of c_M at any coordinate can be computed in polynomial time.

Recall that ϕ can be written as $\phi = \phi_{\text{once}} \wedge \phi_{\text{eq}}$, where ϕ_{once} is a read-once formula and ϕ_{eq} is a conjunction of equalities. Furthermore observe that the formulas ϕ_{once} and ϕ_{eq} can be computed from ϕ in polynomial time, by simply letting ϕ_{eq} be the conjunction of all the equality constraints in ϕ.

We now show how to construct the codeword c_M that is consistent with ϕ. Let c_A be the encoding of the truth table of ϕ_{once} via $(C_A)^n$, and let c_B be the encoding of the truth table of ϕ_{eq} via $(C_B)^n$. We choose $c_M \stackrel{\text{def}}{=} c_A \cdot c_B$. Observe that $c_A \cdot c_B$ is indeed consistent with ϕ, and that it is a codeword of $(C_M)^n$ by Proposition 3.13.

It remains to show that for every coordinate $\bar{i}$ of c_M, the value $c_M(\bar{i})$ can be computed efficiently. Let ℓ denote the block length of C_A, C_B, and C_M. Propositions 5.6 and 5.8 below imply that for every $\bar{i} \in [\ell]^n$, the values $c_A(\bar{i})$ and $c_B(\bar{i})$ can be computed efficiently. It follows that for every $\bar{i} \in [\ell]^n$, we can compute the value $c_M(\bar{i})$ efficiently by first computing $c_A(\bar{i})$ and $c_B(\bar{i})$ and then setting $c_M(\bar{i}) = c_A(\bar{i}) \cdot c_B(\bar{i})$. This concludes the proof of Lemma 5.2 up to the proofs of Propositions 5.6 and 5.8.

We stress that while c_A and c_B are *the unique encodings* of ϕ_{once} and ϕ_{eq} via $(C_A)^n$ and $(C_B)^n$ respectively, c_M is merely a codeword of $(C_M)^n$ that is *consistent* with ϕ, and not the encoding of ϕ via $(C_M)^n$. The reason is that, if we consider two messages $x, y \in \mathbb{F}^2$, then $C_A(x) \cdot C_B(y)$ is a codeword of C_M that is *consistent* with $x \cdot y$, but is not the *encoding* of $x \cdot y$ via C_M; in particular, note that the message length of C_M is greater than the length of $x \cdot y$.

Notation 5.5. In the statements of the following propositions, we denote by $C : \mathbb{F}^2 \to \mathbb{F}^{\ell_C}$ a fixed arbitrary code, and for every Boolean formula φ over n variables, we denote by c_φ the encoding of the truth table of φ via C^n.

Proposition 5.6 (Codeword for read-once formulas)**.** *There exists a polynomial time algorithm such that when the algorithm is given as input a read-once Boolean formula φ, the generator matrix of a code C, and a coordinate $\bar{i}$ of c_φ, the algorithm outputs $c_\varphi(\bar{i})$.*

Proof We show a recursive construction of a codeword c_φ, and use it later to derive a recursive algorithm for computing the coordinates of c_φ. We have the following recursive construction:

1. If $\varphi = x_t$ for some Boolean variable x_t, then c_φ is the encoding of the vector $(0, 1)$ via C (recall that the message length of C is 2).

2. Suppose that $\varphi = \neg\varphi'$ for some Boolean formula φ' over n variables. Let $\overline{1}_n$ be the all-ones function that maps all the elements of $\{0,1\}^n$ to 1, and let $c_{\overline{1}_n}$ be the encoding of $\overline{1}_n$ via C^n. Then, it holds that $c_\varphi = c_{\overline{1}_n} - c_{\varphi'}$.

3. Suppose that $\varphi = \varphi_1 \wedge \varphi_2$ for some Boolean formulas φ_1 and φ_2 over n_1 and n_2 variables respectively. Observe that φ_1 and φ_2 must be over disjoint sets of variables, since by assumption every variable occurs in φ exactly once. Let us relabel the variables of φ such that the first n_1 variables are the variables of φ_1 and the last n_2 variables are the variables of φ_2. We now obtain that $c_\varphi = c_{\varphi_1} \otimes c_{\varphi_2}$.

4. If $\varphi = \varphi_1 \vee \varphi_2$, then c_φ can be constructed from c_{φ_1} and c_{φ_2} using the de Morgan laws and the previous cases.

The above recursive construction immediately yields the following recursive algorithm for computing $c_{A,\varphi}(\bar{i})$ where $\bar{i} = (i_1, \ldots, , i_n) \in [\ell_C]^n$:

1. If $\varphi = x_t$, then the algorithm computes c_φ directly by encoding the vector $(0,1)$ with C, and outputs $c_\varphi(\bar{i})$.

