On Locally Verifiable Proofs of Proximity

Tom Gur

Under the Supervision of Professor Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science

Submitted for the degree of Doctor of Philosophy
to the Scientific Council of the Weizmann Institute of Science

February 2017

To Eynat and Roy

Le but de cette thèse est de munir
son auteur du titre de Docteur

– Adrien Douady

Abstract

The study of property testing is concerned with algorithms that solve approximate decision problems, while only probing a small fraction of their inputs. More specifically, a tester for a property Π receives query access to an object x and is required to determine whether $x \in \Pi$ or x is far from Π, using as few queries to x as possible.

A fundamental question that arises in any computational model is to understand the power of proof systems within the model. Indeed, the $\mathcal{P} \neq \mathcal{NP}$ conjecture, which deals with the power of proofs in polynomial time computation, is arguably the most important open problem in the theory of computation. The focus of this thesis is on understanding the power and limitations of proof systems within the framework of property testing.

We study *locally verifiable proofs of proximity* (LVPP), which are probabilistic proofs systems wherein the verifier queries a sublinear number of bits of a statement and is only required to reject statements that are far from valid. In their most basic form, the verifier receives, in addition to query access to the statement, also *free* access to a proof of sublinear length; such proof systems are called *Merlin-Arthur proofs of proximity* (MAP) and can be viewed as the $\mathcal{MA}$ (i.e., "randomized $\mathcal{NP}$") analogue of property testing.

Other notable forms of LVPPs include *interactive proofs of proximity* (IPP), in which the verifier is allowed to communicate with an omniscient prover (rather than obtain a static proof), and *probabilistically checkable proofs of proximity* (PCPP), in which the verifier is only allowed to make a small number of queries to *both* statement and proof (which is typically longer than the statement, in the case of PCPPs). These proofs systems can be viewed as the $\mathcal{IP}$ and $\mathcal{PCP}$ analogues of property testing.

In this thesis, we initiate the study of some types of LVPPs and continue the study of others. Our main contributions include:

- Introducing the notion of non-interactive (Merlin-Arthur) proofs of proximity (MAP) and initiating its systematic study.

- Exponential separations between the power of property testers, MAPs, and IPPs. In particular, denoting by $\mathcal{PT}$, $\mathcal{MAP}$, and $\mathcal{IPP}$ the classes of properties that admit testers and verifiers of polylogarithmic query and communication complexity, we show that $\mathcal{PT} \subsetneq \mathcal{MAP} \subsetneq \mathcal{IPP}$, which can be interpreted as separating $\mathcal{BPP}$, $\mathcal{MA}$, and $\mathcal{IP}$ in the settings of property testing.

- A hierarchy theorem for IPPs, which shows that the power of IPPs gradually increases with the number of rounds of communication allowed between the prover and the verifier.

- Constructions of MAPs and IPPs for several complexity classes, including constraint satisfaction problems (such as 3SAT formulas), properties of graphs, languages accepted by small branching programs, and context-free languages; as well as a strong form of PCPPs for affine subspaces.
- Several constructions of error-correcting codes admitting local features (such as a strong form of local testability, a relaxed form of local decodability, and testability of numerous subcodes) that are useful for constructing LVPPs.

Acknowledgements

First and foremost, it is my pleasure to thank my advisor, Oded Goldreich, for, simply put, being the best. Words cannot express how fortunate I feel to have had Oded's guidance and friendship these last five years. Working with Oded was a delightful experience, full of humor and always inspiring. Thank you Oded for teaching me, sharing your vast knowledge and creativity with me, and always being so kind and generous to me. I am forever grateful for everything you have done for me, and I am very proud to call you my mentor and my friend.

I was also fortunate to collaborate with quite a few remarkable scientists throughout the past few years. I thank my coauthors, Eric Blais, Cleḿent Canonne, Oded Goldreich, Ilan Komargodski, Govind Ramnarayan, Ran Raz, Ron Rothblum, and Omer Tamuz for all they taught me and for making the whole scientific process much more enjoyable.

I would like to thank Robi Krauthgamer and Ran Raz for serving as my Ph.D. committee, and for their advice and insights. I also wish to thank Irit Dinur for enlightening conversations throughout these years.

During my Ph.D. studies I had the pleasure of visiting Rocco Servedio at Columbia University and Eric Blais at the University of Waterloo. I have learned so much from Rocco and Eric, and I am grateful to them for hosting me and creating such a fruitful and enjoyable working atmosphere. I truly hope that our paths will cross again often and soon.

Last, but certainly not least, I would like to thank my family. I am grateful to my parents, Iris and Tzvi, without their unconditional love and support, none of this would have been possible. I thank my brilliant and beautiful wife Eynat, who has been my best friend and my love ever since we were both merely kids, and I thank my son Roy, for being the light of my life, my inspiration, and for making it all worthwhile.

Contents

Chapter 1

Introduction

> No, no! The adventures first,
> explanations take such a dreadful time.
>
> Lewis Carroll, Alice in Wonderland

The study of *property testing*, initiated by Rubinfeld and Sudan [RS96] and by Goldreich, Goldwasser and Ron [GGR98], has attracted significant attention in the last two decades (see, e.g., recent books [Gol10a, Gol17, BY17] and surveys [Ron08, Ron09, Can15]). Loosely speaking, property testers are highly efficient randomized algorithms, usually running in sublinear time, that solve approximate decision problems, while only inspecting a tiny fraction of their inputs. More accurately, a tester T for property Π with respect to proximity parameter $\varepsilon > 0$ is a randomized algorithm that, given query access to an object x, decides whether $x \in \Pi$ or x is ε-far (typically, in relative Hamming distance) from every object in Π. Remarkably, the long line of works in property testing has shown that a myriad of natural properties can be tested by making relatively few queries to the object, often independently of the size of the object.

Once a model of computation has been established, a fundamental question that naturally arises is to understand the power of *proof systems* in this model. Indeed, proof systems play a key role in the theory of computation and their study led to novel reenvisionings of the notion of mathematical proofs, such as *interactive proofs* ($\mathcal{IP}$) and *zero-knowledge proofs* [GMR89] ($\mathcal{ZKP}$), as well as *probabilistically checkable proofs* ($\mathcal{PCP}$) [BGKW88]. Moreover, the famous $\mathcal{P} \neq \mathcal{NP}$ conjecture, which is concerned with understanding the power of proofs in the setting of polynomial time computation, is widely considered as one of the most important open problem in the theory of computation, and perhaps in all of mathematics.

The focus of this thesis is concerned with understanding the power and limitations of proof systems within the framework of property testing. To this end, we study property testers augmented with various probabilistic proof systems,[1] which we call by the collective name of *locally verifiable proofs of proximity* (LVPP), or in short, *proofs of proximity*.

[1] We remark that proof systems that rely on randomness naturally fit property testers, which are inherently probabilistic algorithms.

Loosely speaking, these are probabilistic proofs systems wherein the verifier checks the proximity of a statement to a correct one (i.e., solves an approximate decision problem, in the sense of property testing), by querying a sublinear number of bits of the statement. Such proof systems are being captured by the general notion of *approximate* PCP*s*, introduced by Ergün, Kumar and Rubinfeld [EKR04].

In this thesis, we initiate the study of some types of LVPPs and continue the study of others. From a bird's eye, our main contributions include:

- Introducing the notion of non-interactive proofs of proximity (MAP) and initiating its systematic study.
- Showing MAPs can be exponentially stronger than testers and exponentially weaker than their interactive counterparts, known as *interactive proofs of proximity* (IPP).
- Proving a hierarchy theorem for IPPs, which shows that the power of IPPs gradually increases with the number of rounds of communication.
- Constructing MAPs and IPPs for several natural complexity classes.
- Constructing PCPs of proximity that satisfy a strong type of soundness condition.
- Constructing error-correcting codes admitting local features that are useful for applications to LVPPs.

In Section 1.1 we present the various forms of LVPPs that we study in this thesis and highlight key results about them, then, in Section 1.2 we briefly describe our own contributions to the study of LVPPs. Finally, in the following chapters we will present our results in full detail.

1.1 Locally Verifiable Proofs of Proximity

Recall that property testing deals with *approximate decision problems*, which are promise problems of deciding whether an input is a member of a set (property) Π or far from any element of Π. In this section we describe several types of locally verifiable proofs of proximity, which are probabilistic proof systems for approximate decision problems, and discuss some key results regarding each of them. We begin with the arguably simplest form of LVPPs, known as MA *proofs of proximity*, which we introduced in [GR15b].

1.1.1 MA Proofs of Proximity

We augment the property testing framework by allowing the tester full and free access to a purported proof. Such a proof-aided tester (verifier) for a property Π is given query access to an input x and free access to a proof string w. The verifier is required to distinguish between the case that $x \in \Pi$ and the case that x is far from Π while using a sublinear number of queries. We require that for inputs $x \in \Pi$, there exist a proof that

the tester accepts with high probability, and for inputs x that are *far* from Π no proof will make the tester accept, except with some small probability of error.

This type of proof system can be viewed as the property testing analogue of an NP proof system. However, in contrast to polynomial-time algorithms, sublinear time algorithms inherently rely on *randomization* (cf. [GS10b]). Since an NP proof system in which the verifier is randomized is known as a *Merlin-Arthur* (MA) proof system, we call these sublinear non-interactive proof systems MA *proofs of proximity*, or simply MAPs.

Definition 1.1 (MAP, informally stated)**.** *A* MAP *for a property Π, with respect to proximity parameter $\varepsilon > 0$, is an oracle machine V that is given* query *access to an object x and* free *access to a proof string w, such that the following conditions are satisfied.*

- *Completeness: For every input $x \in \Pi$, there exists a proof w such that*

$$\Pr[V^x(w, \varepsilon) = 1] \geq 2/3,$$

- *Soundness: For every input x that is ε-far from Π and every w' it holds that*

$$\Pr[V^x(w', \varepsilon) = 1] < 1/3.$$

To facilitate the presentation, throughout this chapter, unless the proximity parameter ε is specified, we shall refer to MAPs (and other LVPPs) with respect to a small constant $\varepsilon > 0$. We shall use n to denote the length of the tested object.

Following the property testing literature, it is natural to consider the *query complexity* (i.e., the number of queries that the verifier makes) as a primary resource. Moreover, note that using a proof of length that is linear in the input size, *any* property can be tested using $O(1)$ queries.[2] Hence, we shall also view the *proof length* as a central computational resource that we aim to minimize. This requirement is also quite natural, since property testing is generally concerned with sublinear complexity.

To illustrate the power of MAPs with short proofs, we present the following simple example, due to Fischer, Goldhirsh, and Lachish [FGL14]. Consider the property of strings consisting of two concatenated palindromes (strings that read the same backward or forward), i.e., $\Pi = \left\{xx^Ryy^R \;:\; x, y \in \{0,1\}^*\right\}$, where x^R denotes the string x in reversed order. Alon et al. [AKNS00] showed that testing Π requires $\Omega(\sqrt{n})$ queries. However, note that a MAP with proof of length $\log(n)$ can point to the location wherein one palindrome ends and the other starts, reducing testing Π to two instances of testing a palindrome, which can be easily done (via sampling) with $O(1)$ queries.

We conclude this subsection by mentioning a few of the known results about MAPs (which we will cover in more detail in Sections 1.2.1 and 1.2.2). We know that the gap

[2] To see this, for every property Π, consider a proof system for the statement $x \in \Pi$, wherein the proof w is simply equal to x. To verify the statement, the tester need only verify that indeed $w \in \Pi$ and that w is close to x (i.e., that the relative Hamming distance between w and x is a small constant). The former check can be carried out without any queries to x, whereas for the latter a constant number of queries suffice.

between the complexity of MAPs and testers can be even larger; in particular, there exists a property that has a MAP with proof length $O(\log(n))$ and query complexity $O(1)$, whereas any tester for this property must make $\Omega(n^{0.99})$ queries.[3] On the other hand, we know that in some cases MAPs cannot do better than testers; specifically, there exists a property for which every MAP must make $\Omega(n)$ queries, even given a proof of length, say, $n/100$. Finally, we remark that there exist MAPs with sublinear query and communication complexity for many natural problems, such as languages accepted by small branching programs, context-free languages, and several properties of graphs.

1.1.2 Interactive Proofs of Proximity

The notion of *interactive proofs of proximity* (IPP), introduced by Rothblum, Vadhan, and Wigderson [RVW13], aims to extend the notion of *interactive proofs* to approximate decision problems. In an IPP, instead of a static proof (as in a MAP), the verifier is allowed to interact with an all-powerful, yet untrusted prover who sees the entire tested object.

Definition 1.2 (IPP, informally stated)**.** *An* IPP *for a property* Π *is a pair of interactive strategies that consists of a prover* $\mathcal{P}$ *with* free *access to an object* x *and a verifier* $\mathcal{V}$ *with* query *access to* x*. The parties communicate and satisfy the following conditions.*

- *Completeness: For every input* $x \in \Pi$*, there exists a proof strategy* $\mathcal{P}$ *such that*

$$\Pr[\langle \mathcal{P}(x,\varepsilon), \mathcal{V}^x(\varepsilon)\rangle = 1] \geq 2/3,$$

- *Soundness: For every input* x *that is* far *from* Π *and every proof strategy* $\mathcal{P}'$ *it holds that*

$$\Pr[\langle \mathcal{P}'(x,\varepsilon), \mathcal{V}^x(\varepsilon)\rangle = 1] < 1/3.$$

The main parameters of interest in an IPP are the *query complexity*, which is the number of queries to x made by the verifier, the *communication complexity*, which is the total number of bits exchanged by both parties, and the *round complexity*, which is the number of rounds of communication, where each round consists of a message from one party to the other and vice-versa.

We remark that the notion of IPP generalizes that of MAP. Indeed, every MAP is an IPP in which only one message, from the prover to the verifier, is being sent. As we shall see in Section 1.2.1, this generalization is strict; specifically, there exists properties that have IPPs with polylogarithmic query and communication complexity, whereas every MAP for these properties must make at least $\Omega(\sqrt{n})$ queries, or use a proof of length $\Omega(\sqrt{n})$. Moreover, in Section 1.2.3 we will discuss a hierarchy theorem for IPPs, which shows that the power of IPPs gradually increases with the number of rounds of communication allowed between the prover and the verifier.

[3] Note that this gap is (nearly) the largest possible, since a MAP of proof length p and query complexity q yields a tester of query complexity $\tilde{O}(2^p) \cdot q$, via a simple transformation (see Section 2.4.1).

While currently there are no known constructions of general purpose MAPs (except for relatively low complexity classes such as context-free languages and languages that are accepted by small read-once branching programs), the situation is dramatically different with IPPs: Rothblum, Vadhan, and Wigderson [RVW13] showed that every language in NC has an IPP with query and communication complexities $\tilde{O}(\sqrt{n})$, albeit this IPP requires a large ($\mathsf{polylog}(n)$) number of rounds of interaction. Furthermore, Reingold, Rothblum, and Rothblum [RRR16] showed constant-round IPPs with sublinear query and communication complexity for languages computed in bounded space and time. In addition, in Section 1.2.6 we will present a result that shows that even given as little as three rounds of communication, there exist IPPs with sublinear query and communication complexity for the set of assignments that satisfy fixed constraint satisfaction problems (such as 3SAT instances).

1.1.3 Probabilistically Checkable Proofs of Proximity

The next form of LVPP we discuss is called PCP*s of proximity* (PCPP), and as its name implies, it can be thought of as the PCP analogue of property testing. The notion of PCPP was first studied by Ben-Sasson et al. [BSGH+06] and by Dinur and Reingold [DR06] (wherein they are called *assignment testers*), motivated by applications to standard PCPs.

Recall that a standard PCP is given explicit access to a statement (i.e., an input that is supposedly in some NP language) and oracle access to a proof (i.e., a "probabilistically checkable" NP witness). The PCP verifier is required to probabilistically verify whether the (explicitly given) statement is correct, while making few queries to the alleged proof. In contrast, a PCPP is given oracle access both to a statement and to a (potentially long) proof and is only allowed to make a small number of queries to each of them. As in the other forms of LVPP, the verifier is only required to accept correct statements and reject statements that are far from being correct.

Definition 1.3 (PCPP, informally stated)**.** *A* PCPP *for a property* Π*, with respect to proximity parameter* $\varepsilon > 0$*, is an oracle machine* V *that is given* query *access to both an object* x *and a proof string* w*, such that the following conditions are satisfied.*

- *Completeness: For every input* $x \in \Pi$*, there exists a proof* w *such that*

$$\Pr[V^{x,w}(\varepsilon) = 1] \geq 2/3,$$

- *Soundness: For every input* x *that is* ε-far *from* Π *and every purported proof* w'*, it holds that*

$$\Pr[V^{x,w'}(\varepsilon) = 1] < 1/3.$$

The main parameters of interest in a PCPP are its *query complexity* (i.e., the total number of queries to the input and to the proof that the verifier makes) and its proof *length*, which can be thought as measuring the amount of redundancy of information in the proof.

We stress that in stark contrast to MAPs and IPPs, in which a proof of linear length trivialize the models, obtaining PCPPs of even polynomial length is extremely non-trivial.

Nevertheless, Ben Sasson et al. [BSGH+06] showed a PCPP for 3SAT with constant query complexity and length $n^{1+o(1)}$; furthermore, Dinur [Din07a] (building on [BS05]) showed that the length of the PCPP can be reduced to $n \cdot \mathsf{polylog}(n)$.

We remark that PCPPs were shown to be useful in various applications, including PCP composition and alphabet reduction [BSGH+06, DR06], as well as construction of locally testable and locally decodable codes. We further discuss such applications in Section 1.2.4.

We conclude this section with Table 1.1, which presents a taxonomy of probabilistic proof systems, according to the type of access given to the input and proof (or prover).

	Access to Main Input	
Access to Prover	**Free Access**	**Query Access**
No Proof	BPP	Testers
Non-Interactive, Free	MA	MAP
Non-Interactive, Query	PCP	PCPP
Interactive, Free	IP	IPP

Table 1.1: Taxonomy of probabilistic proof systems.

1.2 Our Results

In this section we provide brief descriptions of our main contributions to the study of locally verifiable proofs of proximity. Full details will follow in the subsequent chapters.

1.2.1 Non-Interactive Proofs of Proximity

Together with Ron Rothblum [GR15b], we introduced the notion of *non-interactive* proofs of proximity, known as *Merlin-Arthur proofs of proximity* (MAP), and initiated their study. Recall that MAPs can be viewed as the $\mathcal{NP}$ (or more accurately $\mathcal{MA}$) analogue of *property testing*. We explored both the power and limitations of MAPs, studied the relation of MAPs to other proofs of proximity, as well as showed MAPs with sublinear query and communication complexity for several natural properties, including parameterized concatenation problems and properties of graphs.

Our main results include showing:

- MAPs can be exponentially *stronger* than testers: There exists a property that has a MAP with proof length $O(\log(n))$ and query complexity $O(1)$, whereas any tester for this property must make $\Omega(n^{0.99})$ queries.

- MAPs can be exponentially *weaker* than IPPs: There exists a property that has a polylog(n)-round IPP with polylog(n) query and communication complexity, whereas any MAP for this property must make at least $\Omega(\sqrt{n})$ queries, or use a proof of length $\Omega(\sqrt{n})$.
- An extremely hard property for MAPs: There exists a property for which every MAP must make $\Omega(n)$ queries, even given a proof of length $n/100$.
- MAPs with query/proof complexity tradeoff: There exists a property Π such that, for every $p \geq 1$, there is an MAP for Π that uses a proof of length p and makes $\frac{n^{0.999}}{p}$ queries. Furthermore, for every p, the trade-off is (almost) tight.
- MAPs can cheaply obtain one-sided error: Any MAP can be transformed into a MAP that accepts valid inputs with probability 1, at the cost of only increasing its proof and query complexities by at most a polylog(n) factor.

See Chapter 2 for details.

1.2.2 Proofs of Proximity for Context-Free Languages and Read-Once Branching Programs

To further demonstrate the usefulness of both interactive and non-interactive proofs of proximity, together with Oded Goldreich and Ron Rothblum [GGR15], we showed how to construct proofs of proximity for two natural classes of properties: (1) context-free languages, and (2) languages accepted by small read-once branching programs. Our main results are:

- MAPs for these two classes, in which, for inputs of length n, both the verifier's query complexity and the length of the proof are $\tilde{O}(\sqrt{n})$.
- IPPs for the same two classes, with constant query complexity, poly-logarithmic communication complexity, and logarithmically many rounds of interaction.

See Chapter 3 for details.

1.2.3 A Hierarchy Theorem for Interactive Proofs of Proximity

As we discussed in Section 1.1, the number of rounds, or round complexity, used in any interactive protocol is a fundamental resource. Together with Ron Rothblum [GR17], we considered the significance of round complexity in the context of *Interactive Proofs of Proximity* (IPPs).

Our main result is a round hierarchy theorem for IPPs, showing that the power of IPPs grows with the number of rounds. More specifically, we showed that there exists a gap function $g(r) = \Theta(r^2)$ such that for every constant $r \geq 1$ there exists a language that (1) has a $g(r)$-round IPP with verification time $t = t(n, r)$ but (2) does not have an r-round IPP with verification time t (or even verification time $t' = \text{poly}(t)$).

In fact, we proved a stronger result by exhibiting a *single* language $\mathcal{L}$ such that, for every constant $r \geq 1$, there is an $O(r^2)$-round IPP for $\mathcal{L}$ with $t = n^{O(1/r)}$ verification time, whereas the verifier in *any* r-round IPP for $\mathcal{L}$ must run in time at least t^{100}. Moreover, we showed an IPP for $\mathcal{L}$ with a poly-logarithmic number of rounds and only poly-logarithmic verification time, yielding a sub-exponential separation between the power of constant-round IPPs versus general (unbounded round) IPPs.

From our hierarchy theorem we also derived implications to standard interactive proofs (in which the verifier can run in polynomial time). Specifically, we show that the round reduction technique of Babai and Moran [BM88] is (almost) optimal among all blackbox transformations, and we showed a connection to the algebrization framework of Aaronson and Wigderson [AW09]. See Chapter 4 for details.

1.2.4 Strong Locally Testable Codes with Relaxed Local Decoders

The separation between the power of testers and MAPs, which was discussed in Section 1.2.1, heavily relies on error-correcting codes with local features. Motivated to improve the foregoing separation, together with Oded Goldreich and Ilan Komargodski [GGK15], we studied the problem of constructing codes that simultaneous exhibit both a strong form of testability and relaxed form of decodability.

Locally testable codes (LTCs), whose systematic study was initiated by Goldreich and Sudan [GS06], are *error-correcting codes* that admit very efficient codeword tests. An LTC is said to be strong if it has a *proximity-oblivious* tester; that is, a tester that makes only a *constant number* of queries and reject non-codewords with probability that depends solely on their distance from the code.

Locally decodable codes (LDCs), introduced by Katz and Trevisan [KT00], are complimentary to LTCs. While the latter allow for highly efficient rejection of strings that are far from being codewords, LDCs allow for highly efficient recovery of individual bits of the information that is encoded in strings that are close to being codewords.

Constructions of strong-LTCs with nearly-linear length are known, but the existence of a constant-query LDC with *polynomial* length is a major open problem. In an attempt to bypass this barrier, Ben-Sasson et al. [BSGH+06] introduced a natural relaxation of local decodability, called relaxed-LDCs. This notion requires local recovery of nearly all individual information-bits, yet allows for recovery-failure (but not error) on the rest. Ben-Sasson et. al. constructed a constant-query relaxed-LDC with nearly-linear length (i.e., length $k^{1+\alpha}$ for an arbitrarily small constant $\alpha > 0$, where k is the dimension of the code).

We focused on obtaining strong testability and relaxed decodability *simultaneously*. We constructed a family of binary linear codes of nearly-linear length that are both strong-LTCs (with one-sided error) and constant-query relaxed-LDCs. This improved upon the previously known constructions, which either obtain weak LTCs or require polynomial length.

Our construction heavily relies on *tensor codes* and PCPs. In particular, we provided

strong canonical PCPs *of proximity* for membership in any linear code with constant rate and relative distance. Loosely speaking, these are PCPs *of proximity* wherein the verifier is proximity oblivious (similarly to strong-LTCs) and every valid statement has a unique *canonical* proof. Furthermore, the verifier is required to reject non-canonical proofs (even for valid statements).

Using these new codes, we indeed improved the best known separation result between the power of testers and MAPs. See Chapter 5 for details.

1.2.5 Universal Locally Testable Codes

Motivated by applications to interactive proofs of proximity, together with Oded Goldreich [GG16a], we initiated a study of "universal locally testable codes" (universal-LTCs). These codes admit local tests for membership in numerous possible subcodes, allowing for testing properties of the encoded message. More precisely, a universal-LTC $C : \{0,1\}^k \to \{0,1\}^n$ for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ is a code such that for every $i \in [M]$ the subcode $\{C(x) \ : \ f_i(x) = 1\}$ is locally testable.

We showed a "canonical" $O(1)$-local universal-LTC of length $\tilde{O}(M \cdot s)$ for any family $\mathcal{F}$ of M functions such that every $f \in \mathcal{F}$ can be computed by a circuit of size s, and established a lower bound of the form $n = M^{1/O(k)}$, which can be strengthened to $n = M^{\Omega(1)}$ for any $\mathcal{F}$ such that every $f, f' \in \mathcal{F}$ disagree on a constant fraction of their domain. See Chapter 6 for details.

1.2.6 Universal Locally Verifiable Codes and 3-Round Interactive Proofs of Proximity for CSP

Further investigating the notion of universal-LTCs, together with Oded Goldreich [GG16b], we initiated a study of the NP analogue of these codes, wherein the testing procedures are also given free access to a short proof, akin to MAPs. We called such codes "universal locally *verifiable* codes" (universal-LVCs). A universal-LVC $C : \{0,1\}^k \to \{0,1\}^\eta$ for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ is a code such that for every $i \in [M]$, membership in the subcode $\{C(x) \ : \ f_i(x) = 1\}$ can be verified locally given an explicit access to a short (sublinear length) proof.

We showed universal-LVCs of block length $\tilde{O}(n^2)$ for the family of all functions expressible by t-ary constraint satisfaction problems (t-CSP) over n constraints and k variables, with proof length and query complexity $\tilde{O}(n^{2/3})$, where $t = O(1)$ and $n \geq k$. In addition, we proved a lower bound of $p \cdot q = \tilde{\Omega}(k)$ for every polynomial length universal-LVC, having proof complexity p and query complexity q, for such CSP functions.

Lastly, we gave an application for interactive proofs of proximity. Specifically, we showed a 3-round IPP for the set of assignments that satisfy fixed CSP instances, with sublinear communication and query complexity, which we derived from our universal-LVC for CSP functions. See Chapter 7 for details.

1.3 Organization

The rest of the thesis is organized in chapters, where each chapter contains a full version of a published paper or a paper that is currently in submission. Hence, each chapter may be read independently of all other chapters. We note that the first section of each chapter ends with an organization subsection that outlines the structure of that chapter, and appendices relevant to each chapter appear immediately at the end of the relevant chapter.

In Chapter 2 we present our results regarding MAPs (based on [GR15b]). In Chapter 3 we show how to construct MAPs and IPPs for context-free languages and languages accepted by small read-once branching programs (based on [GGR15]). In Chapter 4 we show a hierarchy theorem for IPPs (based on [GR17]). In Chapter 5 we show a construction of error-correcting code with local features, which is used to prove a stronger separation between the power of testers and MAPs (based on [GGK15]). In Chapter 6 and Chapter 7 we present our results regarding universal-LTCs and universal-LVCs, which in turn enables us to construct our constant-round IPPs for constraint satisfaction problems.

Finally, in Chapter 8 we give high-level overviews of works that were not included in this thesis but were obtained during our doctoral studies. These works are related to property testing but not directly to locally verifiable proofs of proximity.

Chapter 2

Non-Interactive Proofs of Proximity

2.1 Introduction

Understanding the power and limitations of sublinear algorithms is a central question in the theory of computation. The study of *property testing*, initiated by Rubinfeld and Sudan [RS96] and Goldreich, Goldwasser and Ron [GGR98], aims to address this question by considering highly-efficient randomized algorithms that solve approximate decision problems, while only inspecting a small fraction of the input. Such algorithms, commonly referred to as *property testers*, are given oracle access to some object, and are required to determine whether the object has some predetermined property, or is far (say, in Hamming distance) from every object that has the property. Remarkably, it turns out that many natural properties can be tested by making relatively few queries to the object.

Once a model of computation has been established, a fundamental question that arises is to understand the power of *proof-systems* in this model. Recall that a proof-system consists of a powerful prover that wishes to convince a weak verifier, which does not trust the prover, of the validity of some statement. Since verifying is usually easier than computing, using the power of proofs, it is often possible to overcome limitations of the basic model of computation. In this paper we study proof-systems in the context of property testing, with the hope that by augmenting testers with proofs we can indeed overcome inherent limitations of property testers.

Thus, we are interested in proof-systems in which the verifier reads only a small fraction of the input. Of course we cannot hope for such a verifier to reject *every* false statement. Instead, as is the case in property testing, we relax the soundness condition and only require that it be impossible to convince the verifier to accept statements that are *far* from true statements. Such proof-systems were first introduced by Ergün, Kumar and Rubinfeld [EKR04] and were recently further studied by Rothblum, Vadhan and Wigderson [RVW13] who were motivated by applications to *delegation of computation* in sublinear time. Rothblum *et al.* [RVW13] showed that by allowing a property tester to interact with an untrusted prover (who can read the *entire* input), sublinear time verification is indeed possible for a wide class of properties. As in the property testing framework, the tester is only assured of the proximity of the input to the property and

hence such protocols are called *interactive proofs of proximity* (IPPs).

2.1.1 The Notion of MAP

In this work, we also consider proofs of proximity, but restrict the verification process to be *non-interactive*. In other words, we augment the property testing framework by allowing the tester full and free access to an (alleged) proof. Such a proof-aided tester for a property Π, is given oracle access to an input x and free access to a proof string w, and should distinguish between the case that $x \in \Pi$ and the case that x is far from Π while using a sublinear number of queries. We require that for inputs $x \in \Pi$, there exist a proof that the tester accepts with high probability, and for inputs x that are *far* from Π no proof will make the tester accept, except with some small probability of error.

This type of proof-system can be viewed as the property testing analogue of an NP proof-system (whereas IPP is the property testing analogue of IP). However, in contrast to polynomial-time algorithms, sublinear time algorithms inherently rely on *randomization*.[1] Since an NP proof-system in which the verifier is randomized is known as a *Merlin-Arthur* (MA) proof-system, we call these sublinear non-interactive proof-systems *Merlin-Arthur proofs of proximity* or simply MAPs.

Following the property testing literature, we consider the number of queries that the tester makes as the main computational resource. We ask whether non-interactive proofs can reduce the number of queries that property testers make, and if so by how much. (We note that [RVW13] showed that it is possible to significantly reduce the query complexity of property testers using interactive proofs, but their proof systems rely fundamentally on two-way interaction.)

Given the (widely believed) power of proofs in the context of *polynomial-time* computation, one would hope that proofs can help decrease the number of queries that is needed to test various properties. This is indeed the case. In fact, for every property Π, consider a proof-system for the statement $x \in \Pi$, wherein the proof w is simply equal to x. In order to verify the statement, the tester need only verify that indeed $w \in \Pi$ and that w is close to x (i.e., that the relative Hamming distance between w and x is a small constant). The former check can be carried out without any queries to x, whereas for the latter a constant number of queries suffice. Thus, using a proof of length linear in the input size, *any* property can be tested using a constant number of queries (furthermore, the tester has one-sided error). In contrast, there exist properties for which *linear* lower bounds on the query complexity of standard property testers are known (cf. [GGR98]).

The foregoing discussion leads us to view the proof length, in addition to the number of queries, as a central computational resource, which we should try to minimize. Thus, we measure the complexity of an MAP by the total amount of information available to the tester, namely, the sum of the MAPs query complexity (i.e., the number of queries that the tester makes) and proof complexity (i.e., the length of the proof). In this work

[1] It is not difficult to see that the sublinear time *deterministic* computation or even verification is limited to trivial properties (cf. [GS10b]).

we study the complexity of MAPs in comparison to property testers and to the recently introduced IPPs.