2. Suppose that $\varphi = \neg\varphi'$. In this case, the algorithm computes $c_{\bar{1}_n}(\bar{i})$ and $c_{\varphi'}(\bar{i})$ and outputs $c_{\bar{1}_q}(\bar{i}) - c_{\varphi'}(\bar{i})$. The value $c_{\varphi'}(\bar{i})$ is computed recursively. In order to compute $c_{\bar{1}_n}(\bar{i})$, observe that $c_{\bar{1}_n} = \underbrace{c_{\bar{1}_1} \otimes \ldots \otimes c_{\bar{1}_1}}_{n}$. It therefore follows that

$$c_{\bar{1}_n}(\bar{i}) = c_{\bar{1}_1}(i_1) \cdot \ldots \cdot c_{\bar{1}_1}(i_n)$$

   Thus, in order to compute $c_{\bar{1}_n}(\bar{i})$, the algorithm computes $c_{\bar{1}_1}$ by encoding the vector $(1,1)$ with C, and outputs $c_{\bar{1}_1}(i_1) \cdot \ldots \cdot c_{\bar{1}_1}(i_n)$.

3. Suppose that $\varphi = \varphi_1 \wedge \varphi_2$. Again, we assume that the first n_1 variables of φ are the variables of φ_1, and that the last n_2 variables of φ are the variables of φ_2. Also, observe that $n = n_1 + n_2$. Then, it holds that

$$c_\varphi(\bar{i}) = c_{\varphi_1}(i_1, \ldots, i_{n_1}) \cdot c_{\varphi_2}(i_{n_1+1}, \ldots, i_n)$$

   The algorithm thus computes $c_{\varphi_1}(i_1, \ldots, i_{n_1})$ and $c_{\varphi_2}(i_{n_1+1}, \ldots, i_n)$ recursively and outputs their product.

Clearly, the above algorithm is efficient, and computes $c_\varphi(\bar{i})$ correctly. ■

Remark 5.7. Note that the assumption that every variable occurs exactly once in φ is critical for the proof of Proposition 5.6. Specifically, this assumption is used in handling the case of $\varphi = \varphi_1 \wedge \varphi_2$, and allows us to simulate the effect of multiplication using the tensor product operation (i.e., by setting $c_\varphi = c_{\varphi_1} \otimes c_{\varphi_2}$). Without the assumption, it could be the case that φ_1 and φ_2 have common variables, which would imply that $c_\varphi \neq c_{\varphi_1} \otimes c_{\varphi_2}$.

Proposition 5.8 (Codeword for equality constraints)**.** *There exists a polynomial time algorithm such that when the algorithm is given as input a Boolean formula φ which is a conjunction of equality constraints, the generator matrix of C, and a coordinate $\bar{i}$ of c_φ, the algorithm outputs $c_\varphi(\bar{i})$.*

Proof We first deal with the special case in which φ is satisfied if and only if *all* its variables are equal to each other. Let $\bar{i} \in [\ell_C]^n$ be a coordinate. We wish to compute $c_\varphi(\bar{i})$ efficiently. By Claim 3.7, there exist scalars $\alpha_{t,j} \in \mathbb{F}$ (for every $1 \leq t \leq n$ and $j \in \{0,1\}$) such that

$$c(\bar{i}) = \sum_{j_1 \in \{0,1\}} \alpha_{1,j_1} \cdot \sum_{j_2 \in \{0,1\}} \alpha_{2,j_2} \cdot \ldots \sum_{j_n \in \{0,1\}} \alpha_{n,j_n} \cdot c_\varphi(j_1, \ldots, j_n)$$

By our assumption on φ, each term $c_\varphi(j_1, \ldots, j_n)$ in the above exponential sum is 1 if $j_1 = \ldots = j_n$ and 0 otherwise. It thus follows that

$$c_\varphi(\bar{i}) = \prod_{t=1}^{n} \alpha_{t,0} + \prod_{t=1}^{n} \alpha_{t,1}$$

Now, the above sum is easy to compute, since by Claim 3.7 the coefficients $\alpha_{t,j}$ can be computed efficiently.