A Concrete Motivation. We note that the non-interactive nature of such proof-systems may have significant importance to applications such as *delegation of computation*. Specifically, consider a scenario wherein a computationally weak client has reliable query access to a massive dataset x. The client wishes to compute a function f on x, but its limited power, along with the massive size of the dataset, prevents it from doing so. In this case, the client can use a powerful server (e.g., a cloud computing provider) to compute $f(x)$ for it. However, the client may be distrustful of the server's answer (as it might cheat or make a mistake). Thus, an MAP for f can be used to verify the correctness of the computation delegated to the server: Given access to x, the server can send the value $y = f(x)$, together with a proof of proximity that ascertains that x is close to a dataset x' for which $f(x') = y$. The latter can be verified using an MAP verifier that makes only a small number of queries to x.

We emphasize that the advantage in using *non-interactive* proofs of proximity (rather than interactive ones) is not only in removing the need for two-way communication, but also: (1) the proof can be "annotated" to the dataset by the server in a cheap off-line phase; and (2) the proof can be re-used for multiple clients.

The Computational Complexity of Generating and Verifying the Proof. As noted above, we view the number of queries and proof length as the main computational resources. It is natural to also consider the computational complexity of generating and verifying the proof. However, in this work our main focus is on the query and proof complexities. Still, we note that unless stated otherwise, our protocols can be implemented efficiently; that is, the proof can be generated in *polynomial-time* and verified in *sublinear-time*.

Comparison with PCPs of Proximity. PCPs of proximity (PCPPs), first studied by Ben-Sasson *et al.* [BSGH+06] and by Dinur and Reingold [DR06] (where they are called assignment testers) are also non-interactive proof-systems in which the verifier has oracle access to an object, and needs to decide whether the object is close to having a predetermined property. However, PCPPs differ from MAPs in that the verifier is only given *query* (i.e., oracle) access to the proof, whereas in MAPs, the verifier has free (*explicit*) access to the proof. Indeed, in contrast to MAPs, the proof string in PCPPs is typically of super-linear length (but only a small fraction of it is actually read at random). Thus, PCPPs may be thought of as the PCP analogue of property testing, whereas MAPs are the NP analogue of property testing.

In fact, considering a variety of non-interactive proof-systems that differ in whether the main input and the proof are given explicitly or implicitly (i.e., via query access or free access), leads to the taxonomy depicted in Table 2.1. Interestingly, the three other variants, corresponding to NP, PCP and PCPP, have all been well studied. Thus, we view the notion of MAPs as completing this taxonomy of non-interactive proof-systems.

	Access to Proof		
Access to Main Input	**No Proof**	**Free Access**	**Oracle Access**
Free Access	$\mathcal{P}$	NP or MA	PCP
Oracle Access	Property Testers	MAP (**this work**)	PCPP

Table 2.1: Taxonomy of non-interactive proof-systems.

2.1.2 The Power of MAP

The first question that one might ask about the model of MAPs is whether proofs give a significant savings in the query complexity of property testers (indeed, such savings are the main reason to introduce a proof-system in the first place). Given the above discussion on the importance of bounding the proof length, we seek savings in the query complexity while using only a relatively short proof. Our first result shows that indeed there exists a property for which a dramatic saving is possible:

Informal Theorem 2.1 (see Theorem 2.7)**.** *There exists a (natural) property that has an* MAP *that uses a logarithmic-length proof and only a* constant *number of queries, but requires* $n^{0.999}$ *queries for every property tester.*
Here and throughout this work, n denotes the length of the object being tested.

Having established an exponential separation between property testers and MAPs, we continue our study of MAPs by asking how many queries can be saved by slightly increasing the length of the proof. The following result shows a property for which a smooth *multiplicative* trade-off, which is (almost) tight, between the number of queries and length of the proof holds:

Informal Theorem 2.2 (see Theorem 2.13)**.** *There exists a (natural) property* Π *such that, for every* $p \geq 1$*, there is an* MAP *for* Π *that uses a proof of length* p *and makes* $\frac{n^{0.999}}{p}$ *queries. Furthermore, for every* p*, the trade-off is (almost) tight.*

Recall that for property testers huge gaps may exist between the query complexity of testers that have *one-sided error* and the query complexity of testers that have two-sided error (where a one-sided tester is one that accepts every object that has the property with probability 1). Notable examples for properties for which such gaps are known are *Cycle-Freeness* in the bounded degree graph model (see [CGR+12]) and *ρ-Clique* in the dense graph model (see [GGR98]). In contrast, we observe that *such gaps can not exist in the case of* MAP*s.*

Informal Theorem 2.3 (see Theorem 2.20)**.** *Any two-sided error* MAP *can be converted to have* one-sided error *with only a poly-logarithmic overhead to the query and proof complexities.*

Since every property tester can be viewed as an MAP that uses an empty proof, as an immediate corollary, we obtain a transformation from every two-sided error *property tester* into a one sided MAP that uses a proof of only poly-logarithmic length (with only a poly-logarithmic increase in the query complexity). Moreover, since (as noted above) there are well-known properties for which *one-sided error* property testing is exponentially harder than *two-sided error* property testing, Informal Theorem 2.3 implies an exponential separation between MAPs (with poly-logarithmically long proofs) and *one-sided error* property testing. We note that Informal Theorem 2.1 shows such a separation for the more general case of two-sided error.

We note that all of the explicit properties that were discussed thus far are properties "with distance"; that is, properties for which every two objects that have the property are far apart. In other words, the set of objects forms an error-correcting code. This distance, along with a form of local *self-correction*, is a crucial ingredient of the foregoing MAPs. In contrast, all of the properties described next are properties "without distance". Hence, the power of MAPs is not limited to properties with distance.

MAPs for parameterized concatenation problems. We identify a family of natural properties, for which it is possible to construct efficient MAPs, by using a generic scheme. Specifically, for every problem that can be expressed as a parameterized concatenation problem, we show how to construct an efficient MAP that allows a trade-off between the query and proof complexity. Loosely speaking, a property Π is a parameterized concatenation problem if $\Pi = \Pi_{\alpha_1} \times \cdots \times \Pi_{\alpha_k}$, for some integer k, where each property Π_{α_i} is a property parameterized by α_i.

Using this generic scheme, we obtain MAPs for a couple of natural problems, including: (1) approximating the Hamming weight of a string, and (2) graph orientation problems. (For more details, see Section 2.6).

MAPs for graph properties. To see that MAPs are also useful for testing graph properties, we consider the problem of testing bipartiteness in the *bounded-degree* graph model. We construct an MAP protocol for verifying bipartiteness of *rapidly-mixing graphs*, with proof complexity p and query complexity q, *for every* p and q such that $p \cdot q \geq N$ (where N is the number of vertices in the graph). In particular, we obtain an MAP verifier that uses a proof of length $N^{2/3}$ and makes only $N^{1/3}$ queries. This stands in contrast to the $\Omega(\sqrt{N})$ lower bound on the query complexity of property testers (which do not use a proof), shown by Goldreich and Ron [GR02], which also holds for *rapidly-mixing graphs*. We remark that in [RVW13] a (multi-round) IPP was given for the same problem (see Section 2.7).

We note that in the *dense* graph model, testing bipartiteness (or more generally k-colorability) can be easily done using only $O(1/\varepsilon)$ queries (where ε represents the desired proximity to the object) when given a proof that is simply the k-coloring of the graph (which can be represented by $N \log_2 k$ bits where N is the number of vertices and k is the

number of colors).[2] In contrast, for standard property testers such query complexity is impossible (see [BT04]). We note that a similar protocol (described as a PCPP) for testing bipartiteness in the dense graph model was suggested in [EKR04] and in [BSGH+06].

MAPs for sparse properties. If a property is relatively sparse, in the sense that it contains only t objects, then a proof of length $\log_2 t$ (which fully describes the object) can be used, and only $O(1/\varepsilon)$ queries suffice to verify the proof's consistency with the object. Using this observation we note that testing k-juntas and k-linearity can be verified using only $O(1/\varepsilon)$ queries and a proof of length $O(k \log n)$, whereas a lower bound of $\Omega(k)$ queries is well-known for standard property testers (cf. [Bla10]).

2.1.3 The Limitations of MAP

In the previous section, we described results that exhibit the power of MAPs. But what are the limitations of MAPs? As discussed above, a proof of linear length suffices to reduce the query complexity to $O(1/\varepsilon)$. Moreover, Informal Theorem 2.1 shows that even a logarithmically long proof can be extremely useful for a specific property. Thus, it is natural to ask whether a *sublinear* proof can reduce the query complexity for *every* property. The following result shows that for *almost all* properties, even a proof of length $n/100$ cannot improve the query complexity by more than a constant factor.

Informal Theorem 2.4 (see Theorem 2.22)**.** *For almost all properties, every* MAP *verifier that uses a proof of length* $n/100$ *must make* $\Omega(n)$ *queries.*[3]

Although Theorem 2.22 holds for most properties, finding an *explicit* property for which a similar statement holds remains an interesting open question. We note that Informal Theorem 2.4 improves upon a result of Fischer *et al.* [FGL14] (see discussion in Section 2.1.5).

Since Informal Theorem 2.4 shows that even a relatively long proof cannot help in general for *every* property, one might ask whether there are specific properties for which short proofs do suffice. As was shown in Informal Theorem 2.1, this is indeed the case and a logarithmically long proof allows for an exponential improvement in the query complexity for a specific property. But can an even shorter, say constant-size proof, help? Unfortunately, the answer is negative since an MAP with query complexity q and proof complexity p can be emulated by a property tester that enumerates all possible proofs and makes a total of $\tilde{O}(2^p \cdot q)$ queries. Still, are there any further limits to how proofs can help a tester?

We first note that the ability to query the object in a way that depends on the proof is essential to the power of MAP. In contrast, consider *proof-oblivious queries* MAPs, which are MAPs in which the verifer's queries are independent of the provided proof. Such

[2] Note that the size of the tested object is N^2, and so $N \log_2 k$ is sublinear in the input size. In order to verify this proof, the verifier chooses $O(1/\varepsilon)$ edges at random and accepts if all are properly colored.

[3] In fact, we show a general additive tradeoff between proof and query complexities, that is, every MAP verifier that uses a proof of length p must make $\tilde{\Omega}(n - p)$ queries.

MAPs can be viewed as a two step process in which the verifier first (adaptively) queries the object and only then it receives the proof and decides whether to accept or reject based on both the answers and the proof. We say that such MAPs have proof oblivious queries. The following result shows that MAPs with *proof-oblivious queries* can provide at most a *quadratic* improvement over standard property testers.

Informal Theorem 2.5 (see Theorem 2.19)**.** *If a property* Π *has an* MAP *that makes* q proof oblivious *queries and uses a proof of length* p*, then* Π *has a property tester that makes* $O(q \cdot p)$ *queries.*

By Informal Theorem 2.1, the restriction to *proof oblivious queries* is a necessary precondition for Informal Theorem 2.5 (and indeed, the MAP verifier of Informal Theorem 2.1 must make proof-dependent queries).

Having inspected the relationship between MAPs and property testing, we proceed to consider the relationship between MAPs and IPPs. Recall that MAPs are actually a special case of IPPs in which the interaction is limited to a single message sent from the prover to the verifier. When comparing MAPs and IPPs it is natural to compare both the query complexity and the total amount of communication with the prover (which in the case of MAPs is simply the length of the proof).

The following theorem shows that IPPs are stronger than MAPs not only syntactically but also in essence. We show that even 3-message IPPs may have exponentially better query complexity than MAPs (while using the same amount of communication). Moreover, we show that IPPs with *poly-logarithmically* many messages of poly-logarithmic length can also have exponentially better communication complexity.

Informal Theorem 2.6 (see Theorem 2.16 and Theorem 2.17)**.** *There exists a property* Π *such that on the one hand, any* MAP *for* Π *with proof of length* $n^{0.499+o(1)}$ *has query complexity* $n^{0.499+o(1)}$*, and on the other hand,* Π *has:*

1. *A* 3*-message* IPP *that makes* $\mathsf{polylog}(n)$ *queries while using a total of* $n^{0.499+o(1)}$ *communication.*
2. *An* IPP *with only* $\mathsf{polylog}(n)$ *query and communication complexities but using a* poly-logarithmic *number of messages.*

2.1.4 Techniques

Several of our results (in particular Informal Theorems 2.2 and 2.6) are based on a specific algebraic property, which we call *Sub-Tensor Sum* and denote by TensorSum (c.f. [LFKN92]). Let $\mathbb{F}$ be a finite field and let $H \subset \mathbb{F}$ be an arbitrary subset. We consider m-variate polynomials over $\mathbb{F}$ that have individual degree d. The TensorSum property contains all such polynomials whose sum on H^m equals 0.[4] That is, TensorSum contains all polynomials $P : \mathbb{F}^m \to \mathbb{F}$ of individual degree d such that

$$\sum_{x \in H^m} P(x) = 0.$$

[4]The choice of the constant 0 is arbitrary.

Selecting $|\mathbb{F}|, m, d$ and $|H|$ suitably (as poly-logarithmic functions in the input size $n = |\mathbb{F}|^m$), we obtain the following roughly stated upper and lower bounds for TensorSum (for the formal statements, see the technical sections):

1. **PT**: The query complexity of testing TensorSum (without a proof) is $\Theta(n^{0.999\pm o(1)})$ queries.

2. **MAP**: The $\mathcal{MAP}$ complexity of TensorSum is $\Theta\left(n^{0.499\pm o(1)}\right)$. Moreover, for every $p \geq 1$, the MAP query complexity of TensorSum with respect to proofs of length p is $\Theta\left(\frac{n^{0.999\pm o(1)}}{p}\right)$.

3. **IPP[3]**: TensorSum has a 3-*message* IPP with query complexity $\mathsf{polylog}(n)$ and communication complexity $O\left(n^{0.499+o(1)}\right)$.

4. **IPP**: TensorSum has an IPP with query and communication complexities $\mathsf{polylog}(n)$. However, in contrast to Item 3, this IPP uses *poly-logarithmically* many messages.

To get a taste of our proofs, consider the (relatively) simple case wherein we restrict the TensorSum property to dimension $m = 2$ and a field $\mathbb{F}$ of size $\sqrt{n}$ (i.e., bivariate polynomials over a field of size $\sqrt{n}$). Naturally, we call this variant the *Sub-Matrix Sum* property and denote it by MatrixSum. Note that MatrixSum contains all polynomials $P : \mathbb{F}^2 \to \mathbb{F}$ of individual degree $d = |\mathbb{F}|/10$ such that

$$\sum_{x,y \in H} P(x, y) = 0.$$

As an MAP proof to the claim that the polynomial P is in MatrixSum, consider the univariate polynomial $Q(x) \stackrel{\text{def}}{=} \sum_{y \in H} P(x, y)$. To verify that P is indeed in MatrixSum the verifier acts as follows:

1. If $\sum_{x \in H} Q(x) \neq 0$, then reject.

2. Verify that P is (close to) a low degree polynomial and reject if not. This can be done with $O(d)$ queries via the classical low degree test (see Theorem 2.30).

3. Verify that Q is consistent with P. Since both are low degree polynomials, it suffices for the verifier to check that $Q(r) = \sum_{y \in H} P(r, h)$ for a random $r \in \mathbb{F}$.

   Actually, a technical difficulty arises from the fact that P can only be verified to be *close* to a low degree polynomial. The naive solution of reading every point via self-correction is too expensive in the case of MatrixSum. While it is possible to overcome this difficulty using a slightly more sophisticated technique (to appear in a forthcoming revision), the naive solution suffices for our actual setting of parameters (for TensorSum) and so we ignore this difficulty here.

By setting $|H| = O(|\mathbb{F}|)$ we obtain an MAP with proof and query complexity $O(\sqrt{n})$ (since $n = |\mathbb{F}|^2$). Using more sophisticated techniques in the same spirit, we obtain both MAP and IPP upper bounds for the TensorSum problem.[5]

Parameterized Concatenation Problems. Our techniques for showing MAPs for properties that do not have distance (and a structure that allows for self-correction) differ from the above. One class of problems that we consider is that of *parameterized concatenation problems*. Such properties consists of strings that are a concatenation of substrings, where each substring satisfies a particular parameterized property. The actual parameterization is not known a priori to the tester, and so an MAP proof that simply provides this parameterization turns out to be quite useful. Given this parameterization, the MAP verifier can simply test each substring individually (or a random subset of these substrings). Actually, in order to solve the problem more efficiently, the different substrings are tested with respect to different values of the proximity parameter by using a technique known as *precision sampling* (see survey [Gol14, Appendix A]).

Verifying Bipartiteness of Well-Mixing Graphs. Our MAP protocol for proving bipartiteness of a given *well-mixing* graph $G = (V, E)$ of size $N = |V|$ proceeds as follows. The proof consists of a subset $W \subseteq V$ of vertices that are allegedly on the same side of the graph. The verifier selects a random vertex $s \in V$ and takes roughly $N/|W|$ random walks of length $\Theta(\log n)$, starting at s. The verifier rejects if two of the walks pass through vertices of the set W, where the lengths of the paths from s to these vertices of W have opposite parities. Indeed, such walks cannot occur in bipartite graphs, assuming that all vertices in S are on the same side.

We show that if the graph is rapidly mixing and far from bipartite, then, for a $O(1/\log(N))$ fraction of vertices $s \in W$, the probability that a random walk starting in s will end in W with odd (respectively, even) parity is roughly $|W|/N$. Since the verifier takes $N/|W|$ random walks starting in s, with constant probability, it will detect a violation and reject. The analysis of our protocol is inspired by [GR02]. Interestingly, in contrast to the analysis of the rapidly-mixing case in [GR02], our analysis crucially relies on the random selection of the starting vertex.

Lower Bounds via MA Communication Complexity. As for our property testing *lower bounds*, we base these on the recently introduced technique of Blais, Brody and Metulef [BBM11]. The [BBM11] methodology enables one to obtain property testing lower bounds from *communication complexity* lower bounds. To obtain MAP lower bounds, we extend the [BBM11] framework. We show that lower bounds on the MA *communication complexity* of a communication complexity problem related to a property Π can be used to derive lower bounds on the MAP *complexity* of Π.

[5] We use TensorSum rather than MatrixSum because we do not know how to obtain an IPP nor a *full* trade-off between proof and query complexities for MatrixSum.

MA communication complexity, introduced by Babai, Frankl and Simon [BFS86], extends standard communication complexity by adding a third player, Merlin, who sees both the input x of Alice and y of Bob and attempts to convince them that $f(x, y) = 1$ where f is the function that they are trying to compute. We require that if $f(x, y)$ indeed equals 1, then there exist a proof for which Alice and Bob output the correct value (with high probability), but if $f(x, y) = 0$, then no proof will cause them to output a wrong value (except with some small error probability).

In order to show lower bounds for MAP we are thus left with the task of showing lower bounds for related MA communication complexity problems. Fortunately, Klauck [Kla03] showed a strong lower bound for the set-disjointness problem, which we use in our reductions. Additionally, we extend a recent result of Gur and Raz [GR13b] who give an MA communication complexity lower bound on the classical problem of *Gap Hamming Distance*.

We note that nearly all of the lower bounds shown in [BBM11] are proved via reductions from the communication complexity problems of *set-disjointness* and *gap Hamming distance*. Since these communication complexity problems have known MA communication complexity lower bounds (cf. [Kla03, GR13b]), these reductions, together with our extension of the [BBM11] framework to MAPs, gives MAP *lower bounds for the problems studied in [BBM11]* (e.g., testing juntas, Fourier degree, sparse polynomials, monotonicity, etc.).

Lower Bounds via the Probabilistic Method. Lastly, to prove Informal Theorem 2.4, which shows a property that requires $\Omega(n)$ queries even from an MAP that has access to a proof of length $n/100$, we use a technique that is inspired by [GGR98], and also uses ideas from [RVW13]. In more detail, we note that MAPs can be represented by a relatively small class of functions. Since this class of functions is small, using the probabilistic method, we argue that a "random property" (chosen from an adequate distribution) fools every MAP verifier in the sense that the verifier cannot distinguish between a random input that has the property and a totally random input (which will be far from the property).

2.1.5 Related Works

The notion of interactive proofs of proximity was first considered by Ergün, Kumar and Rubinfeld [EKR04] (where it was called approximate interactive proofs). More recently, Rothblum, Vadhan and Wigderson [RVW13] initiated a systematic study of the power of this notion. Their main result is that all languages in NC have interactive proofs of proximity with query and communication complexities roughly $\sqrt{n}$, and $\mathsf{polylog}(n)$ communication rounds. On the negative side, [RVW13] show that there exists a language in NC^1 for which the sum of queries and communication in any constant-round interactive proof of proximity must be polynomially related to n.

The study of interactive proofs-systems (in the polynomial-time setting), of which the class MA is a special case, was initiated in the seminal works of Goldwasser, Micali and

Rackoff [GMR89] and Babai [Bab85]. In the last decade, MA proof-systems were introduced for various computational models. There is a rich body of work in the literature addressing MA communication complexity protocols (e.g., [Kla03, GS10a, Kla11, She12]). Aaronson and Wigderson [AW09] used MA communication complexity lower bounds to show that, for many fundamental questions in complexity theory, any solution will require "non-algebraizing" techniques. In addition, in a recent line of research, the data stream model was extended to support several interactive and non-interactive proof systems. The model of streaming algorithms with non-interactive proofs was first introduced in [CCM09] and extended in [CMT13, GR13b, CCGT13]. Moreover, Cormode *et al.* [CMT12] have made a significant step toward a practical implementation of the generic interactive proof-system of Goldwasser *et al.* [GKR08] for delegation of data stream computation.

Relation to Partial Testing [FGL14]. Independently of this work, Fischer, Goldhirsh and Lachish [FGL14] introduced the notion of *partial testing*, which is closely related to MAPs. A property Π is a said to be Π'-partially testable, for $\Pi' \subseteq \Pi$, if inputs in Π' can be distinguished from inputs that are far from Π by a tester that makes only few queries. As pointed out by [FGL14], an $\mathsf{MAP}(p,q)$ for a property Π is equivalent to the existence of sub-properties $\Pi_1, \ldots, \Pi_{2^p} \subseteq \Pi$ such that $\cup_{i \in [2^p]} \Pi_i = \Pi$ and for every $i \in [2^p]$, the property Π is Π_i-partially testable using q queries.

In our terminology, the main result of [FGL14] is that there exists a (natural) property Π such that every $\mathsf{MAP}(p,q)$ for Π must satisfy that $p \cdot q = \Omega(n)$. In contrast, Informal Theorem 2.2 shows a different property Π' for which $p \cdot q = \Omega(n^{0.999})$. However, we also show an (almost) matching *upper* bound for our property Π' (see Informal Theorem 2.2). We also note that Informal Theorem 2.4 (see Theorem 2.22), which was discovered following the publication of [FGL14], shows a property for which every $\mathsf{MAP}(p,q)$ must satisfy $p + q = \Omega(n)$; that is, if $p = n/100$, then $q = \Omega(n)$. We note that the latter result also resolves (a natural interpretation of) a question asked by [FGL14, Open Question 1.4].[6]

Applications of our Work and Follow-Up Works. Our work has also found applications in unrelated studies. For example, in the study of *sample-based testers*, Goldreich and Ron [GR15a] used the separation between the power of MAPs and property testers (see Theorem 2.7) in order to show that proximity-oblivious testers do not necessarily imply *fair* proximity-oblivious testers (where fair proximity-oblivious testers are such in which every query is almost uniformly distributed). Another example is an application for *testing dynamic environments*. Specifically, the separation between the power of standard MAPs and MAPs with *proof-oblivious queries* (see Lemma 2.6 and Theorem 2.19) was used to show that time-conforming testers can be exponentially weaker than their non-time-conforming counterparts (see [GR14] for details). In addition, following the publication of this work, Goldreich, Gur, and Komargodski [GGK14] improved on Informal

[6] Loosely speaking, in the terminology of [FGL14], Theorem 2.22 implies that for every r there exists a property Π that can be tested with r queries, but every partition of Π into k properties $\Pi_1, \ldots, \Pi_k$, such that Π is P_i-partially testable with $O(1)$ queries, must satisfy that $k = 2^{\Omega(r)}$.

Theorem 2.1 by tightening the separation between MAPs and testers (see Section 2.3.1 for more details).

Non-Deterministic Testing of Graphs Last, we note that Alon *et al.* [AFNS06] discussed the notion of *non-deterministic property testing of graphs*, which was formally stated recently by Lovász and Vesztergombi [LV12], and further studied by Gishboliner and Shapira *et al.* [GS13]. This model is a form of PCP of proximity in which both the proof and verification procedure are restricted to be of a particular form.

2.1.6 Organization

This paper's organization differs from the order in which our results were reviewed in the introduction, so that technically related results are grouped together. In Section 2.2 we formally define MAPs and property testers (which are essentially MAPs with an empty string). In Section 2.3 we formally state and prove all of our separation results, whereas in Section 2.4 we prove our general transformation results. In Section 2.5 we show a property that is hard for MAPs even given a (relatively) long proof. In Section 2.6 we consider MAPs for concatenation problems and in Section 2.7 we show our MAP for verifying bipartiteness of rapidly-mixing graphs in the bounded degree model. Important background material is provided in Section 2.8.1.

2.2 Definitions

In this section we formally define Merlin-Arthur proofs of proximity. We start by introducing some relevant notations and standard definitions.

A property may be defined as a set of strings. However, since we mostly consider properties that consist of (non-Boolean) functions, it will be useful for us to use the following (also commonly used) equivalent definition.

For every $n \in \mathbb{N}$, let D_n and R_n be sets. For simplicity we use the convention that $D_n = [n]$ (and R_n will usually be of size much smaller than n). Let $\mathcal{F}_n$ be the set of all functions from D_n to R_n. A property is an ensemble $\Pi = \cup_{n\in\mathbb{N}} \Pi_n$, where $\Pi_n \subseteq \mathcal{F}_n$. In the (rare) case that we test properties of strings (rather than functions), we view the n-bit string x as a function $I_x : [n] \to \{0,1\}$ where $I_x(i) = x_i$ for all $i \in [n]$. For the rest of this work, it will sometimes be convenient for us to refer to Π as a problem (rather than a property), where we actually refer to the testing problems that are associated with Π (and are defined in the following subsections).

Let $x, y \in \Sigma^n$ be two strings of length $n \in \mathbb{N}$ over a (finite) alphabet Σ. We define the (absolute) distance of x and y as $\Delta(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|$. If $\Delta(x,y) \leq \varepsilon \cdot n$, then we say that x is ε-close to y, and otherwise we say that x is ε-far from y. We define the distance of x from a set $S \subseteq \Sigma^n$ as $\Delta(x,S) \stackrel{\text{def}}{=} \min_{y\in S} \Delta(x,y)$. If $\Delta(x,S) \leq \varepsilon \cdot n$, then we say that x is ε-close to S and otherwise we say that x is ε-far from S. We extend these definitions from strings to functions, while identifying a function with its truth table.

Notation. For a finite set S, we denote by $x \in_R S$ a random variable x that is uniformly distributed in S. We denote by $A^f(x)$ the output of algorithm A given an explicit input x and implicit (i.e., oracle) access to the function f. Last, given a binary string s, we denote its Hamming weight by $\mathsf{wt}(x)$.

Integrality Issues. Throughout this work, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the nearest integer.

2.2.1 Merlin-Arthur Proofs of Proximity

We are now ready to define Merlin-Arthur proofs of proximity.

Definition 2.1. *A* Merlin-Arthur proof of proximity (MAP) *for a property $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ consists of a probabilistic algorithm V, called the verifier, that is given as explicit inputs an integer $n \in \mathbb{N}$, a proximity parameter $\varepsilon > 0$, and a proof string $w \in \{0,1\}^*$; in addition, it is given oracle access to a function $f \in \mathcal{F}_n$. The verifier satisfies the following two conditions:*

1. Completeness: *For every $n \in \mathbb{N}$ and $f \in \Pi_n$, there exists a string w (referred to as a* proof *or* witness*) such that for every proximity parameter $\varepsilon > 0$:*

$$\Pr\left[V^f(n,\varepsilon,w) = 1\right] \geq 2/3.$$

   where the probability is over the random coin tosses of the verifier V.

2. Soundness: *For every $n \in \mathbb{N}$, function $f \in F_n$, string w, and proximity parameter $\varepsilon > 0$, if f is ε-far from Π_n, then:*

$$\Pr\left[V^f(n,\varepsilon,w) = 1\right] \leq 1/3.$$

   where the probability is over the random coin tosses of the verifier V.

If the completeness condition holds with probability 1, then we say that the MAP *has a* one-sided error *and otherwise we say that it has* two-sided error.

We note that MAPs can be viewed as a restricted form of the *interactive* proofs of proximity, studied by [RVW13] (see Section 2.2.2 for the definition of IPP).

An MAP is said to have query complexity $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}$, $\varepsilon > 0$, $f \in \mathcal{F}_n$ and any $w \in \{0,1\}^*$, the verifier makes at most $q(n,\varepsilon)$ queries to f. The MAP is said to have proof complexity $p : \mathbb{N} \to \mathbb{N}$ if for every $n \in \mathbb{N}$ and $f \in \Pi_n$ there exists $w \in \{0,1\}^{p(n)}$ for which the completeness condition holds.[7] If the MAP has query complexity q and proof complexity p, we say that it has complexity $t(n,\varepsilon) \stackrel{\text{def}}{=} q(n,\varepsilon)+p(n)$.

[7] Without loss of generality, using adequate padding, we assume that there is a fixed proof length $p(n)$ for objects of size n. The latter can be complemented by restricting the soundness condition to hold only for strings of length $p(n)$ (rather than strings of arbitrary length), since the verifier can immediately reject proofs that have length that is not $p(n)$.

For every pair of functions $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ and $p : \mathbb{N} \to \mathbb{N}$, we denote by $\mathsf{MAP}_2(p, q)$ (resp., $\mathsf{MAP}_1(p, q)$) the complexity class of all properties that have an MAP with proof complexity $O(p)$, query complexity $O(q)$ and two-sided error (resp., one-sided error). We also use MAP as a shorthand for the class MAP_2.

Note that we defined MAPs such that the proofs do not depend on the proximity parameter ε. Since our focus is on demonstrating the power of MAPs (and our lower bounds refer to fixed valued of the proximity parameter), this makes our results stronger. Nevertheless, see Section 2.2.1 for a discussion of the alternate notion, in which the proof *may* depend on the proximity parameter.

Proof oblivious queries. An aspect of MAP proof-systems, which turns out to be very important, is whether the queries that the verifier makes depend on the proof. An MAP in which the queries *do not depend on the proof* may be thought of as the following two step process:

1. The verifier is given oracle access to the object being tested. The verifier's queries may be adaptively generated (based on answers to previous queries).

2. After getting answers to all of its queries, the verifier is given explicit and free access to the proof string (which is chosen obliviously of the verifier's queries). Based on the queries, answers and the proof, the verifier decides whether to accept or reject.

The foregoing discussion gives rise to the following definition.

Definition 2.2. *An* MAP *verifier for a property* $\Pi \subseteq \{F_n\}_n$ *is said to make* proof oblivious queries *if for every* $n \in \mathbb{N}$*, function* $f \in F_n$*, proximity parameter* $\varepsilon > 0$*, random string* r *and two proof string* $w, w' \in \{0,1\}^*$*, the* MAP *verifier, given oracle access to* f*, the random string* r *and explicit access to* n, ε*, and given either the proof string* w *or* w'*, makes the same sequence of queries.*

MA proximity-oblivious testing. We also present an MA version of *proximity-oblivious testing* (defined in [GR11]). Loosely speaking, a proximity-oblivious tester (POT) is a testing algorithm that satisfies the following conditions: (1) it is oblivious of the proximity parameter ε (i.e., it does not get ε as part of its input) and (2) it rejects statements that are ε-far from true statements with probability that is some increasing function of ε. A standard property tester can be obtained by repeating the POT sufficiently many times.

We give a definition of *one-sided error* MA proximity-oblivious testers, and note that a *two-sided error* variant of MA proximity-oblivious testers can be defined similarly to [GS12].