We turn to consider the general case, in which φ may be any conjunction of equality constraints over its variables. In this case, one can partition the variables of φ to sets $S_1, \ldots, S_t$ such that two variables are in the same set if and only if they are equal in every satisfying assignment of φ. For each such S_j, let φ_j be the formula over the variables in S_j that is satisfied if and only if all the variables in S_j are equal. Observe that

$$\varphi = \varphi_1 \wedge \ldots \wedge \varphi_t$$

Let us relabel the variables of φ such that the first $|S_1|$ variables are the variables of S_1, the next $|S_2|$ variables are the variables of S_2, etc. After the relabeling, it holds that

$$c_\varphi = c_{\varphi_1} \otimes \ldots \otimes c_{\varphi_t}$$

Therefore, if we let $\bar{i}$ be any coordinate of c_φ and denote $\bar{i}_{|S_j}$ the restriction of $\bar{i}$ to S_j, it holds that

$$c_\varphi(\bar{i}) = c_{\varphi_1}(\bar{i}_{|S_1}) \cdot \ldots \cdot c_{\varphi_t}(\bar{i}_{|S_t})$$

Now, each of the formulas φ_j matches the special case we already dealt with, and therefore we can efficiently compute the value $c_{\varphi_j}(\bar{i}_{|S_j})$. We can thus compute $c_{\varphi_1}(\bar{i}_{|S_1})$ efficiently as well, as required. ■

Remark regarding algebrization. Recall that the arithmetization technique is the only non-relativizing ingredient of the proof of the **IP** theorem. Indeed, a main motivation of the algebrization framework of [AW08] was to try to capture the arithmetization technique. While our arithmetization generalization (Lemma 5.2) does not seem to fit into the algebrization framework, one can prove the following "algebrization-like" variant of this lemma: Let $\mathcal{O} = \{\mathcal{O}_n : \{0,1\}^n \to \{0,1\}\}_n$ be an infinite sequence of Boolean oracles, and let us denote $C_{A,\mathcal{O}} = \{C_{A,\mathcal{O}_n}\}_n$ where $C_{A,\mathcal{O}_n}$ is the encoding of the truth table $\mathcal{O}_n$ by $(C_A)^n$. Then, there exists an algorithm that given oracle access to $C_{A,\mathcal{O}}$ satisfies the following requirement: when the algorithm is given as input a specialized formula φ that contains oracle predicates from the sequence $\mathcal{O}$ and a coordinate $\bar{i}$ of $c_{M,\varphi}$, the algorithm outputs $c_{M,\varphi}(\bar{i})$. Here $c_{M,\varphi}$ is a codeword of $(C_M)^n$ that is consistent with φ, as before.

6 The Proof of IP = PSPACE

In this section, we finally prove the **IP** theorem, that is,

Theorem 6.1. IP = PSPACE.

Since **TQBF** is a **PSPACE**-complete problem, it suffices to devise an interactive protocol for verifying the validity of a quantified Boolean formula. Recall that a quantified Boolean formula is a logical expression of the form

$$\underset{y_1 \in \{0,1\}}{\mathcal{Q}_1} \underset{y_2 \in \{0,1\}}{\mathcal{Q}_2} \cdots \underset{y_n \in \{0,1\}}{\mathcal{Q}_n} \phi(y_1, \ldots, y_n) \tag{1}$$

where ϕ is a Boolean formula and each $\mathcal{Q}_i$ denotes one of the quantifiers $\exists$ and $\forall$. A quantified Boolean formula is said to be valid if and only if the expression evaluates to 1 (i.e., evaluates to TRUE). We wish to design an interactive protocol which takes as an input a quantified Boolean formula, such that if the formula is valid the verifier accepts with probability 1, and otherwise accepts with probability at most $\frac{1}{2}$.

We begin with an overview of the proof in Section 6.1, and then give the full details in Section 6.2. We mention that our proof borrows ideas from the work of [GKR08].

6.1 Proof overview

The formulas ψ_i. Given a quantified formula as in (1), we define the following quantified formulas

$$\psi_t(y_1,\ldots,y_t) = \underset{y_{t+1}\in\{0,1\}}{\mathcal{Q}_{t+1}} \cdots \underset{y_n\in\{0,1\}}{\mathcal{Q}_n} \phi(y_1,\ldots,y_n)$$

That is, ψ_t is a formula in which $y_1,\ldots,y_t$ are free variables and $y_{t+1},\ldots,y_n$ are bounded variables. In particular, ψ_0 is the original quantified formula and ψ_n is the formula ϕ. We also consider the encodings of the truth table of ψ_t via $(C_A)^t$ and $(C_B)^t$, and denote them by $c_{A,t}$ and $c_{B,t}$ (where (C_A, C_B, C_M) are the multiplication codes of Proposition 3.10).

We mention that the actual proof will work with the specialized version ϕ_{sp} instead of ϕ itself (see Definition 5.1 and Fact 5.3). We ignore this technicality throughout this overview.

The structure of the protocol. Our interactive protocol begins by reducing the task of verifying the validity of Formula (1) to the task of verifying a claim of the form

$$c_{A,1}(\bar{i}_1) = v_{A,1} \quad \text{and} \quad c_{B,1}(\bar{i}_1) = v_{B,1} \tag{2}$$

where $\bar{i}_1$ is a coordinate of $c_{A,1}$ and $c_{B,1}$ - note that $\bar{i}_1$ is shared by both equalities in (2).

Next, the protocol proceeds to work in iterations: The prover and the verifier enter the t-th iteration with a claim of the form

$$c_{A,t}(\bar{i}_t) = v_{A,t} \quad \text{and} \quad c_{B,t}(\bar{i}_t) = v_{B,t} \tag{3}$$

Throughout the t-th iteration, the parties engage in a sub-protocol, in order to reduce the task of verifying the claim in (3) to the task of verifying a claim of the same form about $c_{A,t+1}$ and $c_{B,t+1}$.

Eventually, the parties end up with a claim about $c_{A,n}$ and $c_{B,n}$. This means that the prover is required to prove a claim about encodings of the truth table of $\psi_n = \phi$, which can be done in the same way as in the proof of co**NP** $\subseteq$ **IP**: The parties engage in the sum-check protocol in order to reduce the claim about $c_{A,n}$ and $c_{B,n}$ to a claim about a codeword of $(C_M)^n$ that is consistent with the truth table of ϕ, and then the verifier checks the latter claim by itself, by using the "arithmetization generalization" (Lemma 5.2).

A single iteration. We now describe how a single iteration of the protocol is performed. Let us focus on the t-th iteration, and assume that $\mathcal{Q}_{t+1} = \forall$ (the case where $\mathcal{Q}_{t+1} = \exists$ is similar). We consider the codeword $c_{M,t}$ of $(C_M)^t$ constructed by setting each coordinate $\bar{j}$ of $c_{M,t}$ as follows:

$$c_{M,t}(\bar{j}) = c_{A,t+1}(\bar{j},0) \cdot c_{B,t+1}(\bar{j},1) \tag{4}$$

Observe that $c_{M,t}$ is indeed a codeword of $(C_M)^t$ and that it is consistent with the truth table of ψ_t. Recall that our purpose is to reduce the verification of the claim in (3) to the verification of the

same claim for $t+1$. The codeword $c_{M,t}$ serves as a "bridge" between those two claims: On the one hand, $c_{M,t}$ is consistent with the message encoded by $c_{A,t}$ and $c_{B,t}$, whereas on the other hand $c_{M,t}$ is related to $c_{A,t+1}$ and $c_{B,t+1}$ by Equality (4). Our strategy is to first reduce the verification of the claim about $c_{A,t}$ and $c_{B,t}$ to the verification of a claim about $c_{M,t}$, and then reduce the latter to the verification of a claim about $c_{A,t+1}$ and $c_{B,t+1}$.

More specifically, the parties begin the iteration by reducing the task of verifying the claim in (3) to the task of verifying an equality of the form

$$c_{M,t}(\overline{r}) = v_{M,t} \tag{5}$$

Such a reduction can be done by invoking the sum-check protocol of Theorem 4.4 twice in parallel, once with $C = C_A$ and $D = C_M$, and once with $C = C_B$ and $D = C_M$, with the verifier using the same randomness for both invocations. The reason for using the same randomness for both invocations is that we want both invocations to output the same coordinate $\overline{r}$.

Next, the prover sends to the verifier two functions f_A and f_B, which are expected to be $c_{t+1,A}(\overline{r},\cdot)$ and $c_{t+1,B}(\overline{r},\cdot)$ respectively. The verifier checks that f_A and f_B are indeed codewords of C_A and C_B respectively, and that $f_A(0)\cdot f_B(1) = v_{M,t}$, where $v_{M,t}$ is the value from Equality (5). Finally, the verifier chooses a random coordinate s, and the parties enter the next iteration with the claim

$$c_{A,t+1}(\overline{r},s) = f_A(s) \quad \text{and} \quad c_{B,t+1}(\overline{r},s) = f_B(s)$$

6.2 The full proof

Fix a quantified formula

$$\underset{y_1\in\{0,1\}}{\mathcal{Q}_1} \cdots \underset{y_n\in\{0,1\}}{\mathcal{Q}_n} \phi(y_1,\ldots,y_n) \tag{6}$$

where ϕ is a Boolean formula over n variables and m occurrences of variables.