Definition 2.3. *Let* $\rho : (0,1] \to (0,1]$ *be some increasing function. A (one-sided error)* MA proximity-oblivious tester *for a property* $\Pi = \cup_{i \in \mathbb{N}} \Pi_n$ *with* detection probability ρ *consists of a probabilistic verifier* V *that is given as explicit inputs an integer* $n \in \mathbb{N}$ *and a proof string* $w \in \{0,1\}^*$*, and is given oracle access to a function* $f \in \mathcal{F}_n$*. The verifier satisfies the following two conditions:*

1. Completeness: *For every $n \in \mathbb{N}$ and $f \in \Pi_n$, there exists a proof w such that:*
$$\Pr\left[V^f(n,w) = 1\right] = 1.$$
2. Soundness: *For every $n \in \mathbb{N}$, function $f \in F_n$, and proof w, if f is ε-far from Π_n, then:*
$$\Pr\left[V^f(n,w) = 0\right] \geq \rho(\varepsilon).$$

(In both conditions the probability is over the random coin tosses of the verifier V.)

We remark that a few of the MAPs presented in this work are based on corresponding MA proximity-oblivious testers. The most notable example is the MAP in Theorem 2.9.

MAPs with Proximity-Dependent Proofs We defined the notion of MAPs such that the proof of proximity is *oblivious* of the proximity parameter ε. However, it is also natural to consider a relaxation of MAPs wherein the proof of proximity *may* depend on the proximity parameter. In fact, one can consider two levels of relaxation: (1) the content of the proof *but not its length* may depend on the proximity parameter, and (2) both the contents and the length of the proof may depend on the proximity parameter. We note that the first possibility is almost equivalent to the standard definition of MAP, since it always suffices to refer to only a logarithmic number of values of ε (i.e., $\varepsilon = 2^i$ for all $i \in [\log n]$), and concatenate the proofs for these values, thus obtaining a standard MAP with only a logarithmic overhead to the proof complexity.

Property Testing The standard definition of property testing may be derived from Definition 2.1 by restricting both the completeness and soundness conditions to hold when the proof length is fixed to 0. Hence, MAPs are a strict syntactic generalization of property testers. We will always refer to a tester that uses a proof as an "MAP verifier" and reserve "tester" solely for (standard) property testers that *do not use a proof*.

For a property Π and a proximity parameter $\varepsilon > 0$, we denote by $\mathsf{PT}_\varepsilon(\Pi)$ the minimum, over all testers T for Π, of the query complexity of T with respect to proximity ε. For every function $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$, we denote by $\mathsf{PT}_2(q)$ (resp., $\mathsf{PT}_1(q)$) the class $\mathsf{MAP}_2(0,q)$ (resp., $\mathsf{MAP}_1(0,q)$). We also use PT as a shorthand for the class PT_2.

For a detailed introduction to property testing, see the surveys [Ron08, Ron09] and the collection [Gol10b].

2.2.2 Interactive Proofs of Proximity

In this section we define *interactive proofs of proximity*, following Rothblum *et al.* [RVW13].[8] For two interactive algorithms A and B, we denote by $(A^f, B^f)(x)$ the output of (say) A when interacting with B when both algorithms are given x as an explicit input and implicit (i.e., oracle) access to the function f.

[8] Our definition of IPP slightly differs from that of [RVW13] in that they consider the absolute distance of objects from the property rather relative distance. (Needless to say, we take this into account when discussing their results.)

Definition 2.4. *An* interactive proof of proximity *system (*IPP*) for a property* Π *is an interactive protocol with two parties: a (computationally unbounded) prover* $\mathcal{P}$ *and a verifier* $\mathcal{V}$*, which is a probabilistic algorithm. The parties send messages to each other, and at the end of the communication, the following two conditions are satisfied:*

1. Completeness: *For every* $\varepsilon > 0$*,* $n \in \mathbb{N}$*, and* $f \in \Pi_n$ *it holds that,*

$$\Pr\left[(\mathcal{V}^f, \mathcal{P}^f)(n, \varepsilon) = 1\right] \geq 2/3.$$

   where the probability is over the coin tosses of $\mathcal{V}$*.*

2. Soundness: *For every* $\varepsilon > 0$*,* $n \in \mathbb{N}$*,* $f \in \mathcal{F}_n$ *that is* ε*-far from* Π_n *and for every computationally unbounded (cheating) prover* $\mathcal{P}^*$ *it holds that*

$$\Pr\left[(\mathcal{V}^f, \mathcal{P}^*)(n, \varepsilon) = 1\right] \leq 1/3.$$

   where the probability is over the coin tosses of $\mathcal{V}$*.*

If the completeness condition holds with probability 1, then we say that the IPP *has a* one-sided error *and otherwise the* IPP *is said to have a* two-sided error*.*

An IPP is said to have query complexity $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}$, $\varepsilon > 0$, $f \in \mathcal{F}_n$ and any prover strategy $\mathcal{P}^*$, the verifier makes at most $q(n, \varepsilon)$ queries to f when interacting with $\mathcal{P}^*$. The IPP is said to have communication complexity $c : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}$, $\varepsilon > 0$ and $f \in \Pi_n$ the communication between $\mathcal{V}$ and $\mathcal{P}$ consists of at most $c(n, \varepsilon)$ bits. If the IPP has query complexity q and communication complexity c, we say that it has IPP complexity $q + c$.

For every pair of functions $c, q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$, we denote by $\mathsf{IPP}_2(c, q)$ (resp., $\mathsf{IPP}_1(c, q)$) the complexity class of all properties that have an IPP with communication complexity $O(c)$, query complexity $O(q)$ and two-sided error (resp., one-sided error). We also use IPP as a shorthand for the class IPP_2.

An important parameter of an IPP is the number of messages m sent between the two parties. We denote by $\mathsf{IPP}[m](c, q)$ the set of properties that have m-message IPP protocols in which the verifier uses at most $O(c)$ bits of communication, and makes at most $O(q)$ oracles queries.

2.2.3 Useful Conventions

The proximity parameter. We view the proximity parameter as a function $\varepsilon = \varepsilon(n)$. For simplicity we assume that $\varepsilon(n)$ is a non-increasing function.

Our definition of MAPs requires that soundness hold with respect to *every* value of $\varepsilon > 0$. However, throughout this work we sometimes find it convenient to restrict the proximity to $\varepsilon \in (0, \varepsilon_0)$ for some constant $\varepsilon_0 \in (0, 1)$. We note that latter type of MAPs can be extended to the more general form by simply running the base tester with respect to proximity $\varepsilon' = \min(\varepsilon, \varepsilon_0)$ (incurring only a constant overhead).

Implicit input length and proximity parameter. Throughout this work, for simplicity of notation, we use the convention that the input length n and proximity parameter ε are given *implicitly* to all testers and verifiers (e.g., when we write T^f we actually mean $T^f(n, \varepsilon)$).

2.3 Separation Results

In this section we explore the power of MAP verifiers in comparison to other types of testers, such as property testers and IPP verifiers and present properties that exhibit a separation between these different types of testers.

In Section 2.3.1 we show an exponential gap between the complexity of PT and MAP. In Section 2.3.2 we show a problem that has an MAP with an (almost) tight multiplicative tradeoff between the proof length and number of queries. In Section 2.3.3 we consider 3-*message* IPP verifiers and show that they may have exponentially smaller *query* complexity than MAP verifiers (when using a proof of similar length). Finally, in Section 2.3.4 we also show an exponential gap between the total complexity (i.e., query plus proof/communication complexities) of MAP and general IPP (which uses a poly-logarithmic number of messages).

2.3.1 Exponential Separation between PT and MAP

In this section we show an *exponential* separation between the power of property testing and MAP. Roughly speaking, we show a property that requires roughly $n^{0.999}$ queries for every property tester but has an MAP that, while using a proof of only *logarithmic* length, requires only a *constant* number of queries. We prove the following incomparable variants of this result.

Theorem 2.7. *For every constant $\alpha > 0$, there exists a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(1/\varepsilon)$ queries for every $\varepsilon > 1/\mathsf{polylog}(n)$, but for which every* property tester *must make $\Omega(n^{1-\alpha})$ queries. Furthermore, the* MAP *has one-sided error.*

A limitation of the foregoing theorem is that the proximity parameter is required to be larger than $1/\mathsf{polylog}(n)$. We also consider two incomparable variants of Theorem 2.7 that let us handle general values of ε. In Theorem 2.8 we do so but at the cost of increasing the MAP query complexity to depend poly-logarithmically on n.

Theorem 2.8. *For every constant $\alpha > 0$, there exists a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(\log n, 1/\varepsilon)$ queries, but for which every* property tester *must make $\Omega(n^{1-\alpha})$ queries. Furthermore, the* MAP *has one-sided error.*

The above separation results refer to the general (i.e., two-sided error) classes PT_2 and MAP_2. As noted in the introduction, a more restricted separation between the *one-sided error* classes (i.e., between PT_1 and MAP_1) can be obtained by using Theorem 2.20. We

remark that the preliminary technical report [GR13c] also contained a proof of the following (incomparable) variant, which can handle all values of the proximity parameter while using $\mathsf{poly}(1/\varepsilon)$ query complexity, at the cost of having a smaller (yet still exponential) gap between the power of property testers and MAPs.

Theorem 2.9 ([GR13c])**.** *There exists a universal constant $c \in (0,1)$ and a property Π that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(1/\varepsilon)$ queries (without limitation on ε), but for which every* property tester *must make n^c queries. Furthermore, the* MAP *has one-sided error.*[9]

A different proof of Theorem 2.9 is sketched in [FGL14] who, using a result of Alon *et al.* [AKNS00], showed a property that requires $\Omega(\sqrt{n})$ queries (without a proof) but can be tested using only $O(1/\varepsilon)$ queries and a proof of length $O(\log n)$.

Follow-Up Work. Following the publication of this work, Goldreich, Gur, and Komargodski [GGK14] improved the separation between MAPs and testers, achieving the best of Theorems 2.7 to 2.9 simultaneously; that is, they obtain a separation for all values of the proximity parameter, with constant query complexity for the MAPs, and nearly-linear query complexity for testers.

Theorem 2.10 ([GGK14])**.** *For every constant $\alpha > 0$, there a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(1/\varepsilon)$ queries (without limitation on ε), but for which every* property tester *must make $n^{1-\alpha}$ queries. Furthermore, the* MAP *has one-sided error.*

In the next subsections we will show two lemmas (Lemmas 2.5 and 2.6) that allow us to reduce the problem of separating the power of MAPs and testers to the problem of designing error-correcting codes that are both locally testable and locally decodable. Theorems 2.7 to 2.10 are then obtained by instantiating Lemmas 2.5 and 2.6 with such codes. Since the codes of [GGK14] improve upon the codes that are used to obtain Theorems 2.7 to 2.9, we omit the more involved proof of Theorem 2.9, which consists of a construction of a code with the desired properties (see technical report [GR13c] for details and proof). We provide the proofs of Theorems 2.7 and 2.8, which are instantiations of Lemmas 2.5 and 2.6 for known codes.

2.3.1.1 Our Approach

The proof of Theorem 2.7 is heavily based on error correcting codes. Recall that a code is an injective function $C : \Sigma^k \to \Sigma^n$ over an alphabet Σ. The relative distance of the code is the minimal relative distance between every two (distinct) codewords, and the stretch of the code is n when viewed as a function of k. Further necessary background is provided in Section 2.8.1.3.

[9] We remark that the proof of Theorem 2.9 can be adapted to yield an MA *proximity-oblivious tester* (see Definition 2.3) for Π.

As discussed in the introduction, the complexities of property testers and MAP verifiers with *proof oblivious queries* are polynomially related (see Theorem 2.19). Thus, in order to show an *exponential* separation between PT and MAP, one has to use an MAP for which the queries inherently depend on the proof. That is, the property Π should satisfy the following:

1. Π can be efficiently verified by an MAP in which the queries are "strongly affected" by the proof;
2. Π is hard for property testers (and hence for MAPs with proof oblivious queries).

Thus, intuitively, we seek a property that is based on a "hidden structure" that can be tested locally if one knows where to look but cannot be tested locally otherwise.

As a first (naive) candidate, consider the property containing the set of all non-zero strings. A short proof for this property could direct us to the exact location of a non-zero bit, which can then be verified by a single query. However, the aforementioned property is (almost) trivial — as all strings are close to a string with a non-zero bit. Hence, we seek a robust version of this property.

This naturally leads us to consider an encoded version of the foregoing naive property. Fix an error-correcting code C and consider the property that contains all codewords that encode non-zero strings. Assuming that the code is both locally testable and locally decodable (i.e., both an LTC and an LDC, see Section 2.8.1.3), it is easy to test this property using an MAP that simply specifies a non-zero coordinate of the encoded message. However, this property may also be easy to test without a proof since all one needs to do is test that the string is not the (single) encoding of the zero message but is (close to) a codeword.

To overcome this difficulty, we consider a "twist" of the foregoing property in which we consider two codewords that must be non-zero on the same coordinate. That is, for every code C, we define the encoded intersecting messages property, denoted by EIM_C as:

$$\mathsf{EIM}_C \stackrel{\text{def}}{=} \left\{ \big(C(x), C(y)\big) \;:\; x, y \in \Sigma^k,\ k \in \mathbb{N} \text{ and } \exists i \in [k] \text{ s.t. } x_i \neq 0 \text{ and } y_i \neq 0 \right\},$$

where we assume that $0 \in \Sigma$. We note that we could have slightly modified our definition by requiring that $x_i = y_i = 1$ (where the choice of 1 is arbitrary) rather than $x_i, y_i \neq 0$. Another notable variant is obtained by requiring that $\Sigma = \{0, 1\}$; then the property EIM_C contains all pairs of codewords whose corresponding encoded messages (viewed as sets) intersect (i.e., are not disjoint).

For the lower bound, we only require that C have constant relative distance and the quality of the lower bound is directly related to the stretch of the code. For the upper bound, in addition to the constant relative distance, we need C to be both an LTC and an LDC with small query complexities. Indeed, the query complexity of the MAP that we construct is proportional to the number of queries required by the LTC and LDC procedures.

It is well-known that (a suitable instantiation of) the Reed-Muller code is both an LTC and LDC with $\mathsf{polylog}(n)$ query complexities, and almost linear stretch. By instantiating

EIM with this code, we can obtain Theorem 2.8; namely, a property that has an MAP with a proof of length $O(\log n)$ and $\mathsf{polylog}(n)$ query complexity, but requires an almost linear number of queries by any (standard) property tester.

In order to obtain a result with *constant* MAP query complexity (as in Theorem 2.7), we need a code that is both an LTC and an LDC, with constant query complexities. While LTCs with constant query complexity (and almost linear stretch) are known, constructing LDCs with constant query complexity (and polynomial stretch) is a major open problem in the theory of computation. However, we observe that for our construction it actually suffices that C be a *relaxed*-LDC. Relaxed-LDCs, introduced by Ben-Sasson *et al.* [BSGH+06], are a weaker form of LDCs in which the decoder is allowed to output a special abort symbol $\perp$ in case it is unable to decode a corrupt codeword. However, the decoder is not allowed to abort when given as input a correct codeword. We refer the reader to Definition 2.27 for the formal definition.

Ben-Sasson *et al.* [BSGH+06] used PCPPs to construct an $O(1)$-relaxed-LDC with almost linear stretch. Furthermore, [BSGH+06] argue that their relaxed-LDC is also a $\mathsf{poly}(1/\varepsilon)$-LTC. However, the LTC property only holds for proximity parameter $\varepsilon > 1/\mathsf{polylog}(n)$. Thus, using the [BSGH+06] code, we (only) obtain Theorem 2.7. In addition, by combining ideas and results of [BSGH+06] and [GS06] we construct an $O(1)$-relaxed-LDC that is also a $\mathsf{poly}(1/\varepsilon)$-LTC *for general values of* $\varepsilon > 0$, albeit with polynomial (rather than almost linear) stretch. Using the latter result, which may be of independent interest, we obtain Theorem 2.9.

Organization. In Section 2.3.1.2 we show that for every code $C : \Sigma^k \to \Sigma^n$ that is a t_1-relaxed-LDC and a t_2-LTC, it holds that $\mathsf{EIM}_C \in \mathsf{MAP}\big(\log k, t_1(n/2) + t_2(n/2, \varepsilon/2)\big)$. In Section 2.3.1.3 we show an $\Omega(k/\log|\Sigma|)$ lower bound on the query complexity of testing EIM_C (without a proof of proximity). In Section 2.3.1.4 we state the result of [BSGH+06] and derive Theorem 2.7, and in Section 2.3.1.5 we prove Theorem 2.8 using an appropriate instantiation of the Reed-Muller code.

2.3.1.2 An MAP Upper Bound for EIM

Lemma 2.5. *Let $C : \Sigma^k \to \Sigma^n$ be a code with constant relative distance that is a t_1-relaxed-LDC and also a t_2-LTC. Then, $\mathsf{EIM}_C \in \mathsf{MAP}_1\big(\log k, t_1(n/2) + t_2(n/2, \varepsilon/2)\big)$.*

Proof. We prove Lemma 2.5 by showing an MAP proof-system for proving proximity to EIM_C. The proof of proximity for the statement $(C(x), C(y)) \in \mathsf{EIM}_C$ is simply a coordinate $i \in [k]$ such that the messages x and y are non-zero i (i.e., $x_i, y_i \neq 0$). Given the proof i and oracle access to a pair of strings (α, β), it suffices for the verifier to check that both α and β are close to codewords (using the LTC property) and if so to reconstruct the i^{th} symbol of the underlying messages (using the relaxed-LDC property). (Lastly, it verifies that both symbols are non zero.)

The full protocol is described in Fig. 2.1, where $\delta_0 \in (0, 1)$ denotes the relative distance of C, and $\delta \in (0, \delta_0/2)$ denotes the decoding radius of C (i.e., strings that are δ-close to codewords are correctly decoded by the relaxed-LDC procedure).

MAP for EIM_C (where $C : \Sigma^k \to \Sigma^n$ is a t_1-relaxed-LDC and a t_2-LTC)

Input: a proximity parameter $\varepsilon \in (0, 2\delta)$ (where δ is the decoding radius) and oracle access to a pair $(\alpha, \beta) \in \Sigma^{n+n}$.

The Proof:

- Let $x, y \in \Sigma^k$ be the unique messages encoded in α and β, respectively; that is, $C(x) = \alpha$ and $C(y) = \beta$. Denote the i^{th} symbol of x by x_i, and the i^{th} symbol of y by y_i.
- The proof consists of a coordinate $i \in [k]$ such that $x_i \neq 0$ and $y_i \neq 0$ (which exists, for $(\alpha, \beta) \in \mathsf{EIM}_C$).

The Verifier:

1. Run the local *testing* algorithm of C on α and on β with respect to proximity parameter $\varepsilon/2$ and reject if either test rejects.
2. Run the (relaxed) local *decoding* algorithm of C to obtain the i^{th} message symbol of α, denoted σ, and the i^{th} message symbol of β, denoted τ.
3. Accept if both $\sigma \neq 0$ and $\tau \neq 0$, and reject otherwise.

Figure 2.1: MAP for EIM_C

Since the code is a t_1-relaxed-LDC and a t_2-LTC, the query complexity of the MAP is $2t_1(n/2) + 2t_2(n/2, \varepsilon/2)$, and the proof complexity is $\log_2 k$. We proceed to show that both completeness and soundness hold.

Completeness. If $(\alpha, \beta) \in \mathsf{EIM}_c$, then there exist $x, y \in \Sigma^k$ such that $\alpha = C(x)$ and $\beta = C(y)$, and therefore the local testing algorithm succeeds. Since the proof consists of a coordinate i for which $x_i, y_i \neq 0$, and the local decoding algorithm always succeeds, the MAP verifier always accepts.

Soundness. Suppose that (α, β) is ε-far from EIM_C and let $i \in [k]$ be some alleged proof to the false statement $(\alpha, \beta) \in \mathsf{EIM}_C$. There are two possible scenarios to consider:

1. either α or β are $\varepsilon/2$-far from C; or
2. both α and β are $\varepsilon/2$-close to C.

In the first case, with probability at least $1/2$, the local testing algorithm will fail and therefore the MAP verifier rejects with probability at least $1/2$. We proceed to the second case.

Suppose that both α and β are $\varepsilon/2$-close to the code. Then, there exist unique $x, y \in \Sigma^k$ s.t. α is $\varepsilon/2$-close to $C(x)$ and β is $\varepsilon/2$-close to $C(y)$, where uniqueness holds

since $\varepsilon/2 < \delta < \delta_0/2$. However, since (α, β) is ε-far from having the property EIM_C, this implies that either $x_i = 0$ or $y_i = 0$ (where i is the alleged proof). Thus, when running the relaxed local decoding algorithm (since $\varepsilon/2 < \delta$), with probability at least $2/3$, the decoder will output either 0 or $\perp$ on one of the two codewords (with respect to coordinate i), in which case the verifier rejects. We conclude that in both scenarios the verifier rejects with probability at least $1/2$. □

2.3.1.3 A PT Lower Bound for EIM

Next, we show a that the query complexity of property testing the EIM property must be linear in k.

Lemma 2.6. *Let $C : \Sigma^k \to \Sigma^n$ be an error-correcting code with relative distance at least $\delta_0 \in (0, 1)$. Then, for any $\varepsilon \in (0, \delta_0/2)$ it holds that:*

$$\mathsf{PT}_\varepsilon\big(\mathsf{EIM}_C\big) = \Omega(k/\log|\Sigma|)$$

The proof of Lemma 2.6 uses the framework of [BBM11] for showing property testing lower bounds via communication complexity lower bounds. The necessary background on communication complexity is provided in Section 2.8.1.1 (for a comprehensive introduction to communication complexity, see [KN97]).

The basic approach of [BBM11] is to reduce a hard communication complexity problem to the property testing problem for which we want to show a lower bound. We follow [BBM11] by showing a reduction from the well-known communication complexity problem of *set-disjointness*. The aforementioned framework allows us to obtain a lower bound on the query complexity of testing the *encoded intersecting messages* property.

For sake of self containment, we state the relevant definitions and lemmas that we need from [BBM11].

Definition 2.7 (Combining operators)**.** *A* combining operator *is an operator ψ that takes as input two functions $f, g : D \to R$ (where D and R are some finite sets) and returns a function $h_{f,g}$. We denote by $|\psi| \stackrel{\text{def}}{=} \log_2|R|$. The combining operator is called* simple *if $h_{f,g}(x)$ can be computed from $x, f(x)$ and $g(x)$ (i.e., without requiring access to f and g).*

Let Π be a property, and let ψ be a combining operator. For every integer $n \in \mathbb{N}$ and proximity parameter $\varepsilon > 0$, we denote by $\mathcal{C}^{\Pi}_{\psi,\varepsilon}$ the communication complexity problem wherein Alice gets a function f, and Bob gets a function g,[10] and their goal is to decide whether $\psi(f, g) \in \Pi$ or $\psi(f, g)$ is ε-far from Π.[11] Next, we state the main lemma from [BBM11].

[10] More formally, the parties get as input strings that represent the truth table of the functions.

[11] Due to the symmetrical definition of the communication complexity model, it is unimportant which of these cases (i.e., $\psi \in \Pi$ or ψ that is ε-far from Π) is viewed as a YES-instance of Π. In contrast, see Footnote 13.

Lemma 2.8. *For any* simple *combining operator* ψ*, any property* Π *and any proximity parameter* $\varepsilon > 0$*, we have that:*

$$\mathsf{PT}_{\varepsilon}(\Pi) \geq \frac{\mathsf{CC}(\mathcal{C}^{\Pi}_{\psi,\varepsilon})}{2|\psi|}$$

where $\mathsf{PT}_{\varepsilon}(\Pi)$ *refers to the query complexity of the property* Π *with respect to proximity* ε *and* $\mathsf{CC}(\mathcal{C})$ *refers to the communication complexity of* $\mathcal{C}$ *(see Section 2.8.1.1).*

Recall that the set-disjointness problem is the communication complexity problem wherein Alice gets an n-bit string x, Bob gets an n-bit string y, and their goal is to decide whether there exists $i \in [n]$ such that $x_i = y_i = 1$. Equivalently, Alice and Bob's inputs can be viewed as indicator vectors of sets $A, B \subseteq [n]$. In this case, the goal of the players is to decide if the sets corresponding to their inputs intersect or not. Following many works in the literature we consider the promise problem (sometimes also called *unique* disjointness) in which the intersection is of size at most 1. That is, the two parties need to distinguish between the case that their intersection is empty, and the case that it is of size exactly 1. We denote the latter problem by DISJ_n.

It is well-known (see Section 2.8.1.1) that the randomized communication complexity of the *set-disjointness* problem is linear in the size of the inputs, even under the promise that A and B intersect in at most one element.

Theorem 2.11 ([KS92])**.** *For every* $n \in \mathbb{N}$*,*

$$\mathsf{CC}(\mathsf{DISJ}_n) = \Omega(n).$$

Using the aforementioned results, we are ready to prove Lemma 2.6.

Proof of Lemma 2.6. Let $C : \Sigma^k \to \Sigma^n$ be an error-correcting code with relative distance $\delta_0 \in (0, 1)$ where we assume without loss of generality that $\{0, 1\} \subseteq \Sigma$. Denote by Pair the operator that takes two strings $x, y \in \Sigma^k$ and returns a function $z : [k] \to \Sigma$ that outputs (x_i, y_i) on input $i \in [k]$. Consider $\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}$, the communication complexity problem wherein Alice gets a string $x \in \Sigma^k$, Bob gets a string $y \in \Sigma^k$, and their goal is to decide whether $(x, y) \in \mathsf{EIM}_C$ or (x, y) is ε-far from EIM_C. Using the results of [BBM11] (see Lemma 2.8) we have,

$$\mathsf{PT}_{\varepsilon}(\mathsf{EIM}_C) \geq \frac{1}{2\log|\Sigma|}\mathsf{CC}\left(\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}\right). \tag{2.1}$$

Since by Theorem 2.11 we have $\mathsf{CC}(\mathsf{DISJ}_k) = \Omega(k)$, then it suffices to show that

$$\mathsf{CC}\left(\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}\right) \geq \mathsf{CC}(\mathsf{DISJ}_k). \tag{2.2}$$

Toward this end, we show a reduction from the communication complexity problem DISJ_k to the communication complexity problem $\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}$. We note that, under the natural association of EIM_C with YES-instances and "far from EIM_C" with NO-instances, our

reduction maps YES (resp., NO) instances of DISJ_k to NO (resp., YES) instances of EIM_C. Let π be a protocol for $\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}$ with communication complexity c. Consider the following protocol for DISJ_k.

Let $x, y \in \{0,1\}^k$ be the inputs of Alice and Bob (respectively) for DISJ_k. Alice computes $\alpha = C(x)$. Bob computes $\beta = C(y)$. The players then run π on (α, β) and return the *negation* of its output.

Indeed, if $(x, y) \in \mathsf{DISJ}_k$ (i.e., their intersection is empty), then for every $i \in [k]$, either $x_i = 0$ or $y_i = 0$. Since the relative distance of C is at least δ_0, it holds that (α, β) is $(\delta_0/2)$-far from EIM_c. On the other hand, if $(x, y) \notin \mathsf{DISJ}_k$ (i.e., their intersection is of size 1), then there exists $i \in [k]$ such that $x_i = y_i = 1$. Hence, $\big(\alpha, \beta\big) \in \mathsf{EIM}_c$. Moreover, note that the total number of bits that were communicated is exactly c.

Using Eq. (2.1) and Eq. (2.2), together with Theorem 2.11, we conclude that for every $\varepsilon > 0$,

$$\mathsf{PT}_\varepsilon(\mathsf{EIM}_c) \geq \frac{1}{2\log|\Sigma|}\mathsf{CC}\left(\mathcal{C}^{\mathsf{EIM}_C}_{\mathsf{Pair},\varepsilon}\right) \geq \frac{1}{2\log|\Sigma|}\mathsf{CC}(\mathsf{DISJ}_k) = \Omega(k).$$

□

2.3.1.4 Proof of Theorem 2.7

In order to obtain an $O(1)$-relaxed-LDC that is also a $\mathsf{poly}(1/\varepsilon)$-LTC, we shall use the following construction of Ben-Sasson *et al.* [BSGH+06].

Theorem 2.12 ([BSGH+06, Remark 4.6])**.** *For every $\alpha > 0$, there exists a binary code that is an $O(1)$-relaxed-LDC and a t-LTC with constant relative distance and stretch $n = k^{1+\alpha}$, where for $\varepsilon > 1/\mathsf{polylog}(n)$ it holds that $t(n, \varepsilon) = \mathsf{poly}\left(\frac{1}{\alpha\varepsilon}\right)$.*

Theorem 2.7 follows by combining Theorem 2.12 with Lemma 2.5 and Lemma 2.6.

2.3.1.5 Proof of Theorem 2.8

In this section we show that a well-known variant of the Reed-Muller error-correcting code is an $\mathsf{polylog}(n)$-LDC (and in particular a $\mathsf{polylog}(n)$-relaxed-LDC) and a $\mathsf{poly}(\log n, 1/\varepsilon)$-LTC. Combining the latter with Lemma 2.5 and Lemma 2.6, we prove Theorem 2.8.

Lemma 2.9. *For every constant $\alpha > 0$, there exists a $\mathsf{polylog}(n)$-LDC that is also a $\mathsf{poly}(\log n, 1/\varepsilon)$-LTC with stretch $n = k^{1+\alpha}$ and relative distance $1 - o(1)$.*

Proof. We construct a code $C : \Sigma^k \to \Sigma^n$ as follows. Fix a finite field $\mathbb{F}$ and an integer m such that $|\mathbb{F}|^m = n$. The alphabet of the code is $\Sigma = \mathbb{F}$. Consider an arbitrary subset $H \subset \mathbb{F}$ of size $k^{1/m}$. We view a message $x \in \mathbb{F}^k$ as a function $x : H^m \to \mathbb{F}$ by identifying H^m and $[k]$ in some canonical way. The encoding $C(x)$ is the low degree extension $\hat{x}$ of x with respect to the field $\mathbb{F}$. Namely, the (unique) m-variate polynomial of individual degree $|H| - 1$ that agrees with x on H^m.

The code stretches $k = |H|^m$ symbols to $n = |\mathbb{F}|^m$ symbols, and by the Schwartz-Zippel Lemma it has relative distance at least $1 - \frac{m|H|}{|\mathbb{F}|}$. Furthermore, the code can be

locally tested using $O(m|H| \cdot \mathsf{poly}(1/\varepsilon))$ queries (see Theorem 2.31), and locally decoded using $O(m|H|)$ queries (see Theorem 2.29). Thus, to obtain our result we need to set our parameters as to maximize the ratio $|H|/|\mathbb{F}|$, while minimizing $m \cdot |H|$ and keeping $|\mathbb{F}| > m \cdot |H|$.

For every constant $\alpha > 0$ and every integer $n \in \mathbb{N}$, we let $\mathbb{F}$ be a finite field of size $(\log n)^{1/\alpha}$, let $m = \alpha \cdot \frac{\log n}{\log\log(n)}$ and let H be some fixed (arbitrary) subset of $\mathbb{F}$ of size $|\mathbb{F}|^{1-\alpha}$. Hence, $\frac{m \cdot |H|}{|\mathbb{F}|} = \alpha \cdot \frac{\log n}{\log\log n} \cdot |\mathbb{F}|^{-\alpha} = o(1)$. The code has relative distance $1 - \frac{(|H|-1)\cdot m}{|\mathbb{F}|} = 1 - o(1)$, stretch $n = |\mathbb{F}|^m = |H|^{m/(1-\alpha)} = k^{1/(1-\alpha)}$. In addition, it can be locally tested using $\mathsf{poly}(\log n, 1/\varepsilon)$ queries, and locally decoded using $\mathsf{polylog}(n)$ queries. □

A natural property. We remark that when the *encoded intersecting messages* property is instantiated with the foregoing variant of the Reed-Muller code, we obtain a *natural* property that consists of pairs (P, Q) of low-degree polynomials, whose product $P \cdot Q$ is non-zero on a given subset of its domain. That is, the property is

$$\Pi_{\mathbb{F},d,m,H} = \left\{ (P,Q) : P, Q : \mathbb{F}^m \to \mathbb{F} \text{ have individual degree } d \text{ and } \sum_{x \in H^m} (P \cdot Q)(x) \neq 0 \right\}.$$

2.3.2 Trade-off between Query and Proof Complexity

In this section we show a property that has a multiplicative trade-off between proof and query complexities for MAP testing. We show a property that can be tested with a nearly smooth tradeoff between the proof and query complexities.

Theorem 2.13. *For every constant $\alpha > 0$, there exists a property Π_α such that for every sublinear function $p : \mathbb{N} \to \mathbb{N}$, the query complexity of Π for MAP verifiers, which use proofs of length p, is upper bounded by $\frac{n^{1-\alpha+o(1)}}{p} \cdot \mathsf{poly}(1/\varepsilon)$ and lower bounded by $\tilde{\Omega}\left(\frac{n^{1-\alpha}}{p}\right)$.*

Our proof is heavily based on multivariate polynomials, and we refer the reader to Section 2.8.1.4 for the necessary background (e.g., the Schwartz-Zippel lemma and low degree testing). In fact, the proof of Theorem 2.13 is based on a specific algebraic property that we call *Sub-Tensor Sum*. We note that this property will also be used in Section 2.3.3 and Section 2.3.4.

We proceed to describe the sub-tensor sum problem. Let $\mathbb{F}$ be a finite field, let $m, d \in \mathbb{N}$ such that $d \cdot m < |\mathbb{F}|/10$ and let $H \subset \mathbb{F}$. Consider the following property.

Definition 2.10. *The* Sub-Tensor Sum *property, denoted $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$, is parameterized by a field $\mathbb{F}$, a dimension $m \in \mathbb{N}$, a degree $d \in \mathbb{N}$ and a subset $H \subset \mathbb{F}$, and contains all polynomials $P : \mathbb{F}^m \to \mathbb{F}$ of individual degree d, such that*

$$\sum_{x \in H^m} P(x) = 0$$

where the arithmetic is over $\mathbb{F}$.

To obtain a tight trade-off, we shall be using some $d = \Theta(|H|)$. To simplify the notation, when the parameters are clear from the context, we shorthand $\mathsf{TensorSum}$ for $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$. Next, we proceed to show the (almost) tight multiplicative trade-off for $\mathsf{TensorSum}$. In Section 2.3.2.1 we prove the upper bound and in Section 2.3.2.2 we prove the lower bound. Finally, in Section 2.3.2.3 we set the parameters for proving Theorem 2.13.

2.3.2.1 MAP Upper Bound for TensorSum

We start by proving the following upper bound.

Lemma 2.11. *If $dm < |\mathbb{F}|/10$, then, for every $\ell \in \{0,\ldots,m\}$, the $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$ property has an MAP with proof complexity $(d+1)^\ell \cdot \log(|F|)$ and query complexity $|H|^{m-\ell} \cdot (dm^2 \log|H|) \cdot \mathsf{poly}(1/\varepsilon)$. Furthermore, the MAP has a one-sided error.*

We note that the additional parameter ℓ essentially controls the proof length (and will be set as roughly the logarithm of the desired proof length). Moreover, d will be set such that $d = \Theta(|H|)$ and therefore $d^\ell \cdot |H|^{m-\ell} \approx |H|^m$ and so we can set ℓ to obtain the desired trade-off between proof and query complexities.

Proof of Lemma 2.11. We prove the lemma by showing an MAP protocol for the statement $P \in \mathsf{TensorSum}$. The main idea is to partition H^m into $|H|^\ell$ sub-tensors of the form $(x_1,\ldots,x_\ell,*,*,\ldots,*)$ for every $x_1,\ldots,x_\ell \in H$, and use a *low degree* ℓ-variate polynomial Q such that $Q(x_1,\ldots,x_\ell)$ equals the sum of the $(x_1,\ldots,x_\ell)^{\text{th}}$ tensor over $H^{m-\ell}$. Specifically, we refer to the polynomial:

$$Q(x_1,\ldots,x_\ell) = \sum_{x_{\ell+1},\ldots,x_m \in H} P(x_1,\ldots,x_m).$$

Thus, the MAP proof for the statement $P \in \mathsf{TensorSum}$, consists of the polynomial Q. The verifier checks that (1) P is (close to) a low degree polynomial, (2) the sum of Q on H^ℓ is 0, and (3) that Q is consistent with P. The last step uses the fact that both Q and P are low degree polynomials and so it suffices to verify consistency of a random point in Q by reading the entire corresponding sub-tensor (i.e., $|H|^{m-\ell}$ points) from P. Actually, since P can only be verified to be *close to* a low degree polynomial, the $|H|^{m-\ell}$ points are read via self-correction. The detailed protocol is presented in Fig. 2.2 (where all arithmetic is over $\mathbb{F}$).

Note that the proof of proximity consists of $|Q| = O((d+1)^\ell \log|\mathbb{F}|)$ bits and that the total number of queries to the oracle is dominated by the $|H|^{m-\ell}$ invocations of the self-correction algorithm (which requires $(m\log(|H|)\cdot dm \cdot \mathsf{poly}(1/\varepsilon)$ queries for each invocation to obtain the desired soundness level). We proceed to show that completeness and soundness hold.

MAP for TensorSum with parameter $\ell \leq m$

Parameters: $\mathbb{F}$ (field), m (dimension), d (individual degree) and $H \subset \mathbb{F}$.

Input: a proximity parameter $\varepsilon \in (0, 1/3)$, and oracle access to a function $P : \mathbb{F}^m \to \mathbb{F}$.

The Proof:

- The proof consists of a multivariate polynomial $\tilde{Q} : \mathbb{F}^\ell \to \mathbb{F}$ of individual degree d (specified by its $(d+1)^\ell$ coefficients), which allegedly equals

$$Q(x_1, \ldots, x_\ell) \stackrel{\text{def}}{=} \sum_{x_{\ell+1}, \ldots, x_m \in H} P(x_1, \ldots, x_m).$$

The Verifier:

1. If $\sum_{x_1, \ldots, x_\ell \in H} \tilde{Q}(x_1, \ldots, x_\ell) \neq 0$, then reject.
2. Run the low individual d-degree test (see Theorem 2.31) on P with respect to the proximity parameter ε. If the test fails, then reject.
3. Select uniformly at random $r_1, \ldots, r_\ell \in_R \mathbb{F}$.
4. For every $x_{\ell+1}, \ldots, x_m \in H$, read the value of $P(r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m)$ using self correction (see Theorem 2.29) repeated $O(m \log(|H|))$ times (to reduce the error probability to $\frac{1}{10|H|^m}$ for each point). Denote the value read by $z_{r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m}$.
5. Accept if $\tilde{Q}(r_1, \ldots, r_\ell) = \sum_{x_{\ell+1}, \ldots, x_m \in H} z_{r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m}$ and otherwise reject.

Figure 2.2: MAP for TensorSum

Completeness. If $P \in$ TensorSum, then $\sum_{x_1, \ldots, x_\ell \in H} Q(x_1, \ldots, x_\ell) = 0$ and P has individual degree d (and so the individual degree test passes). Moreover, in this case $\tilde{Q} = Q$ and

$$Q(r_1, \ldots, r_\ell) = \sum_{x_{\ell+1}, \ldots, x_m \in H} P(r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m).$$

By the zero-error feature of the self-correction procedure, with probability 1,

$$z_{r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m} = P(r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m),$$

and therefore $\sum_{x_{\ell+1}, \ldots, x_m \in H} z_{r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m} = \tilde{Q}(r_1, \ldots, r_\ell)$. Hence, in this case, the verifier accepts with probability 1.

Soundness. Let $\varepsilon > 0$ and let $P : \mathbb{F}^m \to \mathbb{F}$ be a polynomial that is ε-far from TensorSum. Let $\tilde{Q}$ be an alleged proof (to the false statement $P \in$ TensorSum).

Consider first the case that P is ε-far from having individual degree d. In this case, by the individual degree test (Theorem 2.31), the verifier rejects with probability at least $1/2$. Thus, we focus on the case that P is ε-close to a polynomial P' of individual degree d. We may also assume that $\sum_{x_1,\ldots,x_\ell \in H} \tilde{Q}(x_1,\ldots,x_\ell) = 0$ (since otherwise the verifier rejects with probability 1). Define

$$Q'(x_1,\ldots,x_\ell) \stackrel{\text{def}}{=} \sum_{x_{\ell+1},\ldots,x_m \in H} P'(x_1,\ldots,x_m).$$

Clearly $\sum_{x_1,\ldots,x_\ell} Q'(x_1,\ldots,x_\ell) \neq 0$ (since otherwise P is ε-close to $P' \in \mathsf{TensorSum}$). Thus, the individual degree d polynomials Q' and $\tilde{Q}$ differ, and so, by the Schwartz-Zippel Lemma they can agree on at most a $\frac{d\ell}{\mathbb{F}}$ fraction of their domain $\mathbb{F}^\ell$.

To complete the argument note that the self-correction algorithm guarantees that every $z_{r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m}$ is equal to $P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m)$, with probability $1 - \frac{1}{10|H|^m}$ (here we use our assumption that, without loss of generality, $\varepsilon < 1/3$). Therefore, by the union bound, all points are read correctly with probability at least 0.9, and in this case $\sum_{x_{\ell+1},\ldots,x_m \in H} z_{r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m} = Q'(r_1,\ldots,r_\ell)$. Thus, with probability $0.9 \cdot (1 - \frac{dm}{\mathbb{F}}) \geq 2/3$, the verifier rejects when testing that $\tilde{Q}(r_1,\ldots,r_\ell)$ equals $\sum_{x_{\ell+1},\ldots,x_m \in H} z_{r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m}$. □

2.3.2.2 MAP Lower Bound for TensorSum

Next, we give an (almost) matching lower bound on the MAP complexity of *Sub-Tensor Sum*. Formally, we show

Lemma 2.12. *For every $\varepsilon \in (0, 1 - \frac{dm}{|\mathbb{F}|})$, if $d \geq 2(|H|-1)$, then every* MAP *for* TensorSum *(with respect to proximity parameter ε) that has proof complexity $p \geq 1$ must have query complexity $q = \Omega\left(\frac{|H|^m}{p \cdot \log|\mathbb{F}|}\right)$.*

As an immediate corollary of Lemma 2.12 we obtain the following:[12]

Corollary 2.14. *For every $\varepsilon \in (0, 1 - \frac{dm}{|\mathbb{F}|})$, if $d \geq 2(|H| - 1)$,*

$$\mathsf{PT}_\varepsilon(\mathsf{TensorSum}) = \Omega\left(\frac{|H|^m}{\log(|\mathbb{F}|)}\right).$$

In order to prove Lemma 2.12, we first extend the framework of [BBM11] from the property testing model to the MAP model. More specifically, we show a methodology for proving lower bounds on MAPs via MA communication complexity lower bounds. We refer the reader to Section 2.8.1.2 for background on MA communication complexity.

Let Π be a property and let ψ be a simple combining operator (see Definition 2.7). For every proximity parameter $\varepsilon > 0$, denote by $\mathcal{C}^{\Pi}_{\psi,\varepsilon}$ the communication complexity problem in which Alice gets as input a function f and Bob gets as input a function g and they

[12] The corollary can be derived by setting $p = 1$, and the fact that any property tester is an MAP.

need to decide between a YES-instance, wherein $\psi(f,g) \in \Pi$, and a NO-instance, wherein $\psi(f,g)$ is ε-far from Π.[13] We prove the following lemma.

Lemma 2.13 (MAP lower bounds via MA communication complexity)**.** *For any* simple *combining operator* ψ*, any property* Π *and any proximity parameter* $\varepsilon > 0$*, if* $\Pi \in \mathsf{MAP}(p,q)$*, then* $\mathcal{C}^{\Pi}_{\psi,\varepsilon}$ *has an* MA *communication complexity protocol with a proof of length* p *and total communication* $2q|\psi|$*.*

Proof. Let V be an MAP verifier for Π with proof complexity p and query complexity q. We construct an MA communication complexity protocol for $\mathcal{C}^{\Pi}_{\psi,\varepsilon}$. Recall that Alice and Bob get as input function f and g (respectively) and have free access to a proof string $w \in \{0,1\}^p$.

The (honest) proof string for the protocol is simply the proof string w of the MAP with respect to $h \stackrel{\text{def}}{=} \psi(f,g)$. As their first step, Alice and Bob emulate the execution of the MAP protocol with respect to the proof string w using their common random string as the source of randomness (for the emulated verifier). Whenever the MAP verifier V queries the input at a point x, Alice and Bob compute $f(x)$ and $g(x)$ (respectively) and send their values to each other. Since ψ is a *simple* combining operator, each player can compute $h(x)$ from $x, f(x)$ and $g(x)$, and feed it as an answer to the emulated MAP verifier. The players accept if V accepts, and reject otherwise.

Observe that both players use the same common random string as the source of randomness, and forward the same values to the MAP verifier (i.e., both the proof string and the oracle answers). Therefore, they emulate the verifier identically.

Note that by the definition of the communication complexity problem, if $(f,g) \in \mathcal{C}^{\Pi}_{\psi,\varepsilon}$, then $h \in \Pi$; hence the verifier will accept. On the other hand, if the pair $(f,g) \notin \mathcal{C}^{\Pi}_{\psi,\varepsilon}$, then h is ε-far from Π, so the verifier will reject.

During the entire reduction, the players communicated $2|\psi|$ bits for every query of the verifier. Hence the total number of bits that were communicated is $2|\psi| \cdot q$. □

We proceed by stating Klauck's lower bound on the MA communication complexity of set-disjointness [Kla03], and use Lemma 2.13 to show a lower bound on the MAP complexity of the *Sub-Tensor Sum* property.

Theorem 2.15 ([Kla03])**.** *Every* MA *communication complexity protocol for* DISJ_n *with proof complexity* p *and communication complexity* c *satisfies* $p \cdot c = \Omega(n)$*.*

Proof of Lemma 2.12. Denote $k = |H|^m$ and by $f \cdot g$ the function $h(x) \stackrel{\text{def}}{=} f(x) \cdot g(x)$. Let $\mathcal{C}^{\mathsf{TensorSum}}_{\cdot,\varepsilon}$ be the communication complexity problem wherein Alice gets a function

[13] When proving property testing lower bounds via standard (i.e., non-MA) communication complexity lower bounds (using [BBM11] framework) one may also map YES-instances (respectively, NO-instances) of communication complexity problems to NO-instances (respectively, YES-instances) of property testing problems. This is possible due to the *symmetrical* definition of standard communication complexity (in fact, the above was used in the proof of Lemma 2.6). In contrast, the definition of MA communication complexity is *asymmetrical*; therefore when using our extension of the framework to MA one must map YES-instances to YES-instances, and NO-instances to NO-instances.

$f : \mathbb{F}^m \to \mathbb{F}$, Bob gets a function $g : \mathbb{F}^m \to \mathbb{F}$, and their goal is to decide whether $f \cdot g \in \mathsf{TensorSum}$ or $f \cdot g$ is ε-far from $\mathsf{TensorSum}$.

Recall that by Theorem 2.15 we know that every MA communication complexity protocol for DISJ_k with proof complexity p and communication complexity c satisfies $p \cdot c = \Omega(k)$. On the other hand, by Lemma 2.13 we know that if $\mathsf{TensorSum} \in \mathsf{MAP}(p, q)$, then $\mathsf{CC}(\mathcal{C}_{\cdot,\varepsilon}^{\mathsf{TensorSum}})$ has an MA communication complexity protocol with a proof of length p and a total of $2q \log |\mathbb{F}|$ communication.

Hence, to prove the lemma, it suffices to reduce DISJ_k to $\mathcal{C}_{\cdot,\varepsilon}^{\mathsf{TensorSum}}$ (this reduction takes place entirely within the setting of MA communication complexity). Toward this end, suppose that π is an MA communication complexity protocol for $\mathcal{C}_{\cdot,\varepsilon}^{\mathsf{TensorSum}}$ with proof complexity p and communication complexity c. We use π to construct an MA protocol for DISJ_k.

Let $a \in \{0,1\}^k$ and $b \in \{0,1\}^k$ be the respective inputs of Alice and Bob for the set-disjointness problem. Recall that $\mathbb{F}$ (a finite field), d (the individual degree), m (the dimension) and $H \subset \mathbb{F}$ are parameters of the $\mathsf{TensorSum}$ problem. The reduction to $\mathsf{TensorSum}$ proceeds as follows. First, Alice and Bob compute the low degree extension $\hat{a}$ and $\hat{b}$ of their respective inputs with respect to $\mathbb{F}, m, d$ and H. Namely, they associate their inputs a and b with indicator functions $a, b : H^m \to \{0,1\}$ by mapping $[k]$ to H^m in some canonical way. Then, they compute the (unique) polynomials $\hat{a}, \hat{b} : \mathbb{F}^m \to \mathbb{F}$ of individual degree $|H| - 1$ that agree with a and b (respectively) on H^m.

Denote by w the proof for the protocol π with respect to the input pair $(\hat{a}, \hat{b})$. The proof for the set disjointness problem is simply w. Alice and Bob proceed by running π on input $(\hat{a}, \hat{b})$, with respect to the proof w and proximity parameter ε and return its output.

Observe that if $(a, b) \in \mathsf{DISJ}_k$, then $\sum_{i \in [k]} a_i b_i = 0$ (where the summation is over the integers). Hence,

$$\sum_{x_1,\ldots,x_m \in H} \hat{a}(x_1, \ldots, x_m) \cdot \hat{b}(x_1, \ldots, x_m) = \sum_{x_1,\ldots,x_m \in H} a(x_1, \ldots, x_m) \cdot b(x_1, \ldots, x_m) = 0$$

(where the first summation is over $\mathbb{F}$, and the second summation is over the integers). Thus, $\hat{a} \cdot \hat{b} \in \mathsf{TensorSum}_{\mathbb{F},m,d,H}$ (here we use the lemma's hypothesis that $d \geq 2(|H| - 1)$ since $\hat{a} \cdot \hat{b}$ is the product of two polynomials of individual degree $|H| - 1$). We conclude that there exists a proof w of length p such that the MA communication complexity protocol for DISJ_k accepts with high probability.

On the other hand, if $(a, b) \notin \mathsf{DISJ}_k$, then (by the promise of having an intersection of size at most 1) it holds that $\sum_{i \in [k]} a_i b_i = 1$ (where the summation is over the integers). Hence

$$\sum_{x_1,\ldots, \in H} \hat{a}(x_1, \ldots, x_m) \cdot \hat{b}(x_1, \ldots, x_m) = \sum_{x_1,\ldots,x_m \in H} a(x_1, \ldots, x_m) \cdot b(x_1, \ldots, x_m) = 1$$

(where the first summation is over $\mathbb{F}$, and the second summation is over the integers). Thus, $\hat{a} \cdot \hat{b}$ is an m-variate polynomials of (individual) degree d ($\geq 2(|H| - 1)$) whose

sum over H^m is non-zero. By the Schwartz-Zippel lemma (see Section 2.8.1.4), and since $\varepsilon < 1 - \frac{dm}{|\mathbb{F}|}$, the function $\hat{a} \cdot \hat{b}$ is at least ε-far from TensorSum.

We conclude that every MAP verifier for TensorSum with q queries and p proof length must satisfy $q \cdot p \geq \Omega\left(\frac{k}{\log(|\mathbb{F}|)}\right)$. □

2.3.2.3 Proof of Theorem 2.13

In this section we complete the proof of Theorem 2.13, which states that for every constant $\alpha > 0$, there exists a property Π_α such that for every sublinear function $p : \mathbb{N} \to \mathbb{N}$, the query complexity of Π for MAP verifiers that use proofs of length p is upper bounded by $\frac{n^{1-\alpha+o(1)}}{p} \cdot \mathsf{poly}(1/\varepsilon)$ and lower bounded by $\tilde{\Omega}\left(\frac{n^{1-\alpha}}{p}\right)$.

Toward this end, we need to set the parameters of the TensorSum problem. Our parameters are governed by $n = |\mathbb{F}|^m$ (i.e., the size of the object equals n), $dm < |\mathbb{F}|/10$ (so that we can apply the Schwartz-Zippel lemma) and $d = 2(|H| - 1)$ (see Lemma 2.12). Since $p \cdot q = \tilde{\Omega}(|H|^m)$, and the object size is $|\mathbb{F}|^m$, we need to maximize the ratio $|H|/|\mathbb{F}|$ to obtain a better lower bound (while recalling that $|H| \leq d/2 - 1$).

For every constant $\alpha > 0$ and every integer $n \in \mathbb{N}$, let $\mathbb{F}$ be a finite field of size $(\log n)^{1/\alpha}$, let $m = \alpha \cdot \frac{\log n}{\log\log(n)}$, let H be some fixed (arbitrary) subset of $\mathbb{F}$ of size $|\mathbb{F}|^{1-\alpha}$ and let $d = 2(|H| - 1)$. Note that $|\mathbb{F}|^m = n$ and $|H|^m = n^{1-\alpha}$.

Lemma 2.11 guarantees the existence of an MAP for $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$ with proof complexity $(d+1)^\ell \cdot \log(|F|)$ and query complexity $|H|^{m-\ell} \cdot dm^2 \log(|H|)$ for every $\ell \in [m]$. Thus, for every parameter $p \in \{(d+1)^i \cdot \log(|\mathbb{F}|) : i \in \mathbb{N}\}$ (which corresponds to the proof length), we set:

$$\ell = \frac{\log(p) - \log\log(|F|)}{\log(d+1)}.$$

and apply Lemma 2.11. We obtain an MAP protocol for computing $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$ with a proof of length

$$(d+1)^\ell \cdot \log(|F|) = p$$

and query complexity:

$$|H|^{m-\ell} \cdot dm^2 \log(|H|) \cdot \mathsf{poly}(1/\varepsilon) = \frac{n^{1-\alpha}}{|H|^\ell} \cdot \mathsf{polylog}(n) \cdot \mathsf{poly}(1/\varepsilon). \tag{2.3}$$

By our setting of ℓ we have:

$$|H|^\ell = |H|^{\frac{\log p - \log\log|\mathbb{F}|}{\log(d+1)}} \geq 2^{\frac{\log|H|}{\log(2|H|)} \cdot (\log p - \log\log|\mathbb{F}|)} = \left(\frac{p}{\log|\mathbb{F}|}\right)^{1-\frac{1}{1+\log H}} \geq \frac{p}{n^{o(1)}} \tag{2.4}$$

where the first inequality follows from $d = 2(|H|-1) \leq 2|H|-1$ and the second inequality follows from our setting of $|H|$ and $|\mathbb{F}|$ (and since $p \leq n$). Combining Eq. (2.3) and Eq. (2.4) we have that the query complexity of the MAP is $\frac{n^{1-\alpha+o(1)}}{p} \cdot \mathsf{poly}(1/\varepsilon)$.

On the other hand, by Lemma 2.12, for every MAP for TensorSum with proof complexity p and query complexity q, it holds that $p \cdot q \geq \Omega\left(\frac{|H|^m}{\log|\mathbb{F}|}\right) = \tilde{\Omega}(n^{1-\alpha})$. The theorem follows.

2.3.3 MAP vs. IPP$[O(1)]$

In this section and the following one, we consider the power of MAP in comparison to the more general notion of IPP (for a formal definition of IPP, see Section 2.2.2.) Roughly speaking, in this section we show a property that requires $\sqrt{n}$ queries by an MAP verifier that uses a proof of length $\sqrt{n}$ but requires only $\mathsf{polylog}(n)$ queries by an IPP[3] verifier (i.e., an IPP with only 3-messages) that also uses a proof of length $\sqrt{n}$.

Theorem 2.16. *For every $\alpha > 0$, there exists a property Π_α such that:*

1. *The MAP complexity of Π_α is $\tilde{\Omega}\left(n^{1/2-\alpha}\right)$; and*
2. *There is an IPP[3] for Π_α with $\mathsf{polylog}(n) \cdot \mathsf{poly}(1/\varepsilon)$ query complexity and communication complexity $\tilde{O}(n^{1/2-\alpha+o(1)})$.*

The property that we use is the TensorSum property (introduced in Section 2.3.2). Note that the first part of Theorem 2.16 was already shown in Theorem 2.13, and so, to prove Theorem 2.16, what remains to be shown is that TensorSum can be tested by a 3-message IPP verifier that uses roughly $\sqrt{n}$ communication and $\mathsf{polylog}(n)$ queries.

Lemma 2.14. *If $dm < |\mathbb{F}|/10$, then there is a 3-message IPP for $\mathsf{TensorSum}_{\mathbb{F},d,m,H}$ (where $\mathbb{F}$ is a finite field, m is the dimension, d is the degree and $H \subset \mathbb{F}$) with communication complexity $O\left((d+1)^{m/2}\log(|\mathbb{F}|)\right)$ and query complexity $O\left(dm \cdot \mathsf{poly}(1/\varepsilon)\right)$.*

We note that Theorem 2.16 follows from Lemma 2.14 (and Lemma 2.12) by setting the parameters $\mathbb{F}, m, d, H$ as in Section 2.3.2.3. Namely, fix a finite field $\mathbb{F}$ of size $(\log n)^{1/\alpha}$, a dimension $m = \alpha \cdot \frac{\log n}{\log\log(n)}$, an arbitrary subset $H \subset \mathbb{F}$ of size $|F|^{1-\alpha}$ and set $d = 2(|H|-1)$. We proceed to prove Lemma 2.14

Proof of Lemma 2.14. The first part of the protocol closely resembles the MAP that was presented in Lemma 2.11. Indeed, the first message from the prover to the verifier is the polynomial Q that is (allegedly) the sum of P on H^ℓ sub-tensors of H^m, each of dimension $m-\ell$. The verifier checks that P is close to a low degree polynomial and that Q sums to 0, but the consistency check of P and Q is different. Recall that in Lemma 2.11, the verifier chose a random sub-tensor and checked the consistency of Q and P by reading all points in the sub-tensor. Using two additional messages we replace these queries by having the prover provide them. That is, after the prover "commits" to the sum of all sub-tensors, the verifier chooses one of them at random and sends its choice to the prover. Then, the prover provides the value of *all* points in that sub-tensor via a polynomial $W : \mathbb{F}^{m-\ell} \to \mathbb{F}$ of individual degree $|H| - 1$. The verifier can readily check the that the two polynomials Q and W sent by the prover are consistent with each other (using no queries to P), and that the second polynomial (i.e., W) is consistent with P using only a constant number of queries.

Similarly to the protocol of Section 2.3.2, the protocol uses a parameter ℓ except that in this case, an optimal result is obtained by fixing $\ell = m/2$ (but for simplicity of notations we keep ℓ as a parameter). The IPP[3] protocol, in which the prover is denoted

by $\mathcal{P}$ and the verifier is denoted by $\mathcal{V}$, is described in Section 2.3.3. It can be readily verified that by setting $\ell = m/2$, the query and communication complexities are as stated. We proceed to prove that completeness and soundness hold.

IPP[3] for TensorSum

Parameters: $\mathbb{F}$ (field), m (dimension), d (individual degree), $H \subset \mathbb{F}$ and $\ell = m/2$.

Input: a proximity parameter $\varepsilon \in (0, 1/3)$, and oracle access to a function $P : \mathbb{F}^m \to \mathbb{F}$.

1. $\mathcal{V}$ runs the low individual d-degree test (see Theorem 2.31) on P with respect to the proximity parameter ε. If the test fails then $\mathcal{V}$ rejects.
2. $\mathcal{P}$ sends to $\mathcal{V}$ an individual degree d multivariate polynomial $\tilde{Q} : \mathbb{F}^\ell \to \mathbb{F}$ of individual degree d (by specifying its $(d+1)^\ell$ coefficients), which allegedly equals
$$Q(x_1, \ldots, x_\ell) \stackrel{\text{def}}{=} \sum_{x_{\ell+1}, \ldots, x_m \in H} P(x_1, \ldots, x_m).$$
3. If $\sum_{x_1, \ldots, x_\ell \in H} \tilde{Q}(x_1, \ldots, x_\ell) \neq 0$, then $\mathcal{V}$ rejects.
4. $\mathcal{V}$ selects uniformly at random $r_1, \ldots, r_\ell \in_R \mathbb{F}$ and sends $r_1, \ldots, r_\ell$ to $\mathcal{P}$.
5. $\mathcal{P}$ sends to $\mathcal{V}$ an individual degree d multivariate polynomial $\tilde{W} : \mathbb{F}^{m-\ell} \to \mathbb{F}$ of individual degree d (by specifying its $(d+1)^{m-\ell}$ coefficients), which allegedly equals
$$W(x_{\ell+1}, \ldots, x_m) \stackrel{\text{def}}{=} P(r_1, \ldots, r_\ell, x_{\ell+1}, \ldots, x_m).$$
6. $\mathcal{V}$ selects at random $s_{\ell+1}, \ldots, s_m \in_R \mathbb{F}$, reads the value $z_{r_1, \ldots, r_\ell, s_{\ell+1}, \ldots, s_m}$ of the polynomial $P(r_1, \ldots, r_\ell, s_{\ell+1}, \ldots, s_m)$ using the self-correction algorithm (see Theorem 2.29) with soundness error $1/10$ and rejects if $z_{r_1, \ldots, r_\ell, s_{\ell+1}, \ldots, s_m} \neq W(s_{\ell+1}, \ldots, s_m)$.
7. $\mathcal{V}$ accepts if $\tilde{Q}(r_1, \ldots, r_\ell) = \sum_{x_{\ell+1}, \ldots, x_m \in H} \tilde{W}(x_{\ell+1}, \ldots, x_m)$ and rejects otherwise.

Figure 2.3: IPP[3] for TensorSum

Completeness. If $P \in$ TensorSum, then P has individual degree d and the low degree tests passes. In this case $\tilde{Q} = Q$ and $\tilde{W} = W$ and therefore all the verifier's tests pass (since $\sum_{x_1, \ldots, x_\ell \in H} Q(x_1, \ldots, x_\ell) = 0$ holds as well).

Soundness. Let $\varepsilon > 0$ and let $P : \mathbb{F}^m \to \mathbb{F}$ be ε-far from TensorSum. If P is ε-far from having individual degree d, then the low degree test rejects with probability at least $1/2$ and so we assume that P is ε-close to an individual degree d polynomial P'. The

(cheating) prover sends two polynomials $\tilde{Q}$ and an $\tilde{W}$. We proceed to prove two claims regarding these polynomials.