Moving to specialized formulas. Our first step is moving to work with specialized formulas, which will allow us to use the "arithmetization generalization" (Lemma 5.2). To this end, consider the specialized version ϕ_{sp} of ϕ, whose existence is guaranteed by Fact 5.3, and let us denote its variables by $x_1,\ldots,x_m$. Recall that each variable $x_{i'}$ of ϕ_{sp} is treated as a "copy" of some variable y_i. Let us relabel the variables of ϕ_{sp} such that for each $1 \le i \le n$, the variable x_i is a copy of y_i. Now, consider the formula

$$\underset{x_1\in\{0,1\}}{\mathcal{Q}_1} \cdots \underset{x_n\in\{0,1\}}{\mathcal{Q}_n} \underset{x_{n+1}\in\{0,1\}}{\exists} \cdots \underset{x_m\in\{0,1\}}{\exists} \phi_{\mathrm{sp}}(x_1,\ldots,x_m) \tag{7}$$

Observe that that Formula (6) is valid if and only if Formula (7) is valid. For the rest of the proof, we will work with the Formula (7). For convenience, we will denote Formula (7) as

$$\underset{x_1\in\{0,1\}}{\mathcal{Q}_1} \cdots \underset{x_m\in\{0,1\}}{\mathcal{Q}_m} \phi_{\mathrm{sp}}(x_1,\ldots,x_m) \tag{8}$$

even though we know that $\mathcal{Q}_{n+1} = \ldots = \mathcal{Q}_m = \exists$.

The formulas ψ_i and their encodings. As in the above overview, we define formulas ψ_t, but this time we define those formulas with respect to ϕ_{sp}. That is, for every $1 \le t \le m$, we define

$$\psi_t(y_1,\ldots,y_t) = \underset{x_{t+1}\in\{0,1\}}{\mathcal{Q}_{t+1}} \cdots \underset{x_m\in\{0,1\}}{\mathcal{Q}_m} \phi_{\mathrm{sp}}(x_1,\ldots,x_m)$$

Let (C_A, C_B, C_M) be the multiplication codes that result from invoking the algorithm of Proposition 3.10 with $k \stackrel{\text{def}}{=} 2$, $\delta \stackrel{\text{def}}{=} 1 - \frac{1}{2(m+1)^2}$, and with sufficiently large finite field $\mathbb{F}$, and let ℓ be their block length. For every $1 \leq t \leq m$, we define $c_{A,t}$ and $c_{B,t}$ to be the encodings of the truth table of ψ_t via $(C_A)^t$ and $(C_B)^t$ respectively.

A single iteration. The behavior of the parties in a single iteration is encapsulated in the following theorem, which we first state informally and give the formal statement.

Theorem 6.2 (Single iteration, informal)**.** *There exists an interactive protocol that takes as input a claim of the form "$c_{A,t}(\bar{i}_t) = v_{A,t}$ and $c_{B,t}(\bar{i}_t) = v_{B,t}$" and and behaves as follows:*

- ***Completeness:*** *If the claim is correct, then the protocol outputs a* correct *claim of the form "$c_{A,t+1}(\bar{i}_{t+1}) = v_{A,t+1}$ and $c_{B,t+1}(\bar{i}_{t+1}) = v_{B,t+1}$".*
- ***Soundness****: If the claim is incorrect (i.e., either $c_{A,t}(\bar{i}_t) \neq v_{A,t}$ or $c_{B,t}(\bar{i}_t) \neq v_{B,t}$), then with high probability the protocol either rejects or outputs an* incorrect *claim of the form "$c_{A,t+1}(\bar{i}_{t+1}) = v_{A,t+1}$ and $c_{B,t+1}(\bar{i}_{t+1}) = v_{B,t+1}$".*

We turn to state the formal version of the theorem, and to prove it.

Theorem 6.3 (Single iteration, formal)**.** *Let ϕ_{sp}, m, C_A, and C_B be defined as above, and for every $1 \leq t \leq m-1$ let ψ_t, $c_{A,t}$, and $c_{B,t}$ be defined as above with respect to ϕ_{sp}. There exists an interactive protocol between an unbounded prover and a polynomial time verifier that satisfies the following requirements:*

- ***Input:*** *The parties enter the protocol with a common input $\left(\bar{i}_t, v_{A,t}, v_{B,t}\right)$, where $\bar{i}_t \in [\ell]^t$ and $v_{A,t}, v_{B,t} \in \mathbb{F}$. Additional inputs are the numbers m, t, the generating matrices of C_A, C_B, C_M, and the quantified formula in (8).*
- ***Output:*** *At the end of the protocol, the verifier either rejects, or outputs a triplet $\left(\bar{i}_{t+1}, v_{A,t+1}, v_{B,t+1}\right)$, where $\bar{i}_{t+1} \in [\ell_D]^{t+1}$ and $v_{A,t+1}, v_{B,t+1} \in \mathbb{F}$.*