Claim 2.14.1. *If $\tilde{Q}(x_1,\ldots,x_\ell) \equiv \sum_{x_{\ell+1},\ldots,x_m\in H} P'(x_1,\ldots,x_m)$ (as formal polynomials over $x_1,\ldots,x_\ell$), then the verifier rejects with probability 1.*

Proof. Observe that $\sum_{x_1,\ldots,x_m\in H} P'(x_1,\ldots,x_m) \neq 0$, as otherwise P is ε-close to TensorSum. Therefore, if the polynomials $\tilde{Q}(x_1,\ldots,x_\ell)$ and $\sum_{x_{\ell+1},\ldots,x_m\in H} P'(x_1,\ldots,x_m)$ are equal, then the verifier rejects when testing whether $\sum_{x_1,\ldots,x_\ell\in H} \tilde{Q}(x_1,\ldots,x_\ell) = 0$. □

Claim 2.14.2. *For every value of $r_1,\ldots,r_\ell \in \mathbb{F}$, if the prover sends an individual-degree d polynomial $\tilde{W}(x_{\ell+1},\ldots,x_m)$ (which depends on $r_1,\ldots,r_\ell$) that differs from the polynomial $P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m)$ (as formal polynomials in $x_{\ell+1},\ldots,x_m$), then the verifier rejects with probability at least $2/3$.*

Proof. Assume that $\tilde{W}(x_{\ell+1},\ldots,x_m) \not\equiv P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m)$. Thus, the polynomials $\tilde{W}(x_{\ell+1},\ldots,x_m)$ and $P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m)$ are two different $(m-\ell)$-variate polynomials of individual degree d and, by the Schwartz-Zippel Lemma, they can agree on at most a $\frac{d(m-\ell)}{|\mathbb{F}|} < 0.1$ fraction of their domain. Therefore, with probability 0.9 over the verifier's choice of $s_{\ell+1},\ldots,s_m \in \mathbb{F}$, it holds that

$$\tilde{W}(s_{\ell+1},\ldots,s_m) \neq P'(r_1,\ldots,r_\ell,s_{\ell+1},\ldots,s_m).$$

Using the self-correction procedure, with probability at least 0.9, the verifier correctly obtains the value $z_{r_1,\ldots,r_\ell,s_{\ell+1},\ldots,s_m} = P'(r_1,\ldots,r_\ell,s_{\ell+1},\ldots,s_m)$. Hence, with probability at least $0.9^2 > 2/3$, the verifier rejects when testing whether $z_{r_1,\ldots,r_\ell,s_{\ell+1},\ldots,s_m} = \tilde{W}(s_{\ell+1},\ldots,s_m)$. □

By Claim 2.14.2, we can assume that

$$\tilde{W}(x_{\ell+1},\ldots,x_m) \equiv P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m) \tag{2.5}$$

(since otherwise the verifier rejects). On the other hand, by Claim 2.14.1 and using the Schwartz-Zippel Lemma, with probability at least $1-\frac{d\ell}{|\mathbb{F}|}$ over the choice of $r_1,\ldots,r_\ell \in_R \mathbb{F}$, it holds that

$$\tilde{Q}(r_1,\ldots,r_\ell) \neq \sum_{x_{\ell+1},\ldots,x_m\in H} P'(r_1,\ldots,r_\ell,x_{\ell+1},\ldots,x_m) = \sum_{x_{\ell+1},\ldots,x_m\in H} \tilde{W}(x_{\ell+1},\ldots,x_m)$$

where the last equality is due to Eq. (2.5). Hence, the verifier rejects with probability $1-\frac{d\ell}{|\mathbb{F}|} > 0.9$ when testing whether $\tilde{Q}(r_1,\ldots,r_\ell) = \sum_{x_{\ell+1},\ldots,x_m\in H} W(x_{\ell+1},\ldots,x_m)$. This completes the proof of Lemma 2.14. □

2.3.4 Exponential Separation between MAP and IPP

In this section we show an exponential separation between MAP and general IPP. Namely, we show a property that has MAP complexity roughly $\sqrt{n}$ but has IPP complexity $\mathsf{polylog}(n)$. In contrast to the IPP of Section 2.3.3 (which used $O(1)$ messages) here we use an IPP with *poly-logarithmically* many messages.

Theorem 2.17. *For every $\alpha > 0$, there exists a property Π_α such that:*

1. *The* MAP *complexity of Π_α is $\tilde{\Omega}\left(n^{1/2-\alpha} \cdot \mathsf{poly}(1/\varepsilon)\right)$; and*
2. *Π_α has an* IPP *with query complexity $\mathsf{polylog}(n) \cdot \mathsf{poly}(1/\varepsilon)$ and communication complexity $\mathsf{polylog}(n)$.*

Moreover, the PT *complexity of Π_α is $\tilde{\Theta}(n^{1-\alpha})$.*

To prove Theorem 2.17, we yet again use the TensorSum problem. The first part of the theorem follows directly from Theorem 2.13 and the query complexity of property testers (without a proof) is implied by Corollary 2.14.[14] Thus, to prove the theorem, all that remains is to show an IPP protocol for TensorSum.

Lemma 2.15. *If $d \cdot m < \mathbb{F}/10$, then there exists an m-round* IPP *for $\mathsf{TensorSum}_{\mathbb{F},m,d,H}$ with communication complexity $O(dm \log |F|)$, and query complexity $O(dm \cdot \mathsf{poly}(1/\varepsilon))$.*

Proof. The proof of Lemma 2.15 follows by adapting the well-known sum-check protocol of Lund *et al.* [LFKN92] to the settings of interactive proofs of proximity. Recall that the sum-check protocol is an interactive protocol that enables verification of the a claim of the form:

$$\sum_{x_1,\ldots,x_m \in H} P(x_1,\ldots,x_m) = 0.$$

where P is a low-degree polynomial. The difference between our setting and the classical setting of the sum-check protocol of [LFKN92] is that in the latter the verifier has explicit and direct access to P.[15] In our setting the verifier only has *oracle access* to a function that is *allegedly* a low-degree polynomial. However, we observe that the sum-check protocol can be extended to this setting by having the verifier (1) test that the function is close to a low-degree polynomial P, (2) obtain values from P via self-correction, and (3) run the sum-check protocol as-is with respect to the self-corrected P. The IPP protocol is described in Fig. 2.4, where the prover is denoted by $\mathcal{P}$, the verifier is denoted by $\mathcal{V}$ and all arithmetic is over the field $\mathbb{F}$. (For a high level description of the sum-check protocol, see Section 2.8.1.5.)

We note that during the run of the IPP the prover sends m degree d univariate polynomial, and the verifier sends m elements in $\mathbb{F}$. Thus, the total communication complexity of the IPP is $O(dm \log |F|)$. The only queries that the verifier performs are for the low degree test and the self-correction, which total in $O(dm \cdot \mathsf{poly}(1/\varepsilon))$ queries.

[14] We note that the property testing upper bound of $\tilde{O}(n^{1-\alpha})$ can be obtained by a verifier that tests for low degree and reads all points in H^m using self correction.

[15] An additional minor difference is that in the [LFKN92] protocol the set H is fixed to $\{0,1\}$, but this is common in the PCP literature (most notably in [BFLS91]).

IPP for TensorSum

Parameters: $\mathbb{F}$ (field), m (dimension), d (individual degree) and $H \subset \mathbb{F}$.

Input: a proximity parameter $\varepsilon \in (0, 1/3)$, and oracle access to a function $P : \mathbb{F}^m \to \mathbb{F}$.

1. $\mathcal{V}$ runs the individual degree d test (see Theorem 2.31) on P with respect to proximity parameter ε, and rejects if the test fails.
2. Let $\nu_0 \stackrel{\text{def}}{=} 0$.
3. For $i \leftarrow 1, \ldots, m$:
   (a) $\mathcal{P}$ sends to $\mathcal{V}$ a degree d univariate polynomial $\tilde{P}_i : \mathbb{F} \to \mathbb{F}$ (by specifying its $d+1$ coefficients), which allegedly equals:
   $$P_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1},\ldots,x_m \in H} P(r_1, \ldots, r_{i-1}, z, x_{i+1}, \ldots, x_m).$$
   (b) $\mathcal{V}$ verifies that $\sum_{z \in H} \tilde{P}_i(z) = \nu_{i-1}$.
   (c) $\mathcal{V}$ selects uniformly at random $r_i \in_R \mathbb{F}$ and sets $\nu_i \stackrel{\text{def}}{=} \tilde{P}_i(r_i)$.
   (d) If $i \neq m$, then $\mathcal{V}$ sends r_i to $\mathcal{P}$.
4. $\mathcal{V}$ obtains the value of z^* of $P(r_1, \ldots, r_m)$ via self-correction (see Theorem 2.29) with soundness error 0.1.
5. $\mathcal{V}$ verifies that $z^* = \nu_m$.

Figure 2.4: IPP for $\mathsf{TensorSum}_{m,d,\mathbb{F},S,c}$

Completeness. If $P \in$ TensorSum, then the low degree test always passes, and since we have $\sum_{x \in H^m} P(x) = 0$, and the prover supplies the correct polynomials (i.e., $\tilde{P}_i = P_i$ for every $i \in [m]$), the verifier always accepts.

Soundness. Suppose that $P : \mathbb{F}^m \to \mathbb{F}$ is ε-far from TensorSum. Let $\mathcal{P}^*$ be a cheating prover that attempts to convince the verifier of the false statement $P \in$ TensorSum. If P is ε-far from having individual degree d, then the verifier rejects with probability $1/2$. Thus, we focus on the case that P is ε-close to a polynomial P' of individual degree d.

For every $i \in [m]$, let:

$$P'_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1},\ldots,x_m \in H} P'(r_1, \ldots, r_{i-1}, z, x_{i+1}, \ldots, x_m)$$

(where the values r_i are those sent from the verifier to the prover). The next two claims

relate the polynomials P'_i to the polynomials $\tilde{P}_i$ sent by the prover $\mathcal{P}^*$. Recall that both polynomials depend only on $r_1, \ldots, r_{i-1}$.

Claim 2.15.1. *If $\tilde{P}_1 \equiv P'_1$, then the verifier rejects with probability 1.*

Proof. Observe that $\sum_{x \in H^m} P'(x) \neq 0$ must hold, since otherwise $P \in \mathsf{TensorSum}$. Therefore $\sum_{z \in H} P'_1(z) = 0$, and so, if $\tilde{P}_1 \equiv P'_1$, then the verifier rejects when testing that $\sum_{z \in H} \tilde{P}_1(z) = 0$. □

Claim 2.15.2. *For every $i \in [m-1]$ and every $r_1, \ldots, r_{i-1} \in \mathbb{F}$, if $\tilde{P}_i \not\equiv P'_i$ then, with probability at least $1 - d/|\mathbb{F}|$ over the choice of r_i, if $\tilde{P}_{i+1} \equiv P'_{i+1}$ then the verifier rejects.*

Proof. If $\tilde{P}_{i+1} \equiv P'_{i+1}$ then $\sum_{z \in H} \tilde{P}_{i+1}(z) = \sum_{z \in H} P'_{i+1}(z) = P'_i(r_i)$. Thus, since the polynomials $\tilde{P}_i$ and P'_i differ, with probability at least $1 - d/|\mathbb{F}|$ over the choice of $r_i \in_R \mathbb{F}$ it holds that $\tilde{P}_i(r_i) \neq P'_i(r_i)$, and in this case the verifier will reject when testing whether $\sum_{z \in H} \tilde{P}_{i+1}(z) = \nu_i$, since $\nu_i = \tilde{P}_i(r_i)$. □

By Claim 2.15.3 and an application of the union bound, with probability $1 - dm/|\mathbb{F}|$, if there exists an $i \in [m-1]$ such that $\tilde{P}_i \not\equiv P'_i$ but $\tilde{P}_{i+1} \equiv P'_{i+1}$ then the verifier rejects. By Claim 2.15.1, we can assume that $\tilde{P}_1 \not\equiv P'_1$ and so we need only consider the case that for every $i \in [m]$ it holds that $\tilde{P}_i \not\equiv P'_i$. The following claim shows that also in this case the verifier rejects with probability at least $2/3$. The theorem follows.

Claim 2.15.3. *For every $r_1, \ldots, r_{m-1} \in \mathbb{F}$, if $\tilde{P}_m \not\equiv P'_m$, then the verifier rejects with probability at least $2/3$ (over the choice of r_m and the self-correction procedure).*

Proof. If $\tilde{P}_m \not\equiv P'_m$ then these are two distinct degree d polynomials, which can agree on at most d points. Thus, with probability $1 - d/|\mathbb{F}|$, it holds that $\tilde{P}_m(r_m) \neq P'_m(r_m)$ (over the choice of $r_m \in_R \mathbb{F}$). Now, the self-correction algorithm guarantees that the verifier computes $z^* = P'(r_1, \ldots, r_m) = P'_m(r_m)$ correctly with probability 0.9. In such case, the verifier rejects with probability $1 - d/|\mathbb{F}|$ when testing that $z^* = \tilde{P}_m(r_m)$. It follows that the verifier rejects with probability $0.9 \cdot (1 - d/|\mathbb{F}|) > 2/3$. □

This completes the proof of Lemma 2.15. □

2.4 General Transformations

In this section we show general transformations on MAP proof-systems. In Section 2.4.1 we show general transformations from MAPs with restricted proofs into PT. In Section 2.4.2 we show a general transformation from MAPs that have two-sided error into MAPs that have one-sided error.

2.4.1 From MAP to PT

In this section we show that MAPs with restricted proofs can be emulated by property testers. We show two such results. Theorem 2.18 shows that every MAP that uses a *very short* proof can be emulated by a property tester, and Theorem 2.19 shows that even MAPs with long proofs *in which the verifier's queries are proof oblivious* (see Definition 2.2) can also be emulated. We note that in both constructions the tester may be inefficient in terms of *computational* complexity (even if the original MAP verifier can be implemented efficiently).

Theorem 2.18. *If the property* Π *has an* MAP *verifier that makes* q *queries and uses a proof of length* p*, then* Π *has a property tester that makes* $\tilde{O}(2^p \cdot q)$ *queries. Moreover, if the* MAP *tester has one-sided error, then the resulting property tester has one-sided error.*

Proof. Let V be an MAP verifier for Π with query complexity q and proof complexity p. We start by running the verifier $O(p)$ times using fresh (independent) randomness, but the same proof string, and ruling by majority vote. We obtain an MAP verifier V' for Π that has soundness error $2^{-(p+2)}$, uses $q' \stackrel{\text{def}}{=} O(p \cdot q)$ queries and a proof of length p.

We use V' to construct a property tester T for Π. The tester T, given oracle access to a function f, simply enumerates over all possible 2^p proof strings for V'. For each proof string $w \in \{0,1\}^p$, the tester T emulates V' (using fresh randomness) while feeding it the proof string w, and forwarding its oracle queries to f. If for some string w the verifier accepts, then T accepts. Otherwise, it rejects. Clearly, T has query complexity $2^p \cdot q'$.

If $f \in \Pi$, then there exists a proof string w that will make V' accept, with probability at least $1 - 2^{-(p+2)}$. Therefore, T accepts in this case with probability at least $2/3$. On the other hand, if f is ε-far from Π, then no string w will make V' accept with probability greater than $2^{-(p+2)}$. Thus, by the union bound, T will accept with probability at most $2^p \cdot 2^{-(p+2)} < 1/3$.

The furthermore clause of Theorem 2.18, follows by noting that both the parallel repetition and proof enumeration steps preserve one-sided error. □

The tester of Theorem 2.18 makes $O(p \cdot q)$ queries for every one of the possible 2^p proof strings. However, the fact that these queries were chosen independently (i.e., based on fresh randomness) is not used in the soundness argument. Indeed, for soundness we simply applied a union bound, which would have worked just as well if the queries were not independent (i.e., were determined based on the same randomness). This leads us to consider using the *same* sequence of queries for all of the proofs in the emulation step. The problem that we run into is in the completeness condition. Namely, a sequence of queries that was generated with respect to a particular proof may not be "good" for a different proof. More precisely, if the distribution of queries that the MAP verifier generates (heavily) depends on the proof, then the only guarantee that we have is that the MAP verifier will be correct when emulated with a distribution of queries that matches the specific good proof.[16] Hence, we may indeed have to generate a different sequence of queries for every possible proof string.

[16] For an example of such MAPs, see Theorem 2.7 and Theorem 2.20.

However, as proved in the following theorem, if the tester makes *proof oblivious queries* (see Definition 2.2), then the foregoing problem can be avoided and indeed it suffices to make only one sequence of queries, and reuse this sequence for all the 2^p emulations.

Theorem 2.19. *If the property* Π *has an* MAP *verifier that makes* q proof oblivious queries *and uses a proof of length* p*, then* Π *has a property tester that makes* $O(p \cdot q)$ *queries. Moreover, if the* MAP *verifier has one-sided error, then the resulting property tester has one-sided error.*

Proof. Let V be an MAP verifier for Π with query complexity q and proof complexity p, and let V' be exactly as in the proof of Theorem 2.18 (i.e., an MAP verifier for Π with soundness error $2^{-(p+2)}$, using $q' = O(p \cdot q)$ queries and a proof of length p).

As hinted above, the construction of the property tester T differs from that in Theorem 2.18. The tester T is given oracle access to f. It first emulates V' using an arbitrary (dummy) proof string, denoted w_0, a random string r, and by forwarding V''s queries to f. The key observation here is that the distribution of the queries does not depend on the proof at all, and so an arbitrary proof would suffice for our needs. Thus, T obtains a sequence $\bar{a}_r^f = (a_1, \ldots, a_{q'})$ of answers (corresponding to queries specified by r and the previous answers). Now, T enumerates over all possible 2^p proof strings for V', and for each proof string $w \in \{0,1\}^p$ it emulates V' while feeding it the proof string w, the random string r, and the answer sequence $\bar{a}_r^f$. If for some string w the verifier accepts, then T accepts. Otherwise, it rejects.

If $f \in \Pi$, then there exists a proof string w that will make V' accept with probability at least $2/3$. The key point is that since the distribution of the queries does not depend on w. Hence, the queries actually made by T (using the dummy proof w_0) are identical to those V' would have made using the proof w (and the same randomness as T). Hence, T accepts in this case with probability at least $2/3$ (and in case V' has one-sided error, then T accepts with probability 1). On the other hand, similarly to the proof of Theorem 2.18, if f is ε-far from Π then no string w will make V' accept with probability greater than $2^{-(p+2)}$. Thus, by the union bound, T will accept in this case with probability at most $2^p \cdot 2^{-(p+2)} < 1/3$. □

2.4.2 From Two-Sided Error MAP to One-Sided Error MAP

In this section we show a general result transforming any MAP (which may have two-sided error) into an MAP with *one-sided* error, while incurring only a poly-logarithmic overhead to the query and proof complexities. The construction is based on the ideas introduced in Lautemann's [Lau83] proof that BPP is contained the polynomial hierarchy coupled with the observation that MAPs may have very low randomness complexity (adapted from [GS10b], which in turns follows an idea of Newman [New91]). We note that both the verifier and the proof generation algorithm in this construction may be *inefficient* in the computational complexity sense. (This is a consequence of each one of the two parts of the transformation).

Theorem 2.20. *Let Π be a property of functions $f_n : D_n \to R_n$, where $|R_n| \leq \mathsf{exp}\big(\mathsf{poly}(n)\big)$. If Π has a* two-sided error MAP *with q queries and a proof of length p, then Π has a* one-sided error MAP *with $O(q \cdot \mathsf{polylog}(n))$ queries and a proof of length $O(p + \mathsf{polylog}(n))$.*

We note that typically $|R_n| \leq n$ and that properties for which $|R_n| > \mathsf{exp}(\mathsf{poly}(n))$ seem quite pathological. Before proceeding to the proof of Theorem 2.20, we note that as a direct application of the theorem we obtain the following relation between two-sided error property testers and one-sided error MAP (denoted MAP_1).

Corollary 2.21. *For every function $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ it holds that:*

$$\mathsf{PT}(q) \subseteq \mathsf{MAP}_1(\mathsf{polylog}(n), q \cdot \mathsf{polylog}(n)).$$

The proof of Theorem 2.20 is based on two lemmas. The first, Lemma 2.16, shows that a two-sided error MAP verifier that has low *randomness complexity*, can be transformed into a one-sided error MAP. The proof of this lemma is based on the technique of Lautemann [Lau83]. The second lemma (Lemma 2.17) shows that the Goldreich-Sheffet [GS10b] technique for reducing the randomness of *property testers* can also be used to reduce the randomness of MAP verifiers.

Lemma 2.16. *If the property Π has a* two-sided MAP *verifier that makes q queries, uses a proof of length p, and has randomness complexity r, then Π has a* one-sided MAP *verifier that makes $O(q \cdot r \log r)$ queries and uses a proof of length $O(p + r^2 \log r)$.*

Proof. Following [Lau83], the construction involves two main steps. The first step is a parallel repetition step that significantly reduces both the completeness and soundness errors of the MAP. At this point, almost the entire set of possible random strings lead to accepting inputs that have the property and rejecting inputs that are far from the property. The main observation is that there must exist relatively few "shifts" $s_1, \ldots, s_t$ such that for an input that has the property, for *every* random string r there exists a shift s_i such that $r \oplus s_i$ leads to accepting, whereas if the input is far from the property, then with high probability over the choice of r, no shift will result in accepting. Details follow.

Let $V_{(2)}$ be a *two-sided* error MAP verifier for a property Π with query complexity $q \stackrel{\text{def}}{=} q(n, \varepsilon)$, proof complexity $p \stackrel{\text{def}}{=} p(n)$ and randomness complexity $r \stackrel{\text{def}}{=} r(n, \varepsilon)$. To prove the theorem we construct a *one-sided error* MAP verifier $V_{(1)}$ for Π.

Let $V_{(2)'}$ be the two-sided error MAP obtained by taking the majority of $m = \Theta(\log r)$ repetitions of $V_{(2)}$ using fresh random coins but using *the same proof string* for all repetitions. By the Chernoff bound, this amplification yields both completeness and soundness errors that are at most $\delta \stackrel{\text{def}}{=} 2^{-\Omega(m)}$, which may be made smaller than $\frac{1}{c \cdot rm}$ for any desired constant $c > 0$. Note that $V_{(2)'}$ has query complexity $q' \stackrel{\text{def}}{=} qm$, proof complexity $p' \stackrel{\text{def}}{=} p$, and randomness complexity $r' \stackrel{\text{def}}{=} rm$.

Denote by $V^f_{(2)'}(w; s)$ the (deterministic) output of $V^f_{(2)'}(w)$ when invoked with the random string s. We construct the one-sided error MAP verifier $V_{(1)}$ as follows. The

proof string for $V_{(1)}$ consists of the original proof string w for $V_{(2)}$ as well as a sequence of strings $(s_1, \ldots, s_t)$ each of length r', where $t = \Theta(r)$ such that $\delta^t < 2^{-r'}$ and $\delta t < \frac{1}{3}$. Given the proof string $(w, s_1, \ldots, s_t)$, the verifier $V_{(1)}$ chooses a random string $s \in_R \{0,1\}^{r'}$ and runs $V^f_{(2)'}(w; s \oplus s_i)$ for each $i \in [t]$. If for some $i \in [t]$ the test accepts, then $V_{(1)}$ accepts; otherwise it rejects. The proof and query complexities can be readily verified, and so we proceed to prove the completeness and soundness of $V_{(1)}$.

Completeness. Let $f \in \Pi$ of size n and let $\varepsilon > 0$. Then, by the completeness of $V_{(2)'}$, there exists a proof string w such that $\Pr_{s \in \{0,1\}^{r'}}[V^f_{(2)'}(w; s) = 1] \geq 1 - \delta$. We show that there exists a sequence $(s_1, \ldots, s_t)$ such that $\Pr_{s \in \{0,1\}^{r'}}[V^f_{(1)}(w, s_1, \ldots, s_t; s) = 1] = 1$.

To show that such a sequence $(s_1, \ldots, s_t)$ exists we use the probabilistic method. Specifically, we consider a sequence that is chosen uniformly at random, that is, each $s_i \in_R \{0,1\}^{r'}$. By the union bound,

$$\Pr_{s_1,\ldots,s_t}\left[\exists s \text{ s.t. } \forall i \in [t],\, V^f_{(2)'}(w; s \oplus s_i) = 0\right] \leq \sum_s \Pr_{s_1,\ldots,s_t}\left[\forall i \in [t],\, V^f_{(2)'}(w; s \oplus s_i) = 0\right], \tag{2.6}$$

but since the s_i's are independent, for every $s \in \{0,1\}^{r'}$,

$$\Pr_{s_1,\ldots,s_t}\left[\forall i \in [t],\, V^f_{(2)'}(w; s \oplus s_i) = 0\right] = \prod_{i=1}^{t} \Pr_{s_i}\left[V^f_{(2)'}(w; s \oplus s_i) = 0\right] \leq \delta^t. \tag{2.7}$$

Combining Equations (2.6) and (2.7) we obtain that:

$$\Pr_{s_1,\ldots,s_t}\left[\exists s \text{ s.t. } \forall i \in [t],\, V^f_{(2)'}(w; s \oplus s_i) = 0\right] \leq 2^{r'} \cdot \delta^t < 1.$$

and (zero-error) completeness follows.

Soundness. Let f of size n be ε-far from having the property Π for $\varepsilon > 0$. Then, by the soundness of $V_{(2)'}$, for every proof string w, the verifier $V_{(2)'}$ accepts f with probability at most δ. Hence, by the union bound,

$$\Pr_s\left[\exists i \in [t] \text{ s.t. } V^f_{(2)'}(w; s \oplus s_i) = 1\right] \leq \sum_{i \in [t]} \Pr_s\left[V^f_{(2)'}(w; s \oplus s_i) = 1\right] \leq t \cdot \delta < 1/3$$

and the lemma follows. □

Lemma 2.17. *Let Π be a property of functions $f_n : D_n \to R_n$, where $|R_n| \leq \exp(\mathsf{poly}(n))$. If Π has an MAP verifier that makes q queries, uses a proof of length p, and has randomness complexity r, then Π has an MAP verifier that makes q queries, uses a proof of length p and has randomness complexity $O(\log n)$.*

Proof. The proof follows the proof of [GS10b] with a minor modification to handle the dependence of the verifier on the proof. Namely, using the probabilistic method, we show the existence of a small subset of the random strings that behaves similarly to the entire set.

Let Π be a property of functions $f_n : D_n \to R_n$, where $|R_n| = \exp\big(\mathsf{poly}(n)\big)$ (and where $D_n = [n]$, cf. Section 2.2), and let V be the MAP verifier of the lemma statement. Fix an input length n and let $D \stackrel{\text{def}}{=} D_n$, $R \stackrel{\text{def}}{=} R_n$ and $p \stackrel{\text{def}}{=} p(n)$. Consider a $2^r \times |R|^{|D|} \cdot 2^p$ matrix where the rows correspond to all possible random strings γ used by the verifier and the columns correspond to pairs (f, w) of functions $f : D_n \to R_n$ and possible proofs $w \in \{0,1\}^p$. The entry $(\gamma, (f, w))$ of the matrix corresponds to the output of $V^f(w; \gamma)$, that is, the output of the verifier when given oracle access to f, the proof string w and random coins γ.

Note that for every function $f \in \Pi$, by the completeness of V, there exists a proof string w such that the average of the (f, w) column is at least $2/3$. Similarly, by the soundness of V, for functions that are ε-far from Π and *every* proof string w the average of the (f, w) column is at most $1/3$.

We show that there exists a multi-set, S, of size $\mathsf{poly}(n)$ of the rows such that the average of every column when taken over the rows of S is at most $1/7$-far from the average taken over all rows. Thus, we obtain an MAP verifier that uses only $\log_2 |S| = O(\log n)$ random coins, by simply running the original tester V but with respect to random coins selected uniformly from S (rather than from $\{0,1\}^r$). To obtain soundness and completeness error $1/3$ we use $O(1)$ parallel repetitions.

We use the probabilistic method to show the existence of a small multi-set S as above. Consider a multi-set S of the rows, of size t, chosen uniformly at random and fix some function f and proof string w. By the Chernoff bound, with probability $2^{-\Omega(t)}$ over the choice of S, the average over the rows in S of the (f, w)-column is $1/7$-close to the average over all rows. Thus, by setting $t = \log(|R|^{|D|} \cdot 2^p)$ and applying the union bound, we obtain that there exists a multi-set S as desired.

Since the new verifier selects at random from S, it can be implemented using $\log_2 t$ random coins. We complete the proof by noting that the proof length p can always be made to satisfy $p \leq n$ (since a proof of length n suffices to test any property using only $O(1/\varepsilon)$ queries, see discussion in Section 2.1.2), that the domain size is n and that $|R| \leq \exp(\mathsf{poly}(n))$ (by the hypothesis). □

Theorem 2.20 follows by applying the randomness reducing transformation of Lemma 2.17, and then applying Lemma 2.16 to the resulting MAP verifier.

2.5 An Extremely Hard Property for MAPs

As noted in the introduction, every property has an MAP that uses a proof of length n and makes only $O(1/\varepsilon)$ queries (where the proof is simply the object itself). In contrast, in this section we show that for "almost all" properties Π, every MAP for Π that uses a proof that is even $n/100$ bits long, requires $\Omega(n)$ queries.

Our result is actually slightly stronger. Roughly speaking, we show that for every t, a random property of size 2^t can be tested (without a proof) using $O(t)$ queries, but any MAP that uses a proof of length even $t/100$ must make $\Omega(t)$ queries in order test this property.

In the following we consider properties that are sets of strings rather than functions. We note that a function formulation (as in Definition 2.1) can be easily obtained by mapping every string $x \in \{0,1\}^n$ to the function $f_x : [n] \to \{0,1\}$, defined as $f_x(i) = x_i$.

Theorem 2.22. *Let $t = t(n) < n/10$. Every property $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ (where $\Pi_n \subseteq \{0,1\}^n$) of size 2^t can be tested with $O(t/\varepsilon)$ queries (without using a proof), but for every $n \in \mathbb{N}$, for 99% of sets $\Pi_n \subseteq \{0,1\}^n$ of size 2^t, it holds that every MAP for testing $\varepsilon < 1/4$ proximity to Π_n that uses a proof of length p must make at least $t - p - O(\log n)$ queries.*

The rest of this section is devoted to the proof of Theorem 2.22, which is inspired by [GGR98, Section 4.1] and uses also ideas from [RVW13, Section 4]. We remark that while Theorem 2.22 holds for almost all properties, finding an *explicit* property for which a similar statement holds is an interesting open question.

The key idea in the proof of Theorem 2.22 is to show that MAPs that use a relatively short proof and make relatively few queries can be represented by a small class of functions. Since this class of functions is small, we argue that a (small) *random* set $S \subseteq \{0,1\}^n$, viewed as a property, will fool every MAP, in the sense that no MAP verifier can distinguish between a random element in S and a random element in $\{0,1\}^n$.

The foregoing intuition is formalized by the following lemma shows that exists a set of *randomized decision trees* (see definition below) such that for every MAP, there exists a subset of the decision trees such that the MAP accepts an input x (with probability at least $2/3$) if and only if at least one of the randomized decision trees accepts x (with probability at least $2/3$).

Lemma 2.18. *Let $\varepsilon \in (0, 1/4)$. For every $n \in \mathbb{N}$ and for every $p, q \leq n$, there exists a class of functions $\mathcal{F}^{(n)}_{p,q}$ of size $2^{(\mathsf{poly}(n)\cdot 2^{p+q})}$ of functions from $\{0,1\}^n$ to $\{0,1\}$, such that the following holds. For every MAP verifier V for testing ε-proximity to $\Pi_n \subseteq \{0,1\}^n$ that uses a proof of length p and q queries, it holds that $I_V \in \mathcal{F}^{(n)}_{p,q}$, where $I_V(x)$ is defined as the indicator function for the event that there exists some $\pi \in \{0,1\}^p$ such that $\Pr[V^x(n, \varepsilon, \pi) = 1] \geq 2/3$.*

Note that the order of quantifiers in Lemma 2.18 is such that the class of functions is the same for *every* MAP verifier (and depends only on p and q). This will be crucial in showing that a random set fools *every* MAP verifier. Also note that if $p+q \ll n$, then the size of $\mathcal{F}$ is quite small relative to the class of all functions from $\{0,1\}^n$ to $\{0,1\}$ (which has size 2^{2^n}).

Proof of Lemma 2.18. To facilitate the proof of Lemma 2.18, it will be useful to describe standard testers (which do not use a proof) as *randomized decision trees*. Our main observation is that, roughly speaking, an MAP can be expressed as an OR of randomized decision trees.

Recall that a randomized decision tree is a model of computation for computing a randomized function $f : \{0,1\}^n \to \{0,1\}$. The randomized decision tree is a rooted ordered binary tree. Each internal vertex of the tree is labeled with a value $i \in \{1,\ldots,n,*\}$ and the leaves of the tree are labeled with 0 or 1. (We think of a node that is labeled with $i \in [n]$ as representing the reading of the i^{th} bit, and of a node that is labeled with $*$ as representing a random coin toss.) Given an input $x \in \{0,1\}^n$, the decision tree is recursively evaluated as follows. If the root's label is $*$, then one of its two children is selected uniformly at random, and we recurse on that child. Otherwise (i.e., $i \in [n]$), if $x_i = 0$, then we recurse on the left subtree, and if $x_i = 1$, then we recurse on the right subtree. Once a leaf is reached, we output the label of that leaf and halt. If T is a randomized decision tree, we denote by $T(x)$ the (random variable that corresponds to) the output of T on input x.

The size of the decision tree is defined as the number of vertices in the tree, and the depth of the tree is defined as the longest path between the root of the tree and one of its leaves. (See [BdW02] for an extensive survey of decision tree complexity.) Let RDT_s be the set of all randomized decision trees of size s. For every $T_1,\ldots,T_t \in \mathsf{RDT}_s$ let $f_{T_1,\ldots,T_t} : \{0,1\}^n \to \{0,1\}$ be the function defined as $f_{T_1,\ldots,T_t}(x) = 1$ if and only if there exists $i \in [t]$ such that $\Pr[T_i(x) = 1] \geq 2/3$. Consider the class of functions

$$\mathcal{F}_{s,t} = \left\{ f_{T_1,\ldots,T_t} \,:\, T_1,\ldots,T_t \in \mathsf{RDT}_s \right\}.$$

We show that $\mathcal{F}_{\mathsf{poly}(n)\cdot 2^q,2^p}$ satisfies the conditions of the lemma.

Let V be an MAP verifier of ε-proximity for Π_n that uses a proof of length p bits, q queries, and r random bits. The main observation is that for every fixed proof string $\pi \in \{0,1\}^p$, the (randomized) decision $V^x(n,\varepsilon,\pi)$ can be expressed as a randomized decision tree $T_{V,\pi}$ of depth $r+q$ (and size 2^{r+q}), which is defined as follows. The first r vertices in every path from the root to a leaf in the tree are labeled by $*$ (these vertices correspond to the random coin tosses of V). Every other internal vertex is labeled by some $i \in [n]$, corresponding to a query to x_i made by V. The two edges leaving every vertex, labeled by 0 and 1, correspond to the actual value of x_i, and these edges lead to a vertex that is labeled by the next query made by V, given the answer x_i to the query i. Given an input x and a random string $\rho \in \{0,1\}^r$, the leaf that is reached by evaluating the decision tree on input x and the random string ρ is labeled with the value $V^x(n,\varepsilon,\pi;\rho)$. (Recall that $V^x(n,\varepsilon,\pi;\rho)$ denotes the output of the verifier V given oracle access to x, direct access to n, ε, π and the random string ρ.) We are interested in $\Pr\left[V^x(n,\varepsilon,\pi) = 1\right]$.

Let $I_V : \{0,1\}^n \to \{0,1\}$ be defined as $I_V(x) = 1$ if and only if there exists $\pi \in \{0,1\}^p$ such that $\Pr[V^x(n,\varepsilon,\pi) = 1] \geq 2/3$. Since the randomized functions $V^x(n,\varepsilon,\pi)$ and $T_{V,\pi}(x)$ are identically distributed, it holds that $I_V \in \mathcal{F}_{2^{r+q},2^p}$.

By Lemma 2.17, we may assume without loss of generality that V has randomness complexity $r = O(\log n)$. The lemma follows by noting that $|\mathsf{RDT}_s| \leq (n+1)^s$ and therefore $|\mathcal{F}_{s,t}| \leq |\mathsf{RDT}_s|^t \leq (n+1)^{s\cdot t}$. □

Before proceeding to the proof of Theorem 2.22, we state a few standard propositions

(Propositions 2.19, 2.20 and 2.22) whose proofs are deferred to Section 2.8.2. We start by noting that sparse properties can be efficiently tested.

Proposition 2.19 (folklore)**.** *Every property $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ (where $\Pi_n \subseteq \{0,1\}^n$) can be tested by making $O(\log|\Pi_n|/\varepsilon)$ queries (without a proof).*

We note that Proposition 2.19 has standard proofs via learning theory techniques.[17] In Section 2.8.2 we provide an alternative proof that uses the notion of MAPs in a somewhat surprising, but very natural way.

The following (standard) proposition shows that, with high probability, a random n-bit string will be far from any small subset of $\{0,1\}^n$.

Proposition 2.20 (folklore)**.** *For every constant $\varepsilon \in (0,1/4]$ and set $S \subseteq \{0,1\}^n$, it holds that $\Pr_{x\in_R\{0,1\}^n}[x \text{ is } \varepsilon\text{-close to } S] \leq |S| \cdot 2^{-n/8}$.*

For the last claim that we need, recall the definition of a PRG.

Definition 2.21. *A set $S \subseteq \{0,1\}^n$ is called a* pseudorandom generator (PRG) *for fooling a class $\mathcal{F}$ of functions from $\{0,1\}^n$ to $\{0,1\}$ if for every $f \in \mathcal{F}$ it holds that*

$$\left|\Pr_{x\in_R S}[f(x)=1] - \Pr_{x\in_R\{0,1\}^n}[f(x)=1]\right| < 1/10.$$

(note that the choice of the constant $1/10$ is arbitrary.)

The following (well-known) lemma shows that for every class of functions $\mathcal{F}$, a random set of size $O(\log|\mathcal{F}|)$ is a PRG that fools $\mathcal{F}$.

Proposition 2.22 (implicit in [GK92], see also [Gol08, Exercise 8.1])**.** *Let $\mathcal{F}$ be a class of functions from $\{0,1\}^n$ to $\{0,1\}$, of size at most $2^{2^{n/4}}$. Then, 99% of subsets of $\{0,1\}^n$ of size $s = O(\log|\mathcal{F}|)$ are PRGs that fool $\mathcal{F}$.*

We are now ready to prove Theorem 2.22.

Proof of Theorem 2.22. Fix $\epsilon \in (0,1/4)$. Let $t,p,q : \mathbb{N} \to \mathbb{N}$ be functions such that $t = t(n) < n/10$, $p = p(n) \leq n$, $q = q(n) \leq n$ and $t = p + q + O(\log n)$.

Let $S \stackrel{\text{def}}{=} \cup_{n\in\mathbb{N}} S_n$ where for every $n \in \mathbb{N}$, the set $S_n \subseteq \{0,1\}^n$ is a random subset of $\{0,1\}^n$ of size $2^{t(n)}$. By Proposition 2.19, (for any choice of S) the property S can be tested using $O(\log(|S_n|)/\varepsilon) = O(t/\varepsilon)$ queries (without a proof).

Fix $n \in \mathbb{N}$ and let $\mathcal{F}_{p,q}^{(n)}$ be the class of functions of size $2^{(\mathsf{poly}(n)\cdot 2^{p+q})}$ guaranteed by Lemma 2.18, with respect to p and q. Since $O(\log|\mathcal{F}_{p,q}^{(n)}|) = O(2^{p+q}\cdot\mathsf{poly}(n)) = 2^t$, by Proposition 2.22 (applied to the class $\mathcal{F}_{p,q}^{(n)}$), with probability 0.99 over the choice of S_n, it holds that for every $f \in \mathcal{F}_{p,q}^{(n)}$:

$$\left|\Pr_{x\in_R S_n}[f(x)=1] - \Pr_{x\in_R\{0,1\}^n}[f(x)=1]\right| < 1/10. \tag{2.8}$$

[17] Either by an explicit reduction of property testing to learning (see [GGR98, Section 3]), or by applying Occam's razor directly to the testing problem.

Let S_n be a set for which Eq. (2.8) holds and assume toward a contradiction that there exists an MAP verifier V that uses a proof of length p and q queries, and tests ε-proximity to S_n.

By Lemma 2.18, it holds that $I_V \in \mathcal{F}^{(n)}_{p,q}$, where the function I_V is defined as $I_V(x) = 1$ if and only if there exists $\pi \in \{0,1\}^p$ such that $\Pr[V^x(n,\varepsilon,\pi)] \geq 2/3$. We proceed to show that I_V is a distinguisher for the PRG S_n, in contradiction to Eq. (2.8).

By the completeness of the MAP, for every $x \in S_n$ it holds that $I_V(x) = 1$ and therefore

$$\mathop{\mathbf{E}}_{x \in_R S_n}[I_V(x)] = 1.$$

On the other hand, by the soundness of the MAP, for every x that is ε-far from S_n it holds that $I_V(x) = 0$ and so

$$\mathop{\mathbf{E}}_{x \in_R \{0,1\}^n}[I_V(x)] \leq \mathop{\mathbf{E}}_{\substack{x \text{ that is} \\ \varepsilon\text{-far from } S_n}}[I_V(x)] + \Pr_{x \in_R \{0,1\}^n}\left[x \text{ is } \varepsilon\text{-close to } S_n\right] \leq |S_n| \cdot 2^{-n/8} \leq 2^{-\Omega(n)},$$

where the second inequality follows from Proposition 2.20 (and the fact that $I_V(x) = 0$ for every x that is ε-far from S_n), and the last inequality follows from our setting of $t \leq n/10$. Therefore,

$$\mathop{\mathbf{E}}_{x \in_R S_n}[I_V(x)] - \mathop{\mathbf{E}}_{x \in_R \{0,1\}^n}[I_V(x)] \geq 1 - 2^{-\Omega(n)},$$

in contradiction to Eq. (2.8). □

2.6 MAPs for Parametrized Concatenation Problems

In this section we give a scheme for constructing efficient MAPs for *parameterized concatenation problems*. For starters, we review the notion of (non-parameterized) concatenation problems: The k-concatenation problem of a property Π is defined as the property $\Pi^{\times k} \stackrel{\text{def}}{=} \{(x_1,\ldots,x_k) : \forall i \in [k],\ x_i \in \Pi \text{ and } |x_i| = |x_1|\}$. For every $i \in [k]$, we will refer to x_i as the i^{th} block or sub-input.

Concatenation problems (in the context of property testing) were recently studied by Goldreich [Gol14], who showed that the query complexity of the concatenation problem $\Pi^{\times k}$ (of a property Π) is roughly the same as the query complexity of the problem of testing a single instance of Π, regardless of the number of concatenations. More precisely, the query complexity of testing proximity of an input of length $n \cdot k$ (for $\Pi^{\times k}$) is the same, up to a polylogarithmic factor, as the query complexity of testing proximity of an input of length n (for Π), provided that the query complexity of Π increases at least linearly with $1/\varepsilon$ (which is typically the case).

We consider a generalization of the notion of a concatenation problem by allowing the underlying property to depend on some parameter, which may differ between the different blocks. Consider a family of properties $\{\Pi^\alpha\}_{\alpha \in A}$, where α is the parameter and A is some domain. As we shall show, some natural properties can be expressed as

a concatenation $\Pi^{\alpha_1} \times \ldots, \Pi^{\alpha_k}$ of a property Π^{α}, with respect to different values of the parameter. For example, testing whether a given string x has Hamming weight w can be expressed as the question of testing whether x can be partitioned into k blocks such that the i^{th} block has Hamming weight w_i and $\sum_{i\in[k]} w_i = w$. (Other natural examples are reviewed below.)

In this section it will be convenient for us to view the input length $n \in N$, the proximity parameter $\varepsilon \in (0,1)$, and the number of concatenations k as fixed. We note that although we fix n, ε, and k, these parameters should be viewed as generic, and so we allow ourselves to write asymptotic expressions such as $\mathsf{poly}(n)$, $\mathsf{poly}(\varepsilon)$, etc. If $\Pi \subseteq \{0,1\}^n$, then we say that a verifier V is an $\mathsf{MAP}(p,q)$ for Π with respect to proximity ε if V can distinguish between inputs that are in Π and inputs that are ε-far from Π using a proof of length p and q queries. (See the end of Section 2.6.1 for a discussion of the issues involved in providing a uniform treatment of parameterized concatenation problems.)

Additionally, throughout this section we study properties that are more naturally expressed as sets of strings (rather than functions), therefore we present them as such. Note that a function formulation (as in Definition 2.1) can be easily obtained by the (trivial) mapping that maps the string $x \in \Sigma^n$ to the function $f_x : [n] \to \Sigma$ defined as $f_x(i) = x_i$. We proceed to define parameterized concatenation problems.

Definition 2.23. *Let A be a finite set, and $n, k, n/k \in \mathbb{N}$. For every $\alpha \in A$, let $\Pi^{\alpha}_{n/k} \subseteq \{0,1\}^{n/k}$ be a property of n/k-bit strings that is parameterized by α. For every subset $\bar{A} \subseteq A^k$, we say that the property $\Pi^{\bar{A}}_n$ is a* parameterized k-concatenation property *(of n-bit strings), where $\Pi^{\bar{A}}_n$ is defined as*

$$\Pi^{\bar{A}}_n \stackrel{\text{def}}{=} \bigcup_{(\alpha_1,\ldots,\alpha_k)\in\bar{A}} \Pi^{\alpha_1}_{n/k} \times \ldots \times \Pi^{\alpha_k}_{n/k}.$$

If we consider the task of testing $\Pi^{\bar{A}}_n$, it is not a priori clear (for the tester) what value of the parameter α_i to use for each block. This is where MAPs can help us. That is, the proof of proximity will simply tell the MAP verifier the correct value of the parameter for each block. Using this idea, in Section 2.6.1 we construct an MAP for any parameterized concatenation problem. In Sections 2.6.2 to 2.6.3, we demonstrate the applicability of this technique by using it to construct efficient MAPs (which manage to bypass some lower bounds for testers that do not use a proof) for a couple of natural properties:

1. **Approximate Hamming weight:** The first application of our scheme is an efficient MAP for the problem of approximating the Hamming weight of a given string. In this problem, which is parameterized by $w \in [n]$, the tester needs to distinguish between inputs that have Hamming weight exactly w and those that have Hamming weight $\notin [w - \varepsilon n, w + \varepsilon n]$.

   We complement this MAP with a (non-tight) *lower bound* on the MAP complexity of the approximate Hamming weight property. We leave the question of resolving the gap between the upper and lower bounds to future work. See Section 2.6.2.

2. **Graph orientation problems:** In addition, we show an MAP in the graph orientation model (see Section 2.6.3 for details on this model). Specifically, our MAP distinguishes between orientations (of a specific undirected graph) that are Eulerian and those that are far from Eulerian. Our MAP has lower query complexity than the best possible property tester for this problem, and the gap in query complexity increases with the size of the proof. See Section 2.6.3.

Properties with/without distance. Note that all of the explicit properties studied in Section 2.3 are properties of low-degree polynomials and error-correcting codes. The MAPs that we have shown for these properties crucially relied on the fact that these properties have *distance* (i.e., properties wherein every two objects that have the property are far from each other), and moreover, they allow for a local form of self-correction.[18] We note that in contrast, all of the properties that we study in this section are without distance (as is the property of bipartiteness studied in Section 2.7). For example, the Hamming weight property is without distance since there are pairs of strings at distance 2 that have the same Hamming weight.

2.6.1 The Generic Scheme

In this section we show a generic scheme for parameterized concatenation problems.

Theorem 2.23. *Ler $c_1, c_2 \geq 0$ be constants. Let $\Pi_n^{\bar{A}}$ be a parameterized k-concatenation property (of n-bit strings) with respect to A, $\bar{A}$, and $\{\Pi_{n/k}^{\alpha}\}_{\alpha \in A}$, as in Definition 2.23. Suppose that for every $\alpha \in A$, the property $\Pi_{n/k}^{\alpha}$ can be tested with respect to any proximity parameter $\varepsilon' > 0$ (without using a proof) with query complexity $O\left((n/k)^{c_1} \cdot (\varepsilon')^{-c_2}\right)$. Then, the property Π has an MAP, with respect to proximity parameter ε, that uses a proof of length $k \cdot \log |A|$ and has query complexity:*

$$\begin{cases} \tilde{O}\left((n/k)^{c_1} \cdot \varepsilon^{-\max(1,c_2)}\right) & \text{if } c_1 > 0 \text{ and } c_2 \geq 0 \\ \tilde{O}\left((n/k)^{1-1/c_2} \cdot \varepsilon^{-1}\right) & \text{if } c_1 = 0 \text{ and } c_2 \geq 1. \end{cases}$$

Furthermore, if the testers for $\{\Pi_{n/k}^{\alpha}\}_{\alpha \in A}$ have a one-sided error, then the resulting MAP has a one-sided error.

Proof. The key idea is to use the proof in order to "break" the problem of testing property Π into the concatenation problem of testing several sub-properties with smaller inputs. Then, instead of solving each sub-problem independently, we efficiently verify that the (smaller) sub-inputs together are not too far from their corresponding sub-properties.

More specifically, we partition the input x (of length n) into k blocks $x_1, \ldots, x_k$ of length n/k each. If $x \in \Pi_n^{\bar{A}}$, then there must exist $(\alpha_1, \ldots, \alpha_k) \in \bar{A}$ such that $x_i \in \Pi_{n/k}^{\alpha_i}$ for each $i \in [k]$. The proof is simply $(\alpha_1, \ldots, \alpha_k)$; that is, the "hidden" parameter for each

[18] An important natural subset of this type of properties with distance is the set of properties of algebraic objects; see [KS08] for an extensive study of algebraic properties.

sub-property. The verifier, given this alleged proof, checks that indeed $(\alpha_1, \ldots, \alpha_k) \in \bar{A}$ (i.e., the parameterization of the sub-properties is valid), and is then left with the task of ascertaining that the k blocks are not "far" from $\Pi_{n/k}^{\alpha_1} \times \cdots \times \Pi_{n/k}^{\alpha_k}$.

Toward this end, similarly to the approach in [Gol14, Section 5], we note that given an input that is far from $\Pi_{n/k}^{\alpha_1} \times \cdots \times \Pi_{n/k}^{\alpha_k}$, the distance from the property can be either "spread" between all of the sub-inputs, or "concentrated" on a few sub-inputs — or anything in between. The main idea is that if the distance is "concentrated", then the deviation in these sub-inputs must be large, and so, we can detect that such particular sub-inputs do not have their corresponding sub-property by using a test with low query complexity. Since we only read a few bits for this test, we can afford to run it on many sub-inputs (thereby increasing our chance of catching a sub-input that is far from its corresponding sub-property). On the other hand, if the distance is "spread" among the sub-inputs, then it suffices to examine only a few sub-inputs, but for each such sub-input, we need to run a test with high query complexity. Interestingly, in the latter case it is sometimes beneficial for the verifier to simply read the entire block rather than to run the "expensive" tester.

Since the verifier does not know whether it is in one of the extreme situations or anywhere in between, naively we might want to consider the "worst of all worlds" (i.e., small spread and high query complexity per block). We improve upon the performance of the forgoing approach by using the precision sampling technique (originating in Levin [Lev85, last paragraph of Section 9], see also [Gol14, Appendix A.2]), which allows us to deal with all of the possible distributions of the distance economically (specifically, by considering only a logarithmic number of representative distributions). The resulting MAP protocol for parameterized concatenation problems is presented in Fig. 2.5.

Note that the length of the proof, which is $(\alpha_1, \ldots, \alpha_k)$, is bounded by $k \cdot \log |A|$. As for the query complexity, first recall that for any α and $\varepsilon' > 0$, the property $\Pi_{n/k}^{\alpha}$ has a tester with query complexity $T(n/k, \varepsilon') = (n/k)^{c_1} \cdot (\varepsilon')^{-c_2}$. Thus, the total number of queries is at most:

$$O\left(\sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \frac{\log(1/\varepsilon)}{2^j \varepsilon} \cdot \log(1/\varepsilon) \cdot T\left(n/k, 2^{-j}\right)\right) = \tilde{O}\left(\frac{(n/k)^{c_1}}{\varepsilon} \sum_{j \in [\lceil \log_2(2/\varepsilon) \rceil]} 2^{j(c_2-1)}\right)$$
$$= \tilde{O}\left((n/k)^{c_1} \varepsilon^{-\max(1,c_2)}\right).$$

For the special case in which $c_1 = 0$, we tighten the analysis. Observe that, without loss of generality, for any proximity parameter ε, it holds that $T(n, \varepsilon) \leq n$ (simply since the tester can always just read the entire input). Therefore, the query complexity is

MAP for the parameterized k-concatenation problem $\Pi_n^{\bar{A}}$

Input: a proximity parameter $\varepsilon > 0$ and oracle access to a string $x \in \{0,1\}^n$.

The Proof:

- The string x is interpreted as a k sub-inputs $x = (x_1, \ldots, x_k) \in (\{0,1\}^{n/k})^k$.
- The proof consists of the parameters for the concatenated problems; namely, the values $(\alpha_1, \ldots, \alpha_k)$ such that $x_i \in \Pi_{n/k}^{\alpha_i}$, for every $i \in [k]$ (such values must exist for $x \in \Pi_n^{\bar{A}}$).

The Verifier:

1. If $(\alpha_1, \ldots, \alpha_k) \notin \bar{A}$, then reject.
2. For every $j \in [\lceil \log_2(2/\varepsilon) \rceil]$, perform the following test:
   (a) Select uniformly at random $O\left(\frac{\log(1/\varepsilon)}{2^j \varepsilon}\right)$ indices in $[k]$. Denote the chosen indices by I.
   (b) For every $i \in I$: Run the $\Pi_{n/k}^{\alpha_i}$ tester $O(\log(1/\varepsilon))$ times on input x_i, with respect to proximity parameter 2^{-j}. Reject if the majority of the tests failed.
3. If all of the previous tests passed, then accept.

Figure 2.5: MAP for Π

bounded in this case by:

$$O\left(\sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \frac{\log(1/\varepsilon)}{2^j \varepsilon} \cdot \log(1/\varepsilon) \cdot T\left(n/k, 2^{-j}\right)\right) = \tilde{O}\left(\frac{1}{\varepsilon} \sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \min\left(\frac{n/k}{2^j}, 2^{j(c_2-1)}\right)\right)$$
$$\leq \tilde{O}\left(\frac{1}{\varepsilon} \sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} (n/k)^{1-1/c_2}\right),$$

where the last inequality follows from the fact that $c_2 \geq 1$ (by our assumption) and thus $\min\left(n/k \cdot 2^{-j}, 2^{(c_2-1)j}\right) \leq (n/k)^{1-1/c_2}$. Therefore, the total query complexity in this case is $\tilde{O}\left((n/k)^{1-1/c_2} \cdot \varepsilon^{-1}\right)$.

We proceed to prove the completeness and soundness of the protocol.

Completeness. Suppose that $x \in \Pi_n^{\bar{A}}$ and that $(x_1, \ldots, x_k) \in \Pi_{n/k}^{\alpha_1} \times \ldots \times \Pi_{n/k}^{\alpha_k}$. The tester for each sub-property is invoked $O(\log(1/\varepsilon))$ times in Step (2b) on some $x_i \in \Pi_{n/k}^{\alpha_i}$. Therefore, with probability $1 - \mathsf{poly}(\varepsilon)$ the majority of these invocations will accept. The total number of times that this step is run is at most $O(1/\varepsilon \cdot \log^2(1/\varepsilon))$ and therefore, by the union bound, the MAP verifier accepts with probability at least $2/3$.

Soundness. Suppose that $x \in \{0,1\}^n$ is ε-far from $\Pi_n^{\bar{A}}$. Let $(\alpha_1, \ldots, \alpha_k) \in \bar{A}$ be an alleged proof for the false statement $x \in \Pi_n^{\bar{A}}$ (notice that if $(\alpha_1, \ldots, \alpha_k) \notin \bar{A}$, then the tester immediately rejects). Thus, $x = (x_1, \ldots, x_k) \in (\{0,1\}^{n/k})^k$ is ε-far from $\Pi_{n/k}^{\alpha_1} \times \ldots \times \Pi_{n/k}^{\alpha_k}$ (since otherwise x is ε-close to $\Pi_n^{\bar{A}}$).

The following claim shows that it suffices to consider $O(\log(1/\varepsilon))$ different distributions of the distance between the sub-inputs. Since the proof of the claim is similar to results of [Gol14, Section 5], we defer it to Section 2.8.2.1).

Claim 2.23.1 (Precision Sampling (cf. [Lev85, last paragraph of Section 9] or [Gol14, Appendix A.2]))**.** *There exists* $j \in [\lceil \log_2 2/\varepsilon \rceil]$ *such that a* $\frac{2^j \varepsilon}{4 \cdot \lceil \log_2(2/\varepsilon) \rceil}$ *fraction of* $x_1, \ldots, x_k$ *are* 2^{-j}*-far from their corresponding sub-properties* $\Pi_{n/k}^{\alpha_1}, \ldots, \Pi_{n/k}^{\alpha_k}$.

Consider the execution of iteration j, where j is the index guaranteed by Claim 2.23.1. In this iteration, since the verifier selects uniformly at random $O\left(\frac{\log(1/\varepsilon)}{2^j \varepsilon}\right)$ indices in $[k]$, with probability at least 0.9, it selects at least one $i \in [k]$ such that x_i is 2^{-j}-far from Π^{α_i}.

Suppose that such an i is indeed selected. Since the base tester for $\Pi_i^{\alpha_i}$ is run with respect to proximity 2^{-j}, it will reject x_i with probability 2/3. Since the test is repeated $O(\log(1/\varepsilon))$ times, the majority of these tests will reject with probability at least 0.9. Thus, the MAP verifier rejects x with probability at least $0.9 \cdot 0.9 \geq 2/3$. □

On providing a uniform treatment. Recall that throughout this section we have fixed n, ε and k. Before proceeding to describe the applications of Theorem 2.23, we shortly discuss issues that arise when considering a uniform (asymptotic) treatment. In some cases, in order to optimize the total complexity (i.e., the sum of the proof complexity and the query complexity) of the MAP in Theorem 2.23, it is beneficial to allow the number k of concatenations to depend on the proximity parameter ε. However, if k depends on ε, then the following two issues arise.

First, notice that if k depends on ε, then the proof string in Theorem 2.23 becomes dependent on ε too, and therefore this protocol does not fall in our definition of MAP (Definition 2.1), which requires a single proof of proximity that works for *every* value of $\varepsilon > 0$. Hence, one can consider a slight relaxation of Definition 2.1 in which we allow the proof of proximity to depend on ε. Since formally such a protocol is not an MAP, we call it an $\mathsf{MAP}_{\mathsf{PDP}}$ (where PDP stands for *proximity dependent proofs*). Note that in an $\mathsf{MAP}_{\mathsf{PDP}}$ both the contents of the proof of proximity, *and its length* may depend on the proximity parameter. See Section 2.2.1 for further discussion of $\mathsf{MAP}_{\mathsf{PDP}}$.

An additional issue that arises when the number of concatenations k depends on ε is that it is unclear how to define a k-concatenation property, as the naive definition that follows Definition 2.23 would make the property itself depend on k, and therefore also on the proximity parameter. While this issue can be overcome for the specific properties that are studied below, doing so in general would be extremely cumbersome, which is the main reason for our non-uniform treatment.

2.6.2 Approximate Hamming Weight

In this section we consider the problem of deciding whether a given string $x \in \{0,1\}^n$ has Hamming weight approximately w. More specifically, we would like a tester that accepts every string $x \in \{0,1\}^n$ that has Hamming weight $w \in [n]$, and rejects strings that have Hamming weight that is ε-far from having weight w. Namely, the tester should reject every string $x \in \{0,1\}^n$ for which $\mathsf{wt}(x) \notin [w - \varepsilon n, w + \varepsilon n]$, where $\mathsf{wt}(x)$ denotes the Hamming weight of x.

More formally, we consider a family of properties $\{\mathsf{Hamming}_n^w\}_w$, indexed by a weight $w \in \{0, \ldots, n\}$. The property $\mathsf{Hamming}_n^w$ is defined as the set that consists of all strings $x \in \{0,1\}^n$ that have Hamming weight exactly w.

By well-known sampling lower bounds (see, e.g., [BYKS01, Theorem 15], improving upon [CEG95]), the query complexity of any property tester (which does not use a proof) is $\Omega\big(\min(n, \varepsilon^{-2})\big)$. Our goal is to use MAPs in order to bypass this lower bound. We remark that $\mathsf{Hamming}^w$ was already studied by [RVW13] who showed a multiple-message IPP for $\mathsf{Hamming}^w$ with complexity $\tilde{O}(\varepsilon^{-1})$ and a *2-message* IPP with complexity $\tilde{O}\left(n^{\frac{1}{3}} \cdot \varepsilon^{-\frac{2}{3}}\right)$. (Note that for $\varepsilon = 1/\sqrt{n}$, the 2-message protocol of [RVW13] has *sublinear* complexity of $\tilde{O}\left(n^{2/3}\right)$, whereas testing without a proof requires $\Omega(n)$ queries.)

Using Theorem 2.23, we show that the performance of the [RVW13] 2-message IPP can be matched by an MAP (i.e., a 1-message IPP), while essentially preserving its complexity.[19] Thus, we show that even a *non-interactive* proof suffices to bypass the property testing lower bound.

More generally, for every constant parameter $\alpha \in (0,1)$, we show that there exists an explicit MAP for Hamming that uses a proof of length $\tilde{O}(n^\alpha)$, and makes at most $\tilde{O}\left(\sqrt{n^{1-\alpha}} \cdot \varepsilon^{-1}\right)$ queries to the input string. For every value of $\alpha \in (0,1)$, there is a range of ε for which the MAP is more efficient than the best possible property tester (which does not use a proof) for Hamming. A comparison of the efficiency of our MAP versus standard property testers, for different values of α, is provided in Table 2.2.

Before we proceed, we note that we actually prove a slightly stronger result. Namely, that for every $k \in [n]$ there is an MAP for Hamming that uses a proof of length $k \cdot \log n$, and makes at most $\tilde{O}\left(\sqrt{n/k} \cdot \varepsilon^{-1}\right)$ queries (where the more restricted statement above is obtained by setting $k = n^\alpha$). In order to minimize the total complexity (i.e., the sum of the proof complexity and the query complexity) of the MAP, we also consider $\mathsf{MAP}_{\mathsf{PDP}}$ verifiers (recall that $\mathsf{MAP}_{\mathsf{PDP}}$ is a slight relaxation of our definition of MAP that allows the proof of proximity to depend on the proximity parameter, see the discussion at the end of Section 2.6.1. With this relaxation, we can set $k = n^{\frac{1}{3}} \cdot \varepsilon^{-\frac{2}{3}}$ to obtain an $\mathsf{MAP}_{\mathsf{PDP}}$ with (total) complexity $\tilde{O}\left(n^{\frac{1}{3}} \cdot \varepsilon^{-\frac{2}{3}}\right)$. See further discussion in Section 2.2.1.

We complement the foregoing upper bound by showing a *lower bound* on the MAP complexity of Hamming. Specifically, we show that every MAP for Hamming that uses

[19] We note that an MAP for approximating the Hamming distance with similar performance was also discovered independently by (Guy) Rothblum *et al.* following the initial publication of [RVW13].

		MAP	
Parameters	Property Testing	Proof Complexity	Query Complexity
General $\alpha \in (0,1)$	$\Theta\left(\min\left(n, \varepsilon^{-2}\right)\right)$	$\tilde{O}(n^{\alpha})$	$\tilde{O}\left(\sqrt{n^{1-\alpha}} \cdot \varepsilon^{-1}\right)$ Improves for $n^{-\frac{1}{2}-\frac{\alpha}{2}} < \varepsilon < n^{-\frac{1}{2}+\frac{\alpha}{2}}$
$\alpha = 0.02$	$\Theta\left(\min\left(n, \varepsilon^{-2}\right)\right)$	$\tilde{O}\left(n^{0.02}\right)$	$\tilde{O}\left(n^{0.49} \cdot \varepsilon^{-1}\right)$ Improves for $n^{-0.51} < \varepsilon < n^{-0.49}$
$\alpha = 2/3$	$\Theta\left(\min\left(n, \varepsilon^{-2}\right)\right)$	$\tilde{O}\left(n^{2/3}\right)$	$\tilde{O}\left(n^{1/6} \cdot \varepsilon^{-1}\right)$ Improves for $n^{-5/6} < \varepsilon < n^{-1/6}$
$\alpha = 0.98$	$\Theta\left(\min\left(n, \varepsilon^{-2}\right)\right)$	$\tilde{O}\left(n^{0.98}\right)$	$\tilde{O}\left(n^{0.01} \cdot \varepsilon^{-1}\right)$ Improves for $n^{-0.99} < \varepsilon < n^{-0.01}$

Table 2.2: The complexity of testing Hamming for different values of α.

a proof of length $p \geq 1$ must use $\Omega\left(\frac{\min(n,\varepsilon^{-2})}{p}\right)$ queries. Note that the two bounds do not match (e.g., for $\varepsilon = 1/\sqrt{n}$ and $p = n^{2/3}$, the upper bound is $\tilde{O}\left(n^{2/3}\right)$ and the lower bound is $\Omega(n^{1/3})$). We leave the question of resolving this gap for future work.