The output satisfies the following requirements:

- ***Completeness:*** *If both $c_{A,t}\left(\bar{i}_t\right) = v_{A,t}$ and $c_{B,t}\left(\bar{i}_t\right) = v_{B,t}$, then there exists a strategy for the prover that makes the verifier output with probability 1 a triplet $\left(\bar{i}_{t+1}, v_{A,t+1}, v_{B,t+1}\right)$ such that $c_{A,t+1}\left(\bar{i}_{t+1}\right) = v_{A,t+1}$ and $c_{B,t}\left(\bar{i}_{t+1}\right) = v_{B,t+1}$.*
- ***Soundness:*** *If either $c_{A,t}\left(\bar{i}_t\right) \neq v_{A,t}$ or $c_{B,t}\left(\bar{i}_t\right) \neq v_{B,t}$, then for every strategy taken by the prover, the probability that the verifier outputs a triplet $\left(\bar{i}_{t+1}, v_{A,t+1}, v_{B,t+1}\right)$ such that both $c_{A,t+1}\left(\bar{i}_{t+1}\right) = v_{A,t+1}$ and $c_{B,t}\left(\bar{i}_{t+1}\right) = v_{B,t+1}$ is at most $(t+1) \cdot (1-\delta) = \frac{t+1}{2(m+1)^2}$.*

Proof We begin by defining a codeword $c_{M,t}$ of $(C_M)^t$ as follows:

1. If $\mathcal{Q}_{t+1} = \forall$, then for every $\bar{j} \in [\ell]^t$ we define
$$c_{M,t}(\bar{j}) = c_{A,t+1}(\bar{j}, 0) \cdot c_{B,t+1}(\bar{j}, 1)$$
2. If $\mathcal{Q}_{t+1} = \exists$, then for every $\bar{j} \in [\ell]^t$ we define
$$\begin{aligned} c_{M,t}(\bar{j}) \quad = \quad & c_{A,t+1}(\bar{j}, 0) \cdot c_{B,t+1}(\bar{j}, 0) \\ & + c_{A,t+1}(\bar{j}, 1) \cdot c_{B,t+1}(\bar{j}, 1) \\ & - c_{A,t+1}(\bar{j}, 0) \cdot c_{B,t+1}(\bar{j}, 1) \end{aligned}$$

Observe that $c_{M,t}$ is consistent with the truth table of ψ_t, since if we restrict the above equalities to the Boolean hypercube $\{0,1\}^t$ then they become

$$c_{M,t}(\bar{j}) = \begin{cases} c_{A,t+1}(\bar{j},0) \wedge c_{B,t+1}(\bar{j},1) & (\text{If } \mathcal{Q}_{t+1} = \forall) \\ c_{A,t+1}(\bar{j},0) \vee c_{B,t+1}(\bar{j},1) & (\text{If } \mathcal{Q}_{t+1} = \exists) \end{cases}$$

Furthermore, observe that $c_{M,t}$ is indeed a codeword of $(C_M)^t$ for the following reasons: For each $b \in \{0,1\}$, it holds that $c_{A,t+1}(\cdot,b)$ and $c_{B,t+1}(\cdot,b)$ are codewords of $(C_A)^t$ and $(C_B)^t$ respectively. Furthermore, by Propositions 3.10 and 3.13 it holds that the multiplication of codewords of $(C_A)^t$ and $(C_B)^t$ yields a codeword of $(C_M)^t$. Finally, $(C_M)^t$ is a linear code, and therefore a sum of codewords of $(C_M)^t$ yields a codeword of $(C_M)^t$ (this is only relevant for the case that $\mathcal{Q}_{t+1} = \exists$).

The protocol starts with the parties invoking the sum-check protocol (Theorem 4.4) twice in parallel, using the same randomness for both invocations: The first invocation is done with $C = C_A$, $D = C_M$, $c = c_{A,t}$, $d = c_{M,t}$, $\bar{i} = \bar{i}_t$, and $u = v_{A,t}$, and the second invocation is done with $C = C_B$, $D = C_M$, $c = c_{B,t}$, $d = c_{M,t}$, $\bar{i} = \bar{i}_t$, $u = v_{B,t}$. The two invocations result in two pairs $(\overline{r}, v_{M,t})$, $\left(\overline{r}, v'_{M,t}\right)$, where $\overline{r} \in [\ell]^t$ and $v_{M,t}, v'_{M,t} \in \mathbb{F}$ - note that $\overline{r}$ is common to both pairs since the verifier uses the same randomness for both invocations (see the "Furthermore" part of Theorem 4.4). The verifier checks that $v_{M,t} = v'_{M,t}$, and rejects otherwise.