Theorem 2.24. *For every $w \in \{0, \ldots, n\}$, the property $\mathsf{Hamming}_n^w$ has a (two-sided error) MAP, with respect to proximity parameter ε, that uses a proof of length $k \cdot \log n$ and $\tilde{O}\left(\sqrt{n/k} \cdot \varepsilon^{-1}\right)$ queries.*

We remark that by applying Theorem 2.20 to the MAP of Theorem 2.24, we can (somewhat surprisingly) construct a *one-sided error* MAP with proof complexity $O(k \log n + \mathsf{polylog} n)$ and query complexity $\tilde{O}\left(\sqrt{n/k} \cdot \varepsilon^{-1}\right)$. In contrast, the query complexity of every one-sided error property tester for $\mathsf{Hamming}_n^w$ (without a proof) is *linear* in the input size.

Proof of Theorem 2.24. Fix $w \in [n]$. It is well-known (and easy to show, e.g., via the Chernoff bound) that ε-proximity to $\mathsf{Hamming}_n^w$ can be tested, without a proof, using $O(\varepsilon^{-2})$ queries (with a two-sided error). Let

$$\bar{A} \stackrel{\text{def}}{=} \left\{(w_1, \ldots, w_k) \in \{0, \ldots, n/k\}^k \; : \; \sum_{i=1}^{k} w_i = w\right\}.$$

Observe that a string $x = (x_1, \ldots, x_k) \in (\{0,1\}^{n/k})^k$ has Hamming weight w if and only if, for every $i \in [k]$ the string x_i has Hamming weight w_i and $\sum_{i=1}^{k} w_i = w$. Hence,

$$\mathsf{Hamming}_n^w = \bigcup_{(w_1,\ldots,w_k)\in\bar{A}} \mathsf{Hamming}_{n/k}^{w_1} \times \ldots \times \mathsf{Hamming}_{n/k}^{w_k}.$$

The theorem follows from Theorem 2.23 (where $c_1 = 0$ and $c_2 = 2$). □

Relation to TensorSum. The Hamming problem is loosely related to the *Sub-Tensor Sum problem* (see Section 2.3.2), since in both problems we want to compute the sum of the entries of a given input string. In the Sub-Tensor Problem we want an exact answer but are given the string in an error-corrected format (where we think of the input as $f : H^m \to \mathbb{F}$ which is encoded by a low degree polynomial $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on H^m). In the Hamming problem we do not have the benefit of an error-correcting code but allow an approximate answer.

Next, we show a lower bound on the MAP complexity of the property $\mathsf{Hamming}_n^{n/2}$ (the set of all strings of Hamming weight exactly $n/2$, where n is the length of the string). We note that the lower bound can be extended to $\mathsf{Hamming}_n^w$ for more general values of w by reducing to $\mathsf{Hamming}_n^{n/2}$ using adequate padding (while taking care of the integrality issues that arise). We also note that the lower bound only holds for reasonable complexity measures (which are specified formally below).

The lower bound is proved using our extension of the [BBM11] framework to the MAP model that was established in Section 2.3.2.2. Recall that this extension allows us to prove lower bounds on the complexity of MAPs via MA communication complexity lower bounds. We note that since an MAP lower bound refers to a particular value of ε, it immediately implies a lower bound also on $\mathsf{MAP}_{\mathsf{PDP}}$.

One natural candidate for a communication complexity problem on which we can base our Hamming lower bound is the *Hamming Distance* communication problem, wherein Alice and Bob need to decide whether the Hamming distance of their input strings is equal to a predetermined number. However, as opposed to the MAP lower bounds that we have shown before (e.g., for TensorSum, and EIM), Hamming is a property of non-robust objects; i.e., there is no significant distance between every pair of valid objects. In order to overcome the lack of distance between valid objects in Hamming, we wish to reduce Hamming to an MA communication complexity *gap*-problem wherein the YES-instances and NO-instances are far apart. Indeed, the *Gap Hamming Distance* problem, described next, serves this purpose.

Let $n \in \mathbb{N}$, and let $t, g > 0$. The Gap Hamming Distance problem, denoted by $\mathsf{GHD}_{n,t,g}$, is the promise problem wherein Alice gets as input an n-bit string x, Bob gets as input an n-bit string y, and the players need to decide whether the Hamming distance of their strings is greater than $t+g$ (considered a YES-instance), or smaller than $t-g$ (considered a NO-instance). See Section 2.8.2.2 for formal definitions and background. By extending a recent result of Gur and Raz [GR13b], we show

Lemma 2.24. *Let $g, n \in \mathbb{N}$ such that $g \leq n$ and $t = \alpha \cdot n$ for some constant $\alpha \in (0, 1)$. Then, every MA communication complexity protocol for $\mathsf{GHD}_{n,t,g}$, with proof complexity $p \geq 1$, has communication complexity at least $\Omega\left(\frac{\min\left(n,(n/g)^2\right)}{p}\right)$.*

The proof of Lemma 2.24, which is by a reduction to the result of [GR13b], is presented in Section 2.8.2.2 (see Corollary 2.33). Equipped with Lemma 2.24, we proceed to prove

the lower bound for $\mathsf{Hamming}_n^w$.

Theorem 2.25. *For every $n \in \mathbb{N}$ and $\varepsilon \stackrel{\text{def}}{=} \varepsilon(n) \in (0, 1/2)$, if $\mathsf{Hamming}_n^{n/2}$ has an MAP with respect to proximity parameter ε, with proof complexity $p = \Omega(\log n)$ and query complexity q such that $p(O(n)) = O(p(n))$ and $q(O(n)) = O(q(n))$, then $p \cdot q = \Omega\left(\min\left(n, \varepsilon^{-2}\right)\right)$.*

We note that our restriction on the form of p and q is satisfied by reasonable functions such as $f(n) = a \cdot n^b$ for any $a, b \geq 0$ as well as for $f(n) = a \cdot \mathsf{polylog}(n)$.

Proof of Theorem 2.25. Throughout the proof we fix the function w as $w(m) \stackrel{\text{def}}{=} m/2$. By Lemma 2.13, if $\mathsf{Hamming}_n^w \in \mathsf{MAP}(p, q)$, then the communication complexity (promise) problem $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ has an MA communication complexity protocol with a proof of length p and total communication $2q$, where (following [BBM11]) $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ refers to the communication complexity (promise) problem, in which Alice and Bob need to decide whether their inputs have Hamming distance exactly $n/2$ or are ε-far from having such distance. Thus, by Lemma 2.24, the theorem follows by reducing $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ to $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$, which is done next. (We stress that this reduction takes place entirely in the context of MA communication complexity.)

We note that both $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ and $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ are communication complexity (promise) problems that refer to the Hamming distance $\Delta(x, y)$ between the inputs x and y (of Alice and Bob, respectively). In $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ the YES-instances correspond to $\Delta(x, y) \geq n/2$ and the NO-instances correspond to $\Delta(x, y) \leq n/2 - 2\varepsilon n$, whereas in $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ the YES-instances correspond to $\Delta(x, y) = n/2$ and the NO-instances correspond to $\Delta(x, y) \notin [n/2 - \varepsilon n, n/2 + \varepsilon n]$.

We proceed to show a reduction from $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ to $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$. Since the reduction is between two MA communication complexity problems, we may allow the reduction to make use of a proof string. Specifically, the reduction is given as a proof string an integer $\tilde{d} \in \{0, \ldots, n\}$ that allegedly equals $\Delta(x, y)$, and maps a pair $(x, y) \in \{0,1\}^{n+n}$ to a pair $(x', y') \in \{0,1\}^{2n+2n}$ such that a YES (resp., NO) instance of $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ is mapped to a YES (resp., NO) instance of $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$.

The reduction, given input $\tilde{d}$ and (x, y), first checks that $\tilde{d} \geq n/2$ and rejects otherwise (since $\Delta(x, y) < n/2$ does not correspond to a YES instance of $\mathsf{GHD}_{n,n/2-\varepsilon,\varepsilon n}$). Then, the reduction maps the pair $(x, y) \in \{0,1\}^{n+n}$ to the pair $(x', y') \in \{0,1\}^{2n+2n}$ by setting $x' = x \circ 0^n$ and $y' = y \circ 0^{\tilde{d}} 1^{n-\tilde{d}}$. That is, Alice (resp., Bob), given input x (resp., y) and the alleged proof $\tilde{d}$, first checks that $\tilde{d} \geq n/2$ and then computes x' (resp., y'). The parties then run the $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ MA communication complexity protocol on input (x', y').

If (x, y) is a YES-instance of $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ (i.e., $\Delta(x, y) \geq n/2$) and $\tilde{d} = \Delta(x, y)$ (i.e., the provided proof is correct), then

$$\Delta(x', y') = \Delta(x, y) + n - \tilde{d} = n,$$

and so (x', y') is a YES-instance of $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$. On the other hand, if (x, y) is a NO-instance of $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ (i.e., $\Delta(x, y) \leq n/2 - 2\varepsilon n$), then for every $\tilde{d} \geq n/2$

$$\Delta(x', y') = \Delta(x, y) + n - \tilde{d} \leq n - 2\varepsilon n$$

and so (x', y') is a NO-instance of $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$.

Let us spell out how the reduction is used to prove the theorem. Suppose that $\mathsf{Hamming}_w$ is in the class $\mathsf{MAP}(p, q)$, where p and q are as in the hypothesis. Then, by Lemma 2.13, the $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$ problem has an MA communication complexity protocol with proof complexity p and communication complexity $2q$. Our reduction maps inputs of length n (of $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$) to inputs of length $2n$ (of $\mathcal{C}_{\oplus,\varepsilon}^{\mathsf{Hamming}_w}$), while using an additional proof of length $\log_2 n$. Thus, the reduction implies an MA communication complexity protocol for $\mathsf{GHD}_{n,n/2-\varepsilon n,\varepsilon n}$ with proof complexity $p(2n) + \log_2 n = O(p(n))$ and communication complexity $2q(2n) = O(q(n))$. Hence, by Lemma 2.24, it holds that $p \cdot q = \Omega(\min(n, \varepsilon^{-2}))$. □

2.6.3 Graph Orientation Problems

In this section we apply Theorem 2.23 to the problem of testing graph orientations for being Eulerian in the *graph orientation* model. In the graph orientation model, introduced by Halevy *et al.* [HLNT05], an underlying *directed* graph $G = (V, E)$ with a canonical orientation (i.e., wherein each edge is directed from the vertex with the smaller lexicographical order to the vertex with the larger lexicographical order) is given as an explicit input to the tester, and the actual input, to which the tester only has oracle access, is an orientation $\overrightarrow{G} = \{ d(e) \in \{0, 1\} : e \in E \}$ of G, wherein $d(e)$ represents the direction of the edge e.

Given a property Π_G (parameterized by the fixed directed graph G) of graph orientations, a tester for Π_G is given query access to an orientation of G; that is, every query is an edge $e \in E$, and the answer to the query is the direction of e in G (i.e., $d(e) \in \{0, 1\}$). An orientation $\overrightarrow{G}$ of G is ε-close to Π_G if it can be modified to be in Π_G by inverting the direction of at most an ε-fraction of the edges of G. Note that the distance function in the orientation model naturally depends on the size of the underlying graph. Moreover, the testing algorithm may strongly depend on the structure of the underlying graph. We note that the graph orientation model falls within the standard property testing framework, as a special case of property testing of *massively parameterized* problems (see [New10] for a survey on massively parameterized properties).

We consider the graph orientation property of being *Eulerian*, which was first pointed out by Halevy *et al.* [HLNT07] as a natural property for the graph orientation model. Recall that a directed graph is Eulerian if for every vertex v in the graph, the in-degree of v is equal to its out-degree. If G is a directed graph (with canonical orientation), we denote by Euler_G the property that contains all orientations of G to (directed) Eulerian graphs. While no (non-trivial) upper bound is known for this property, Fischer *et al.* [FLM+12] showed that for general graphs, testing proximity to being Eulerian with 1-*sided error* is

hard. Specifically, They showed that for $G = K_{2,n-2}$ (i.e., the full bipartite graph with 2 vertices on one side, and $n-2$ vertices on the other side), a one-sided error tester for Euler_G must use $\Omega(n)$ queries.

Using Theorem 2.23 we show, for every $\alpha \in (0,1]$, an MAP with 1-sided error for $\mathsf{Euler}_{K_{2,n-2}}$, which uses a proof of length $\tilde{O}(n^{\alpha})$ and $\tilde{O}(n^{1-\alpha}\varepsilon^{-1})$ queries. Hence, we have a smooth (up to poly-logarithmic factors) multiplicative trade-off between the query and proof complexities of the MAP. We note that it seems that using similar techniques, it is possible to obtain, using Theorem 2.23, efficient MAPs for several problems in the graph orientation model.

Formally, let $K_{2,n-2}$ be the graph with a set of vertices $V = \{ v_1, ..., v_n \}$ and a set of edges $E = \{ (v_i, v_j) \ : \ i \in \{1,2\},\ j \in \{ 3, ..., n \} \}$.

Theorem 2.26. *The property* $\mathsf{Euler}_{K_{2,n-2}}$ *has a one-sided error* MAP*, with respect to proximity parameter* ε*, that uses a proof of length* $O(k \cdot \log n)$ *and has query complexity* $\tilde{O}\left(\frac{n}{k} \cdot \varepsilon^{-1}\right)$.

Proof. The main idea is to divide $K_{2,n-2}$ into sub-graphs of equal size, wherein v_1 and v_2 are the only vertices that appear in all sub-graphs. We require that for all $j \in \{ 3, \ldots, n \}$, the in-degree of v_j is equal to its out-degree. However, since v_1 and v_2 appear in all of the sub-graphs, we can allow their in-degree in each subgraph to be different than their out-degree in this subgraph, as long as the sum of their in-degrees is equal to the sum of their out-degrees.

We denote the in-degree of a vertex $v \in K_{2,n-2}$ by $d_{in}(v)$ and the out-degree of $v \in K_{2,n-2}$ by $d_{out}(v)$. We start by considering the following generalization of the $\mathsf{Euler}_{K_{2,n-2}}$ property. For every $a, b \in \mathbb{Z}$, let $\mathsf{Euler}^{(a,b)}_{K_{2,n-2}}$ be the set of all orientations of $K_{2,n-2}$ such that:

1. $d_{in}(v_1) - d_{out}(v_1) = a$.
2. $d_{in}(v_2) - d_{out}(v_2) = b$
3. $d_{in}(v_j) = d_{out}(v_j)$, for all $j \in \{ 3, \ldots, n \}$.

(note that a and b may be negative). Let $\bar{A}$ be the set of all sequences $\big((a_1, b_1), \ldots, (a_k, b_k)\big)$, where $a_i, b_i \in \{-(n-2), \ldots, n-2\}$ for every $i \in [k]$ and for which it holds that $\sum_{i=1}^{k} a_i = 0$ and $\sum_{i=1}^{k} b_i = 0$. Consider the property:

$$\Pi \stackrel{\text{def}}{=} \bigcup_{(a_1,b_1),\ldots,(a_k,b_k) \in \bar{A}} \mathsf{Euler}^{(a_1,b_1)}_{K_{2,n/k-2}} \times \ldots \times \mathsf{Euler}^{(a_k,b_k)}_{K_{2,n/k-2}}.$$

This property contains all sequences of k orientations of the graphs $K_{2,n/k-2}$ such that (1) the vertices on the "large" side have in degree that is equal to their out degree and (2) for the vertices on the "small" sides, the sum, over all graphs, of their in-degree equals the sum of their out-degrees. We note that there is a trivial mapping between Π and $\mathsf{Euler}_{K_{2,n-2}}$ which simply identifies the pair of vertices on the smaller side of graphs in Π as a singe pair of vertices.

By applying Theorem 2.23 with $c_1 = 1$, $c_2 = 0$, and using the trivial tester (that queries the entire orientation) for every subgraph, the property Π has an MAP with proof of length $O(k \cdot \log n)$, and query complexity $\tilde{O}\left(\frac{n}{k} \cdot \varepsilon^{-1}\right)$. By the foregoing discussion, this MAP can be easily modified to work also for the property $\mathsf{Euler}_{K_{2,n-2}}$. □

2.7 Bipartiteness in Bounded Degree Graphs

In this section we consider the problem of testing *bipartiteness* for "rapidly-mixing" graphs in the bounded-degree graph model. In a classical result, Goldreich and Ron [GR99] showed that *any* graph can be tested for bipartiteness in the bounded-degree model, using a tester with query complexity $\tilde{O}(\sqrt{N}/\varepsilon)$, where N is the number of vertices in the tested graph. Goldreich and Ron first consider the (far simpler) case in which there is a promise that the graph is "rapidly-mixing" (see definition below). More recently, Rothblum, Vadhan and Wigderson [RVW13] showed a 2-message IPP for bipartiteness, in the rapidly-mixing case, with communication and query complexities that are $\mathsf{poly}(\log N, \varepsilon^{-1})$.

Roughly speaking, using similar techniques to (the rapidly-mixing case in) [GR99], we construct an MAP protocol for testing bipartiteness of rapidly-mixing graphs, with proof complexity p and query complexity q for every p and q such that $p \cdot q \geq N$. Thus, the query complexity of our MAP improves upon that of the [GR99] bipartiteness tester (which does not use a proof) only if the proof is of length $\omega(\sqrt{N})$. In particular, we obtain an MAP verifier that uses a proof of length $N^{2/3}$ and makes only $N^{1/3}$ queries. In contrast, a lower bound of $\Omega(\sqrt{N})$ for testers (which do not use a proof) was shown by Goldreich and Ron [GR02] (and this lower bound holds also in the rapidly-mixing case).

We leave the questions of (1) extending our result to graphs that are not rapidly-mixing, and (2) obtaining an MAP for bipartiteness with query and proof complexities that are both $o(\sqrt{N})$, for future research.

The Bounded Degree Graph Model. In the bounded degree graph model, introduced by Goldreich and Ron [GR02] (see also [Gol11a]), the object that is being tested is a graph $G = (V, E)$ with degree bounded by some constant d. The graph is represented by a function $g : V \times [d] \to V \cup \{\perp\}$ such that $g(u, i) = v$ if v is the i^{th} vertex incident at u, and $g(u, i) = \perp$ if u has less than i neighbors. The distance between two graphs, represented by functions $g, g' : V \times [d] \to V \cup \{\perp\}$ is measured (as usual) as the fraction of pairs (u, i) such that $g(u, i) \neq g'(u, i)$. For further details, see [Gol11a].

Rapidly-Mixing Graphs. Let $G = (V, E)$ be graph with degree bounded by d and let $N \stackrel{\text{def}}{=} |V|$. A (lazy) random walk of length ℓ starting at a vertex $s \in V$ is a random walk that involves ℓ steps. At each step, if the walk is currently at vertex v with degree $d_v \leq d$, then the walk continues to each neighbor of v with probability $1/2d$ and stays at v with probability $1 - \frac{d_v}{2d} \geq 1/2$ (a so-called "lazy" step). We say that G is rapidly-mixing if for every $s, t \in V$, the probability that a (lazy) random walk of length $\Omega(\log N)$ that starts in s ends in t, is at least $1/(2N)$ and at most $2/N$. We will use the fact that

in a rapidly-mixing graph $G = (V, E)$, for every vertex $s \in V$ and subset $T \subseteq V$, the probability that a random walk of length $\Omega(\log N)$ that starts at s ends in T, is at least $|T|/(2N)$ and at most $2|T|/N$. We mention the well-known fact that expander graphs are rapidly-mixing.

We proceed to describe our MAP. Actually since we require a promise that the graph is rapidly-mixing, we will need a "promise-problem" variant of the notion of MAP. For sake of brevity we only define this notion implicitly (in the next theorem).

Theorem 2.27. *There exists a probabilistic verifier V that given oracle access to a graph G of size N (in the bounded degree model), and explicit access to N, the degree bound d, a proximity parameter $\varepsilon \in (0, 1)$, and a proof string w of length $k \cdot \log N$, makes at most $\tilde{O}(\frac{N}{k} \cdot \varepsilon^{-2})$ oracle queries, and satisfies the following two conditions:*

1. *(Completeness:) if G is bipartite, then there exists a proof string $w \in \{0,1\}^{k \log N}$ such that $V^G(N, d, \varepsilon, w) = 1$, with probability 1.*

2. *(Soundness:) if G is* rapidly-mixing *and ε-far from every bipartite graph, then for every proof string w, with probability at least $1/2$, it holds that $V^G(N, d, \varepsilon, w) = 0$.*

Note that our tester has a one-sided error.

Proof. We define the parity of a (lazy) random walk as the parity of the number of actual (i.e., non-lazy) steps that take place in it. Loosely speaking, the proof that the graph G is bipartite is a subset $S \subseteq V$ of k vertices that are allegedly on the same side of G. To verify the proof, the verifier selects roughly $O(\log N)$ starting vertices, and takes approximately N/k random walks of length $O(\log N)$ from each starting vertex s. If there exist two random walks that start in s and end in S with different parities, then two corresponding vertices in S must be on different sides and the verifier rejects. Otherwise, the verifier accepts.

Since the graph is rapidly-mixing, the probability that a random walk that starts in s ends in S is roughly $|S|/N$. The key point (which is proved formally below) is that if the graph is far from bipartite, then for many starting vertices, the probability that the random walk ends in S *with parity* 0 (or equivalently, with parity 1) is $\Omega(|S|/N)$. That is, the probability of reaching S with either parity is significant enough. The protocol is presented in Fig. 2.6.

Note that the proof and query complexities are as stated. We proceed to show that completeness and soundness hold.

Completeness. If $G = ((L, R), E)$ is a bipartite graph such that $|L| \geq |R|$, and $S \subseteq L$ is the proof string, then there is no path between two vertices in S that has an odd length. Therefore, for every vertex $s \in V$, there are no two paths with different parities that end in S.

MAP for Bipartiteness of rapidly-mixing graphs (in the bounded degree graph model)

Input: oracle access to a graph $G = (V, E)$, the size $N \stackrel{\text{def}}{=} |V|$ of the graph, a bound d on the maximal degree in G, a proximity parameter $\varepsilon \in (0, 1)$, and a parameter $k \in [N]$.

The Proof:
Let $V = (L, R)$ such that L, R are disjoint independent sets and $|L| \geq |R|$ (such a partition is guaranteed if the graph is bipartite). The proof is an (arbitrary) subset $S \subseteq L$ of size k.

The Verifier:

1. Repeat $O\left(\frac{\log N}{\varepsilon}\right)$ times:
   (a) Select uniformly at random $s \in V$.
   (b) Take $O\left(\frac{N}{k} \cdot \frac{\log N}{\varepsilon}\right)$ (lazy) random walks starting at s, each of length $\ell \stackrel{\text{def}}{=} O(\log N)$.
   (c) Reject if there are two walks that end in S, having different parities.
2. If all of the previous tests passed, then accept.

Figure 2.6: MAP for Bipartiteness of rapidly-mixing graphs

Soundness. Suppose that $G = (V, E)$ is a rapidly-mixing graph of size $N = |V|$ that is ε-far from every bipartite graph and let $S \subseteq V$. For every $v \in V$ and $\sigma \in \{0, 1\}$, let p_v^σ be the probability that a (lazy) random walk of length $\ell = O(\log N)$ that starts at v, ends in S with parity σ. Since the graph is rapidly-mixing, $p_v^0 + p_v^1 \geq \frac{|S|}{2N}$ for every $v \in V$.

The following claim shows that, for an average vertex v, the probability that one random walk that starts at v ends in S with parity 0 *and* a second random walk that starts at v ends in S with parity 1, is roughly $\Omega((|S|/N)^2)$ (i.e., roughly the same as the probability for two random walks that start at v to end in S without any restriction on the parities of the walks).

Claim 2.24.1. $\sum_{v \in V} p_v^0 p_v^1 > \frac{\varepsilon |S|^2}{64 \ell N}$.

Proof. Suppose otherwise. Consider the following partition of the graph into (V_0, V_1) where $V_0 = \{v \in V : p_v^0 \geq p_v^1\}$ and $V_1 = \{v \in V : p_v^1 > p_v^0\}$. Let $E' = E(V_0, V_0) \cup E(V_1, V_1)$ be the set of all internal edges within V_0 and within V_1. We will obtain a contradiction by showing that G is ε-close to the bipartite graph $((V_0, V_1), E \backslash E')$ that is obtained from G by removing all edges in E'.

For every $v \in V$ and $\sigma \in \{0, 1\}$, let $A_{v,m}^\sigma$ denote the event that a (lazy) random walk of length m (where m is a parameter) that starts at v, ends in S with parity σ. In particular, $\Pr[A_{v,\ell}^\sigma] = p_v^\sigma$. Then, for every $\sigma \in \{0, 1\}$ and $v \in V_\sigma$, it holds that

$$p_v^{1-\sigma} \geq \sum_{u \in V_\sigma \text{ s.t. } (v,u) \in E'} \frac{1}{2d} \cdot \Pr[A_{u,\ell-1}^\sigma], \tag{2.9}$$

since a walk from v to S with parity $1-\sigma$ can be obtained by a step to one of the neighbors of v in V_σ (which happens with probability $1/2d$ for each neighbor), and a walk of length $\ell-1$ from this neighbor u to S with parity σ (i.e., the event $A^\sigma_{u,\ell-1}$).

Intuitively, since we expect the number of *lazy* steps in a lazy random walk to be rather large (at least $\ell/2$ in expectation), the probability that the event $A^\sigma_{u,\ell-1}$ occurs is closely related to the probability that the event $A^\sigma_{u,\ell}$ occurs (indeed, we expect the discrepancy in the number of steps to be "hidden" by the (deviation of the number of) lazy steps). The foregoing intuition is formalized by observing that with very high probability at least one lazy step occurs and the probability that $A^\sigma_{u,\ell}$ occurs, conditioned on a specific step being lazy, is equal to the probability that $A^\sigma_{u,\ell-1}$ occurs. Indeed, by the union bound,

$$\begin{aligned}
p^\sigma_u &= \Pr[A^\sigma_{u,\ell}] \\
&\leq \Pr[A^\sigma_{u,\ell} \wedge \text{ no lazy steps in the walk}] + \sum_{i\in[\ell]} \Pr[A^\sigma_{u,\ell} \wedge \text{ the } i^{\text{th}} \text{ step in the walk is lazy}] \\
&\leq \Pr[\text{no lazy steps in the walk}] + \sum_{i\in[\ell]} \Pr[A^\sigma_{u,\ell} \mid \text{the } i^{\text{th}} \text{ step in the walk is lazy}]
\end{aligned}$$

We can bound the first term by $2^{-\ell}$, which by setting $\ell = \log(4n)$, is at most $1/(4N)$. As for the second term, the probability that a random walk of length ℓ from u ends in S with parity σ *conditioned on the i^{th} step being lazy* is equal to the probability that a random walk of length $\ell-1$ from u ends in S with parity σ. Hence

$$p^\sigma_u \leq \frac{1}{4N} + \ell \cdot \Pr[A^\sigma_{u,\ell-1}] \tag{2.10}$$

Using Eq. (2.9) and Eq. (2.10), we obtain that:

$$\begin{aligned}
\sum_{v\in V} p^0_v p^1_v &= \sum_{\sigma\in\{0,1\}} \sum_{v\in V_\sigma} p^\sigma_v p^{1-\sigma}_v \\
&\geq \sum_{\sigma\in\{0,1\}} \sum_{\substack{(v,u)\in E' \\ \text{s.t. } v,u\in V_\sigma}} p^\sigma_v \cdot \frac{\Pr[A^\sigma_{u,\ell-1}]}{2d} \\
&\geq \sum_{\sigma\in\{0,1\}} \sum_{\substack{(v,u)\in E' \\ \text{s.t. } v,u\in V_\sigma}} p^\sigma_v \cdot \frac{1}{2\ell d} \cdot \left(p^\sigma_u - \frac{1}{4N}\right) \\
&\geq |E'| \cdot \frac{1}{2\ell d} \cdot \frac{|S|}{4N} \cdot \frac{|S|}{8N}
\end{aligned}$$

where the last inequality follows from the fact that for every $w \in V_\sigma$ it holds that $p^\sigma_w \geq (p^\sigma_w + p^{1-\sigma}_w)/2 \geq |S|/4n$.

Hence, by our hypothesis, $|E'| \leq \frac{\varepsilon|S|^2}{64\ell N} \cdot \left(\frac{1}{2\ell d} \cdot \frac{|S|}{4N} \cdot \frac{|S|}{8N}\right)^{-1} = \varepsilon dN$. Therefore, by removing an ε fraction of the edges of G we obtain a bipartite graph, in contradiction to our assumption that G is ε-far from bipartite. This concludes the proof of Claim 2.24.1. □

We say that a vertex v is good if $p_v^0 p_v^1 \geq \frac{\varepsilon|S|^2}{128\ell N^2}$. (Intuitively, a vertex v is good if two random walks that start at v are likely to end in S with different parities.) Let $\alpha \in [0,1]$ be the fraction of good vertices in V. By Claim 2.24.1,

$$\frac{\varepsilon|S|^2}{64\ell N} < \sum_{v\in V} p_v^0 p_v^1 = \sum_{v \text{ is good}} p_v^0 p_v^1 + \sum_{v \text{ is not good}} p_v^0 p_v^1 \leq \alpha N \cdot \left(\frac{2|S|}{N}\right)^2 + N \cdot \frac{\varepsilon|S|^2}{128\ell N^2},$$

where the last inequality uses the fact that for every vertex $v \in V$ it holds that $p_v^0 \cdot p_v^1 \leq (p_v^0 + p_v^1)^2 \leq (2|S|/N)^2$. Hence, the fraction of good vertices is at least $\alpha = \Omega(\varepsilon/\log N)$.

Hence, with probability at least 0.9, at least one of the starting vertices s (which were selected in one of the $O(\log N/\varepsilon)$ iterations) is good. Assume that indeed, in one of the iterations a good vertex s is selected. Hence, $p_s^0 p_s^1 \geq \frac{\varepsilon|S|^2}{128\ell N^2}$ and $p_s^0 + p_s^1 \leq \frac{2|S|}{N}$, which implies that $p_s^0, p_s^1 = \Omega\left(\frac{|S|\varepsilon}{N\log N}\right)$. Therefore, since we take $O\left(\frac{N}{|S|} \cdot \frac{\log N}{\varepsilon}\right)$ random walks starting in s, with probability 0.9, there will be at least one walk thats ends in S with parity 0 *and* one walk that ends in S with parity 1. Hence, the tester rejects with probability at least $0.9^2 \geq 1/2$. □

2.8 Appendices for Chapter 2

2.8.1 Background

2.8.1.1 Communication Complexity

Let X and Y be finite sets, and let $f : X \times Y \to \{0,1\}$ be a function. In the *two-party probabilistic communication complexity model* we have two computationally unbounded players, traditionally referred to as Alice and Bob. Both players share a random string. Alice gets as an input $x \in X$. Bob gets as an input $y \in Y$. At the beginning, neither one of the players has any information regarding the input of the other player. Their common goal is to compute the value of $f(x,y)$, while minimizing the communication between them. In each step of the protocol, one of the players sends one bit to the other player. This bit may depend on the player's input, the common random string, as well as on all previous bits communicated between the two players. At the end of the protocol, both players output $f(x,y)$ with high probability.

We say that a given protocol π computes a (possibly partial) function $f : X \times Y \to \{0,1\}$ if for every $x \in X$ and $y \in Y$ with probability at least $2/3$ Alice outputs $f(x,y)$ after interacting with Bob.[20] We define the communication complexity of the protocol $\mathsf{CC}(\pi)$ to be the maximum number of communicated bits in the protocol π when Alice and Bob are given inputs from X and Y respectively. The communication complexity of a function f is defined as:

$$\mathsf{CC}(f) = \min_{\pi \text{ that compute } f} \mathsf{CC}(\pi).$$

For a family of functions $\mathcal{F} = \{f_n : X_n \to Y_n\}_{n\in\mathbb{N}}$ we define the communication complexity of $\mathcal{F}$ as $\mathsf{CC}_n(\mathcal{F}) = \mathsf{CC}(f_n)$.