Next, the prover should send functions $f_A, f_B : [\ell] \to \mathbb{F}$. If the prover is honest, the functions f_A, f_B are supposed to satisfy $f_A(\cdot) = c_{A,t+1}(\overline{r}, \cdot)$ and $f_B(\cdot) = c_{B,t+1}(\overline{r}, \cdot)$. The verifier checks that f_A and f_B are codewords of C_A and C_B respectively, and rejects otherwise. In addition,

1. If $\mathcal{Q}_{t+1} = \forall$, the verifier checks that $f_A(0) \cdot f_B(1) = v_{M,t}$.
2. If $\mathcal{Q}_{t+1} = \exists$, the verifier checks that $f_A(0) \cdot f_B(0) + f_A(1) \cdot f_B(1) - f_A(0) \cdot f_B(1) = v_{M,t}$.

If the above check fails, the verifier rejects. Finally, the verifier chooses a uniformly distributed $j \in [\ell]$, and outputs the triplet $\left(\bar{i}_{t+1}, v_{A,t+1}, v_{B,t+1}\right)$, where $v_{A,t+1} = f_A(j)$, $v_{B,t+1} = f_B(j)$, and $\bar{i}_{t+1}$ is obtained by appending j to $\overline{r}$.

The completeness of the protocol is easy to verify. We turn to prove the soundness of the protocol. Without loss of generality, suppose that $c_{A,t}\left(\bar{i}_t\right) \neq v_{A,t}$ and that $\mathcal{Q}_i = \forall$- the cases where $c_{B,t}\left(\bar{i}_t\right) \neq v_{B,t}$ and $\mathcal{Q}_i = \exists$ can be handled similarly. By the soundness part of Theorem 4.4, with probability at least $1 - t \cdot (1-\delta)$ it holds that either the verifier rejects or $c_M(\overline{r}) \neq v_{M,t}$. Now, if the verifier does not reject, then it must hold that $f_A(0) \cdot f_B(1) = v_{M,t}$, and therefore $f_A(0) \cdot f_B(1) \neq c_M(\overline{r})$. By the definition of c_M, this implies that either $f_A(0) \neq c_{A,t+1}(\overline{r}, 0)$ or that $f_B(1) \neq c_{B,t+1}(\overline{r}, 1)$ - without loss of generality, assume the first. In this case, f_A is a codeword of C_A that differs from the codeword $c_{A,t+1}(\overline{r}, \cdot)$. Thus, with probability at least δ, it holds that $f_A(j) \neq c_{A,t+1}(\overline{r}, j)$, or in other words, that $c_{A,t+1}(\bar{i}_{t+1}) \neq v_{A,t+1}$. By the union bound, it follows that with probability at least $1 - (t+1) \cdot (1-\delta)$, the verifier either rejects or $c_{A,t+1}(\bar{i}_{t+1}) \neq v_{A,t+1}$, as required. ■

The full protocol. We finally turn to describe the full protocol for verifying the validity of the Quantified Formula (8). At the beginning of the protocol, the prover sends two functions $g_A, g_B : [\ell] \to \mathbb{F}$, that are supposed to be $c_{A,1}$ and $c_{B,1}$ respectively if the prover is honest. The verifier checks that

1. $g_A(0) \cdot g_B(1) = 1$ (if $\mathcal{Q}_1 = \forall$), or that
2. $g_A(0) \cdot g_B(0) + g_A(1) \cdot g_B(1) - g_A(0) \cdot g_B(1) = 1$ (if $\mathcal{Q}_1 = \exists$),

and rejects otherwise. Then, the verifier chooses $i_1 \in [\ell]$ uniformly at random and sets $v_{A,1} = g_A(i_1)$ and $v_{B,1} = g_B(i_1)$.

The parties then proceed in iterations for $1 \leq t \leq m-1$, each iteration invoking the protocol of Theorem 6.3. The parties finish the last iteration with a triplet $\left(\bar{i}_m, v_{A,m}, v_{B,m}\right)$, such that if the prover is honest it holds that $c_{A,m}(\bar{i}_m) = v_{A,m}$ and $c_{B,m}(\bar{i}_m) = v_{B,m}$. Observe that $c_{A,m}$ and $c_{B,m}$ are the encodings of the truth table of ϕ_{sp} via $(C_A)^m$ and $(C_B)^m$ respectively.

By Lemma 5.2, there exists a codeword $c_{M,\phi_{\mathrm{sp}}}$ of $(C_M)^m$ that is consistent with ϕ_{sp}, such that for every $\bar{j} \in [\ell]^m$, the value $c_{M,\phi_{\mathrm{sp}}}(\bar{j})$ can be computed efficiently (note that $c_{M,\phi_{\mathrm{sp}}}$ is *not* the same as the codeword $c_{M,t}$ in the proof of Theorem 6.3). The parties now engage in the sum-check protocol (Theorem 4.4) twice: The first invocation is with $C = C_A$, $D = C_M$, $c = c_{A,m}$, $d = c_{M,\phi_{\mathrm{sp}}}$, $\bar{i} = \bar{i}_m$, and $u = v_{A,m}$, and the second invocation with $C = C_B$, $D = C_M$, $c = c_{B,m}$, $d = c_{M,\phi_{\mathrm{sp}}}$, $\bar{i} = \bar{i}_m$, $u = v_{B,m}$. The two invocations result in two pairs $(\overline{r}, v_M)$, $(\overline{r}', v'_M)$, where $\overline{r}, \overline{r}' \in [\ell]^m$ and $v_M, v'_M \in \mathbb{F}$. Finally, the verifier computes $c_{M,\phi_{\mathrm{sp}}}(\overline{r})$ and $c_{M,\phi_{\mathrm{sp}}}(\overline{r}')$ by itself, accepts if $c_{M,\phi_{\mathrm{sp}}}(\overline{r}) = v_M$ and $c_{M,\phi_{\mathrm{sp}}}(\overline{r}') = v'_M$, and rejects otherwise.

Remark 6.4. The full protocol could be defined slightly differently. Specifically, one could replace the first stage of the protocol with an additional invocation Theorem 6.3 for $t = 0$. This approach has a formal problem, since ψ_0 is not a function but rather a scalar, but the approach can still be implemented by a suitable modification of the relevant definitions. We preferred the current presentation.

Analysis. When given as input a quantified formula over n variables and m occurrences, the foregoing protocol uses $O(m^2)$ rounds: In the first stage, the protocol invokes for each $1 \leq t \leq m-1$ a sum-check protocol of t rounds (twice in parallel), and in the second stage a sum-check protocol of m rounds is invoked (twice). This can be compared to the protocols of [Sha92, She92, GKR08], which use $O(n^2)$ rounds. This difference between our protocol and the previous protocols results from the fact that we work with the specialized formula (7) instead of the original formula (6).

The completeness of the protocol is easy to verify. As for the soundness, note that due to considerations similar to those of the proof of Theorem 6.3, if the input quantified formula is not valid then with probability at least δ it holds that either $c_{A,1}(i_1) \neq v_{A,1}$ or that $c_{B,1}(i_1) \neq v_{B,1}$. By applying the soundness of Theorem 6.3 for the $m-1$ iterations, we get that with probability at least $1 - m \cdot (m+1) \cdot (1-\delta)$ it holds that either $c_{A,m} \neq v_{A,m}$ or that $c_{B,m} \neq v_{B,m}$. Finally, due to the soundness of the sum-check protocol (Theorem 4), we get that with probability at least $1 - m \cdot (1-\delta)$ it holds that either $c_{M,\phi}(\overline{r}) \neq v_M$ or that $c_{M,\phi_{\mathrm{sp}}}(\overline{r}') \neq v'_M$, in which case the verifier rejects. By applying the union bound, it follows that if the input quantified formula is not valid, then the verifier rejects with probability at least $1 - (m+1)^2 \cdot (1-\delta) \geq \frac{1}{2}$, as required.

Acknowledgement. The author is grateful to Oded Goldreich for very helpful discussions and for comments that significantly improved the presentation of this work.

References

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[AW08] Scott Aaronson and Avi Wigderson. Algebrization: a new barrier in complexity theory. In *STOC*, pages 731–740, 2008.

[GI05] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[Köt92] Ralf Kötter. A unified description of an error locating procedure for linear codes. In *Proceedings of the International Workshop on Algebraic and Combinatorial Coding Theory*, pages 113–117, 1992.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[MS88] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*. Elsevier/North-Holland, Amsterdam, 1988.

[Pel92] Ruud Pellikaan. On decoding by error location and dependent sets of error positions. *Discrete Mathematics*, 106-107:369–381, 1992.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. Preliminary version in FOCS 1990.

[She92] Alexander Shen. IP = PSPACE: Simplified proof. *J. ACM*, 39(4):878–880, 1992.

[Sud01] Madhu Sudan. Algorithmic introduction to coding theory (lecture notes), 2001. Available from `http://theory.csail.mit.edu/~madhu/FT01/`.

[Var57] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Doklady Akadamii Nauk*, 117:739–741, 1957.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de