Set-Disjointness. The (unique) *set-disjointness* problem is the classical communication complexity problem wherein Alice gets an n-bit string x, Bob gets an n-bit string y, and their goal is to decide whether there exists $i \in [n]$ such that $x_i = y_i = 1$. Formally,

Definition 2.25. *For every $n \in \mathbb{N}$, $\mathsf{DISJ}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is the communication complexity predicate given by the partial function*

$$\mathsf{DISJ}_n(x,y) = \begin{cases} 1 & if \quad \sum_{i\in[n]} x_i y_i = 0 \\ 0 & if \quad \sum_{i\in[n]} x_i y_i = 1 \end{cases}$$

(where the arithmetic is over the integers).

It is well-known (see [KS92]) that the communication complexity of the *set-disjointness* problem is linear in the size of the inputs.

[20] In the case of a partial function, we consider only relevant x and y's.

2.8.1.2 MA Communication Complexity

In MA *communication complexity protocols*, we have a function $f : X \times Y \to \{0,1\}$ (for some finite sets X, Y), and three computationally unbounded parties: Merlin, Alice, and Bob. The function f is known to all parties. Alice gets as an input $x \in X$. Bob gets as an input $y \in Y$. Merlin sees both x, y but Alice and Bob share a private random string that Merlin cannot see.

At the beginning of an MA *communication complexity protocol*, Merlin, who sees both inputs x and y, sends a proof string $w = w(x, y)$ that asserts that $f(x, y) = 1$ to Alice and Bob. The two players exchanges messages and at the end of the protocol, (say) Alice outputs an answer $z \in \{0,1\}$. Note that the answer may depend on the proof w as well as the input (x, y). For a protocol π, denote by $\pi\big((x, y), w\big)$ the probabilistically generated answer $z \in \{0,1\}$ given by Alice on input (x, y) and proof w.

We define MA *communication complexity protocol* as follows.

Definition 2.26. *An* $\mathsf{MA}(c, p)$*-communication complexity protocol for* f *is probabilistic communication complexity protocol* π *between Alice and Bob in which they both get as input a* p*-bit proof, they can communicate at most* c *bits, and the protocol satisfies the following two conditions:*

1. Completeness*: for all* $(x, y) \in f^{-1}(1)$*, there exists a string* $w \in \{0,1\}^p$ *such that*

$$\Pr\left[\pi\big((x, y), w\big) = 1\right] \geq 2/3$$

   (where the probability is over the common random string).

2. Soundness*: for all* $(x, y) \in f^{-1}(0)$ *and for any string* $w \in \{0,1\}^p$ *we have*

$$\Pr\left[\pi\big((x, y), w\big) = 1\right] \leq 1/3$$

   (where the probability is over the common random string).

The MA Communication Complexity of Set-Disjointness. Recall that there is a well-known linear lower bound on the communication complexity of the the *set-disjointness* problem (DISJ) (see Section 2.3.1.3 for formal definitions and statement of the lower bound). A decade after the communication complexity of DISJ was settled, Klauck [Kla03, Kla11] showed the following lower bound on the MA communication complexity of set-disjointness (later proved to be tight, by Aaronson and Wigderson [AW09]).

Theorem 2.28. *Every* MA *communication complexity protocol for* DISJ_n *with proof complexity* p *and communication complexity* c *satisfies* $p \cdot c = \Omega(n)$.

2.8.1.3 Error Correcting Codes

We first introduce codes as objects of fixed length and then give asymptotic variants of the definitions. Let Σ be a finite alphabet. An error-correcting code (over Σ) is an injective function $C : \Sigma^k \to \Sigma^n$ where $k, n \in \mathbb{N}$ and $k < n$. Every element in the range of C is called a codeword. The stretch of the code is n (viewed as a function of k) and the relative distance is defined as d/n, where d is the minimal distance between two (distinct) codewords.

We say that the code C is a t-locally testable code (LTC), where $t : [0, 1] \to \mathbb{N}$, if there exists a probabilistic algorithm T that given oracle access to $w \in \Sigma^n$ and a proximity parameter $\varepsilon > 0$ makes at most $t(\varepsilon)$ queries. The algorithm accepts every codeword with probability 1, and rejects every string that is ε-far from the code with probability at least $1/2$. For further details on LTCs, see [GS06, Gol10c].

We say that the code C, with relative distance δ_0, is a t-locally decodable code (t-LDC), where $t \in \mathbb{N}$, if there exists a constant $\delta \in (0, \delta_0/2)$ called the decoding radius, and a probabilistic algorithm D that given $i \in [k]$ and oracle access to a string $w \in \{0, 1\}^n$ that is δ-close to a codeword $w' = C(m)$ for some $m \in \{0, 1\}^k$, makes at most t queries to the oracle and outputs m_i (i.e., the i^{th} bit of m) with probability at least $2/3$. Moreover, if w is a *codeword*, then the algorithm outputs m_i with probability 1. For further details on LDCs, see [KT00].

An important parameter of both LTCs and LDCs are their query complexities; that is, the number of queries t made to the string w. In both cases we are interested in codes for which the number of queries t is significantly smaller than n. While there are known LTCs with (almost) linear stretch and constant query complexity (i.e., t does not depend on n), obtaining an LDC with constant query complexity and polynomial stretch is a major open problem in coding theory.

We will also consider a relaxation of LDCs, introduced by Ben-Sasson *et al.* [BSGH+06], known as relaxed-LDC. In this variant, the decoder is allowed to abort on corrupted codewords. Indeed, the main advantage of relaxed-LDCs over standard LDCs is that there are known constructions (see [BSGH+06]) of relaxed-LDCs with constant query complexity and almost linear stretch.

Definition 2.27 (relaxed-LDC, adapted from [BSGH+06, Definition 4.5])**.** *We say that the code $C : \Sigma^k \to \Sigma^n$ with relative distance δ_0 is a t-*relaxed-LDC *if there exists a constant $\delta \in (0, \delta_0/2)$ and a probabilistic algorithm D that, given an integer $i \in [k]$ and oracle access to a string $w \in \Sigma^n$, makes at most t queries and satisfies the following two conditions:*

1. *If $w = C(m)$ is a codeword that encodes the message $m \in \{0, 1\}^k$, then D outputs m_i with probability 1.*
2. *If w is δ-close to a codeword $w' = C(m)$, then, with probability at least $2/3$, the decoder D outputs a value $\sigma \in \{m_i, \perp\}$; that is, $\Pr[D^w(i) \in \{m_i, \perp\}] \geq 2/3$.*

We note that our definition differs from the original definition in [BSGH+06] in two ways. The first difference is that [BSGH+06] require an additional, third, condition that we

do not need. (However, [BSGH+06] show that a code that satisfies conditions 1 and 2 above can be converted into an "equally good" code that satisfies also the additional third condition.) The second difference is that [BSGH+06] only require that the decoder succeed in decoding valid codewords with probability 2/3 whereas we require successful decoding with probability 1. Fortunately, the constructions of [BSGH+06] actually satisfy the stronger requirement.

The asymptotic variants of the foregoing definitions are obtained in the natural way by considering families of codes, one for each input length. Let $k : \mathbb{N} \to \mathbb{N}$ be some (sublinear) function an let $\{\Sigma_n\}_{n\in\mathbb{N}}$ be an ensemble of alphabets. A family of codes is an ensemble $\{C_n\}_{n\in\mathbb{N}}$ such that $C_n : (\Sigma_n)^{k(n)} \to (\Sigma_n)^n$ is a code for every $n \in \mathbb{N}$.

We say that the family of codes is a t-LTC for a function $t : \mathbb{N} \times [0,1] \to \mathbb{N}$ if for every $n \in \mathbb{N}$, the code C_n is a $t(n,\cdot)$-LTC. Similarly we say that a family of codes is a t-LDC (resp., relaxed-LDC) for a function $t : \mathbb{N} \to \mathbb{N}$ if for every $n \in \mathbb{N}$, the code C_n is a $t(n)$-LDC (resp., $t(n)$-relaxed-LDC). We sometimes abuse notation and refer to a family of codes as a single code.

2.8.1.4 Multivariate Polynomials and Low Degree Testing

In this section we recall some important facts on multivariate polynomials (see [Sud95] for a far more detailed introduction). In the following we fix a finite field $\mathbb{F}$ and a dimension m and consider m-variate polynomials over $\mathbb{F}$.

Lemma 2.28 (Schwartz-Zippel Lemma)**.** *Let $P : \mathbb{F}^m \to \mathbb{F}$ be a non-zero polynomial of total degree d. Let $S \subset \mathbb{F}$ and let $r_1, \ldots, r_m$ be selected uniformly at random in S. Then,*

$$\Pr_{r_1,\ldots,r_m \in_R S}[P(r_1,\ldots,r_m) = 0] \leq \frac{d}{|S|}.$$

An immediate corollary of the Schwartz-Zippel Lemma is that two distinct polynomials $P, Q : \mathbb{F}^m \to \mathbb{F}$ of total degree d may agree on at most a $\frac{d}{|\mathbb{F}|}$-fraction of their domain (i.e., $\mathbb{F}^m$).

Theorem 2.29 (Self-Correction Procedure (cf. [GS92, Sud95])**.** *Let $\delta < 1/3$, and $d, m \in \mathbb{N}$. There exists an algorithm that, given $x \in \mathbb{F}^m$ and oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial P' of individual degree d, makes $O(d \cdot m)$ oracle queries and outputs $P'(x)$ with probability 2/3. Furthermore, if P has total degree t, then given $x \in \mathbb{F}^m$, the algorithm outputs $P(x)$ with probability 1.*

In Theorem 2.29, as well as in the two following theorems, the error probability can be decreased to be an arbitrarily small constant using standard error reduction (while increasing the number of queries by a constant factor).

Theorem 2.30 (Total Degree Test (a.k.a. Low Degree Test) (see [RS96, Sud95, AS03])**.** *Let $\varepsilon \in (0, 1/2)$, $t, m \in \mathbb{N}$. There exists an algorithm that, given oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$, makes $O(t \cdot \mathsf{poly}(1/\varepsilon))$ queries and:*

1. *Accepts every function that is a polynomial of* total *degree t with probability* 1*; and*

2. *Rejects functions that are ε-far from every polynomial of total degree t with probability at least* $1/2$*.*

We will also need a more refined version of the test that tests the individual degree of the polynomial. Such a test is implicit in [GS06, Section 5.4.2] but for sake of self-containment we provide a full proof via a reduction to the total degree test.

Theorem 2.31 (Individual Degree Test)**.** *Let $d, m \in \mathbb{N}$ such that $dm < |\mathbb{F}|/10$ and $\varepsilon \in (0, 1 - \frac{dm}{|\mathbb{F}|})$. There exists an algorithm that, given oracle access to an m-variate polynomial $P : \mathbb{F}^m \to \mathbb{F}$, makes $O(dm \cdot \mathsf{poly}(1/\varepsilon))$ queries, and:*

1. *Accepts every function that is a polynomial of* individual *degree d with probability* 1*; and*

2. *Rejects functions that are ε-far from every polynomial of individual degree d with probability at least* $1/2$*.*

Proof. Given oracle access to the function P, the verifier T first runs the total degree test on P with respect to proximity ε and total degree dm. If the total degree verifier rejects, then T rejects.

If the test succeeds, then for every axis $i \in [m]$, the verifier T chooses at random $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_m \in_R \mathbb{F}$, and runs a univariate degree d test on the polynomial $Q_i(z) \stackrel{\text{def}}{=} P(r_1, \ldots, r_{i-1}, z, r_{i+1}, \ldots, r_m)$ with soundness error $1/10$. If for some axis i the univariate test rejects, then T rejects, otherwise it accepts.

Completeness. Completeness follow from the completeness of the total degree test together with the fact that the restriction of an individual degree d polynomial to any of its axes is a degree d univariate polynomial.

Soundness. Suppose that P is ε-far from every polynomial of individual degree d. If P is ε-far from every *total* degree dm polynomial, then the total degree test rejects with probability $1/2$. Thus, we focus on the case that P is ε-close to a total degree dm polynomial P'. In this case the polynomials P and P' are polynomials of total degree dm and since $\varepsilon < 1 - \frac{dm}{\mathbb{F}}$, by the Schwartz-Zippel lemma, they must be identical. Thus, P is a polynomial of total degree dm.

By the hypothesis, P cannot have individual degree d and therefore, there exists $i \in [m]$ such that $P(x_1, \ldots, x_m)$, as a formal polynomial, has degree $d' > d$ in x_i. Thus, there exist polynomials $P_0, \ldots, P_{d'}$ each of total degree at most dm such that

$$P(x_1, \ldots, x_m) = \sum_{j \in \{0, \ldots, d'\}} P_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) \cdot x_i^j$$

and $P_{d'} \not\equiv 0$.

Since $P_{d'}$ is a non-zero polynomial of total degree dm, by the Schwartz-Zippel lemma, it can vanish on only a $\frac{dm}{|\mathbb{F}|}$ fraction of its domain. Thus, when testing the i^{th} axis, with probability $1-\frac{dm}{|\mathbb{F}|}$, the verifier selects $r_1,\ldots,r_{i-1},r_{i+1},\ldots,r_m \in \mathbb{F}$ such that it guaranties that $P_{d'}(r_1,\ldots,r_{i-1},r_{i+1},\ldots,r_m)$ does not vanish. In this case, the polynomial $Q(z) \stackrel{\text{def}}{=} P(r_1,\ldots,r_{i-1},z,r_{i+1},\ldots,r_m)$ is a degree d' univariate polynomial and the verifier rejects it with probability 0.9. Thus, the verifier rejects with probability at least $0.9^2 > 1/2$. □

2.8.1.5 The Sum-Check Protocol

In this appendix we provide some background on the sum-check protocol that was first introduced by Lund *et al.* [LFKN92]. Recall that the sum-check protocol is an interactive proof for a statement of the form

$$\sum_{x_1,\ldots,x_m \in H} P(x_1,\ldots,x_m) = 0.$$

where P is a (relatively) low-degree polynomial over a finite field $\mathbb{F}$.

In order to verify that the polynomial P sums to 0 over H^m it suffices to verify that for every $h \in H$, the sum of the sub-tensor $(h,*,\ldots,*)$ equals some value $a_h \in \mathbb{F}$ and that $\sum_{h\in H} a_h = 0$. However, the straightforward recursion (which computes the sum of *every* sub-tensor) will yield a total query complexity of $\Omega(H^m)$.

The sum-check protocol takes a different approach by having the prover convince the verifier of the sum of just a *single* randomly selected sub-tensor (thus, yielding the desired efficiency). More specifically, the verifier asks the prover to specify the sum of all sum-tensors of the form $(z,*\ldots,*)$ for every $z \in \mathbb{F}$ (rather than $z \in H$). A key point is that these sums can be specified by the *low-degree* polynomial:

$$P_1(z) \stackrel{\text{def}}{=} \sum_{x_2,\ldots,x_m \in H} P(z,x_2,\ldots,x_m).$$

Since P_1 has low-degree, if the prover provides a different (low-degree) polynomial $\tilde{P}_1$, then these two polynomials must differ on almost all points in $\mathbb{F}$. Thus, it suffices for the verifier to select at random a point $r \in_R \mathbb{F}$ and to have the prover recursively prove that $\sum_{x_2,\ldots,x_m\in H} P(r_1,x_2,\ldots,x_m) = \tilde{P}_1(r_1)$. Hence, we reduced the m-dimensional TensorSum problem to an $(m-1)$-dimensional TensorSum problem using 2 messages and *no queries*. The recursion terminates when $m = 1$ in which case the verifier can verify the claim directly.

We note that when extending the sum-check protocol to be an IPP, we need to take into account the possibility that P is not low degree but this is handled by using the low degree test (Theorem 2.30) and self-correction (Theorem 2.29).

subsectionProofs and Adaptations of Known Results In this section we provide proofs and adaptations of known results, which are included here for completeness.

2.8.2 Proofs of Standard Claims from Section 2.5

In this section we provide the missing proofs of the standard claims used in Section 2.5.

Proof of Proposition 2.19. We show that every property $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ (where $\Pi_n \subseteq \{0,1\}^n$) can be tested by making $O(\log|\Pi_n|/\varepsilon)$ queries. Recall that the lemma can be proved via learning theory techniques, but we provide an alternative proof that makes use of the notion of MAPs.

Consider an MAP for Π in which the proof, of length $\log_2|\Pi_n|$, is an explicit and concise description of the object $x \in \Pi_n$ (e.g., its index with respect to the lexicographical ordering of the strings in Π_n). The verifier can verify the proof by querying the object x at $O(1/\varepsilon)$ locations uniformly at random (and compare the answers to the string reconstructed based on the proof). The lemma follows by noting that this MAP makes *proof-oblivious queries* and applying Theorem 2.19, which guarantees that if Π has an MAP verifier that makes q proof oblivious queries and uses a proof of length p, then Π has a tester that makes $O(p \cdot q)$ queries without using a proof. □

Proof of Proposition 2.20. We show that for every constant $\varepsilon \in (0, 1/4]$ and set $S \subseteq \{0,1\}^n$ it holds that $\Pr_{x\in_R\{0,1\}^n}[x \text{ is } \varepsilon\text{-close to } S] \leq |S| \cdot 2^{-n/8}$. Observe that

$$\begin{aligned}\Pr_{x\in_R\{0,1\}^n}[\exists s \in S \text{ such that } x \text{ is } \varepsilon\text{-close to } s] &\leq \sum_{s\in S} \Pr_{x\in_R\{0,1\}^n}[x \text{ is } \varepsilon\text{-close to } s] \\ &= |S| \cdot \Pr_{x\in_R\{0,1\}^n}[x \text{ has at most } \varepsilon n \text{ 1's}] \\ &\leq |S| \cdot \mathsf{exp}(-2 \cdot (1/4)^2 \cdot n).\end{aligned}$$

where the first inequality follows from the union bound, and the last inequality follows from the Chernoff bound and the fact that $\varepsilon < 1/4$. □

Proof of Proposition 2.22. Let $\mathcal{F}$ be a class of functions of size at most $2^{2^{n/4}}$. We show that 99% of sets of size $O(\log|\mathcal{F}|)$ are PRGs that fool $\mathcal{F}$.

For every set $S \subseteq \{0,1\}^n$ and function $f \in \mathcal{F}$, let $\delta_f(S) = |\Pr_{x\in_R S}[f(x) = 1] - \mu_f|$ where $\mu_f \stackrel{\text{def}}{=} \Pr_{x\in_R\{0,1\}^n}[f(x) = 1]$. Let $s \in [2^{n/4}]$ be an integer and let S be a random set of size s. Then, for every $f \in \mathcal{F}$ it holds that

$$\Pr_S[\delta_f(S) \geq 1/10] = \Pr_S\left[\left|\Pr_{x\in_R S}[f(x) = 1] - \mu_f\right| \geq 1/10\right] \leq 2^{-\Omega(t)},$$

where the last inequality follows from the Chernoff bound.[21] Thus, by the union bound, the probability that for *every* $f \in \mathcal{F}$ it holds that $\delta_f(S) < 1/10$, is at least $|\mathcal{F}| \cdot 2^{-\Omega(s)}$ (where the probability is over the choice of S). The lemma follows by setting $s = \Theta(\log|\mathbb{F}|)$. □

[21] We note that since the set S is chosen *with repetitions* one cannot directly apply the Chernoff bound. Still, since $s \leq 2^{n/4}$ the probability for a repetition is at most $s^2/2^n \leq 2^{-\Omega(n)}$. Conditioning on an event (i.e., that there are no repetitions) that occurs with probability $1-\delta$ can increase the probability by at most a $1/(1-\delta)$ factor.

2.8.2.1 Precision Sampling

Proof of Claim 2.23.1. We show that there exists $j \in [\lceil \log_2 2/\varepsilon \rceil]$ such that a $\frac{2^j \varepsilon}{4 \cdot \lceil \log_2(2/\varepsilon) \rceil}$ fraction of $x_1, \ldots, x_k$ are 2^{-j}-far from their corresponding sub-properties $\Pi^{\alpha_1}_{n/k}, \ldots, \Pi^{\alpha_k}_{n/k}$.

Let $d \stackrel{\text{def}}{=} \lceil \log_2(2/\varepsilon) \rceil$. Let $\Delta_{\mathsf{REL}}(z, W)$ be defined as the minimal *relative* Hamming distance of z from the set W. For every $j \in [d]$, let

$$S_j \stackrel{\text{def}}{=} \left\{ i \in [k] : \Delta_{\mathsf{REL}}\left(x_i, \Pi^{\alpha_i}_{n/k}\right) \in \left(2^{-j}, 2^{-(j-1)}\right] \right\},$$

and let $T = [k] \backslash (\cup_{i \in [d]} S_j)$. Notice that the sets $T, S_1, S_2, \ldots, S_d$ form a partition of the k inputs. Also note that, by our setting of d, for every $i \in T$, it holds that x_i is $\varepsilon/2$-close to $\Pi^{\alpha_i}_{n/k}$.

Suppose towards a contradiction that for every $j \in [d]$ it holds that $|S_j| < \frac{2^j \varepsilon}{4d} \cdot k$. Using the fact that for every $i \in S_j$ it holds that x_i is $2^{-(j-1)}$-close to Π^{α_i}, we get

$$\begin{aligned}
\Delta_{\mathsf{REL}}\left(x, \Pi^{\alpha_1}_{n/k} \times \ldots \times \Pi^{\alpha_k}_{n/k}\right) &\leq \frac{1}{k} \sum_{i=1}^{k} \Delta_{\mathsf{REL}}\left(x_i, \Pi^{\alpha_i}\right) \\
&= \frac{1}{k} \sum_{i \in T} \Delta_{\mathsf{REL}}\left(x_i, \Pi^{\alpha_i}\right) + \frac{1}{k} \sum_{j \in [d]} \sum_{i \in S_j} \Delta_{\mathsf{REL}}\left(x_i, \Pi^{\alpha_i}_{n/k}\right) \\
&\leq \frac{|T|}{k} \cdot \frac{\varepsilon}{2} + \frac{1}{k} \sum_{j \in [d]} 2^{-(j-1)} \cdot |S_j| \\
&< \frac{\varepsilon}{2} + \sum_{j \in [d]} \frac{\varepsilon}{2d} \\
&= \varepsilon,
\end{aligned}$$

contradicting our assumption that x is ε-far from $\Pi^{\bar{A}}$. □

2.8.2.2 Lower Bound on the MA Communication Complexity of GHD

Let $n \in \mathbb{N}$, and let $t, g > 0$. The Gap Hamming Distance problem is the promise problem wherein Alice gets as input an n-bit string x, Bob gets as input an n-bit string y, and the players need to decide whether the Hamming distance of their strings is greater than $t + g$ (a YES instance), or smaller than $t - g$ (a NO instance). Formally,

Definition 2.29. *The* Gap Hamming Distance *problem is the communication complexity problem of computing the (partial) Boolean function* $\mathsf{GHD}_{n,t,g} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ *given by*

$$\mathsf{GHD}_{n,t,g}(x, y) = \begin{cases} 1 & if \quad \Delta(x, y) \geq t + g \\ 0 & if \quad \Delta(x, y) \leq t - g \end{cases}.$$

We denote $\mathsf{GHD} \stackrel{\text{def}}{=} \mathsf{GHD}_{n, \frac{n}{2}, \sqrt{n}}$.

The (standard) communication complexity of GHD has been studied extensively, and after a long line of work, Chakrabarti and Regev [CR11] have shown the seminal linear lower bound on the communication complexity of GHD (later, the proof was significantly simplified by [Vid11, She11]).

In a subsequent work, Gur and Raz [GR13b] showed the following tight lower bound on the MA communication complexity of GHD.

Theorem 2.32 ([GR13b])**.** *Every* MA *communication complexity protocol for* GHD*, with proof complexity* $p \geq 1$*, has communication complexity at least* $\Omega(n/p)$*.*

We note that the aforementioned lower bound can be extended for general settings of the parameters of the *Gap Hamming Distance* problem. Specifically, we use the fact that the simple reductions in [CR11, Section 4]) are based solely on padding arguments (and thus are robust to MA) to obtain the following corollary.

Corollary 2.33. *Let* $g, n \in \mathbb{N}$ *such that* $g \leq n$*, let* $\alpha \in (0,1)$ *and* $t = \alpha n$*. Then, every* MA *communication complexity protocol for* $\mathsf{GHD}_{n,t,g}$*, with proof complexity* $p \geq 1$*, has communication complexity at least* $\Omega\left(\frac{\min\left(n,(n/g)^2\right)}{p}\right)$*.*

Chapter 3

Proofs of Proximity for Context-Free Languages and Read-Once Branching Programs

3.1 Introduction

The field of property testing, initiated by Rubinfeld and Sudan [RS96] and Goldreich, Goldwasser and Ron [GGR98], studies a computational model that consists of probabilistic algorithms, called *testers*, that need to decide whether a given object has a certain global property or is far (say, in Hamming distance) from all objects that have the property, based only on a local view of the object.

A line of work [EKR04, BSGH+06, DR06, RVW13, GR13b, FGL14, KR14] has considered the question of designing *proof systems* within the property testing model. The minimal type of such a proof system, which was recently studied by Gur and Rothblum [GR13b], augments the property testing framework by replacing the tester with a *verifier* that receives, in addition to oracle access to the input, also free access to an explicitly given short (i.e., sub-linear length) proof. The guarantee is that for inputs that have the property there exists a proof that makes the verifier accept with high probability, whereas, for inputs that are far from the property, the verifier will reject *every* alleged proof with high probability. These proof systems can be thought of as the NP (or more accurately MA) analogue of *property testing*, and are called *Merlin-Arthur proofs of proximity (*MAP*)*.[1]

A more general notion was considered by Rothblum, Vadhan and Wigderson [RVW13] (prior to [GR13b]). Their proof system, which can be thought of as the IP analogue of property testing, consists of an all powerful (but untrusted) prover who interacts with a verifier that only has oracle access to the input x. The prover tries to convince the verifier

[1] A related notion is that of a *probabilistically checkable proof of proximity (*PCPP*)* [BSGH+06, DR06]. PCPPs differ from MAPs in that the verifier is only given *query* (i.e., oracle) access to the proof, whereas in MAPs, the verifier has free (*explicit*) access to the proof. Hence, PCPPs are a PCP analogue of property testing.

that x has a particular property Π. Here, the guarantee is that for inputs in Π, there exists a prover strategy that will make the verifier accept with high probability, whereas for inputs that are far from Π, the verifier will reject with high probability no matter what prover strategy is employed. The latter proof systems are known as *interactive proofs of proximity (*IPP*s)*.[2]

The focus of this paper is identifying natural classes of properties that are known to be hard to test, but become easy to *verify* using the power of a proof (MAP) or interaction with a prover (IPP).

3.1.1 Our Results

One well-known class of properties that is hard to test is the class of *context-free languages*. Alon *et al.* [AKNS00] showed that there exists a context-free language that requires $\Omega(\sqrt{n})$ queries to test (where here and throughout this work, n denotes the size of the input) and a context-free language that requires $\Omega(n)$ queries to test with *one-sided error*. Furthermore, there are no known (non-trivial) testers for general context-free languages.

Another interesting class is the class of languages that are accepted by small *read-once branching programs (ROBPs)*. Newman [New02] showed that the set of strings accepted by any small width ROBP can be efficiently tested.[3] More specifically, Newman showed that width w ROBPs can be tested using $(2^w/\varepsilon)^{O(w)}$ queries, where ε is the proximity parameter. Bollig [Bol05] showed that Newman's result cannot be extended to polynomial-sized ROBPs, by exhibiting an $O(n^2)$-sized ROBP that requires $\Omega(\sqrt{n})$ queries to test. No (non-trivial) testers for general ROBPs are known for width $\Omega(\sqrt{\log n})$.

In this work we consider the question of constructing *efficient* MAPs and IPPs for these two classes.[4] Here, by "efficient", we mean that *both* the *query complexity* (i.e., the number of queries performed by the verifier to the input) and the *proof complexity* (i.e., the length of the MAP proof) or *communication complexity* (i.e., the amount of communication with the IPP prover) are small and, in particular, sub-linear[5].

Our first pair of results are efficient MAPs for context-free languages and for ROBPs. These MAPs offer a multiplicative trade-off between the query and proof complexities. Here and throughout this work, $n \in \mathbb{N}$ specifies the length of the main input and $\varepsilon \in (0,1)$ denotes the proximity parameter.

[2]Indeed, MAPs can be thought of as a restricted case of IPPs, in which the interaction is limited to a single message sent from the prover to the verifier.

[3]The result in [New02] is stated only for *oblivious* ROBPs but in [Bol05, Section 1.3] it is stated that Newman's result holds also for general *non-oblivious* ROBPs.

[4]To see that these two classes do not contain each other, observe that the language $\{0^i1^j2^i3^j : i,j \geq 1\}$, which is *not* a context-free language [HMU06, Example 7.20], has a $\mathsf{poly}(n)$-width ROBP (which simply counts the number of repeated occurrences of 0, 1, 2 and 3). On the other hand, Kriegal and Waack [KW88] showed that every ROBP for the Dyck_2 language, which is a context-free language, has size $2^{\Omega(n)}$.

[5]As pointed out in [GR13b], if we do not restrict the length of the proof, then *every* property Π can be verified trivially using only a constant amount of queries, by considering an MAP proof that contains a full description of the input.

Theorem 3.1. *For every context-free language $\mathcal{L}$ and every $k = k(n)$ such that $2 \leq k \leq n$, there exists an* MAP *for $\mathcal{L}$ that uses a proof of length $O(k \cdot \log n)$ and has query complexity $O\left(\frac{n}{k} \cdot \varepsilon^{-1}\right)$. Furthermore, the* MAP *has one-sided error.*

Theorem 3.2. *If a language $\mathcal{L}$ is recognized by a size $s = s(n)$ ROBP, then for every $k = k(n)$ such that $2 \leq k \leq n$, there exists an* MAP *for $\mathcal{L}$ that uses a proof of length $O(k \cdot \log s)$ and has query complexity $O\left(\frac{n}{k} \cdot \varepsilon^{-1}\right)$. Furthermore, the* MAP *has one-sided error.*

Hence, by setting $k = \sqrt{n}$, every context-free language and every language accepted by an ROBP of size at most $2^{\mathsf{polylog}(n)}$, has an MAP in which both the proof and query complexity are $\tilde{O}(\sqrt{n})$ (w.r.t. constant proximity parameter).

Next, we ask whether the query and proof complexity in Theorems 3.1 and 3.2 can be significantly reduced by allowing more extensive *interaction* between the verifier and the prover (i.e., arbitrary interactive communication rather than just a fixed non-interactive proof). Very relevant to this question is a recent result of [RVW13] by which, loosely speaking, every language in NC (which contains all context-free languages [Ruz81] and languages accepted by small ROBPs[6]) has an IPP with $\tilde{O}(\sqrt{n})$ query and communication complexities. While the [RVW13] result is more general, for context-free languages and ROBPs it achieves roughly the same query and communication complexities as the MAPs in Theorems 3.1 and 3.2, but uses much more interaction (i.e., at least logarithmically many rounds of interaction compared to just a single message in our MAPs).

Using cryptographic assumptions[7], Kalai and Rothblum [KR14] recently showed that there exists a language in NC_1 for which every IPP requires that either the query or communication complexity be $\Omega(\sqrt{n})$. Hence, we cannot hope to improve the [RVW13] result in general. Still, for the special case of context-free languages and ROBPs, we show that we can actually extend the MAP protocols in Theorems 3.1 and 3.2 into highly efficient IPPs with only *poly-logarithmic* complexity (using a sub-logarithmic number of rounds). More generally, our IPPs offer a trade-off between the number of rounds of interaction and the query and communication complexities.

Theorem 3.3. *For every context-free language $\mathcal{L}$, every $k = k(n) \geq 2$ and $r = r(n) \geq 1$ such that $k^r \leq n$, there exists an r-round* IPP *for $\mathcal{L}$ with communication complexity $O\left((rk \log n) \cdot \varepsilon^{-1}\right)$ and query complexity $O\left(\frac{n}{k^r} \cdot \varepsilon^{-1}\right)$. Furthermore, the* IPP *is public-coin and has one-sided error.*

Theorem 3.4. *If a language $\mathcal{L}$ is recognized by a size $s = s(n)$ ROBP, then for every $k = k(n) \geq 2$ and $r = r(n) \geq 1$ such that $k^r \leq n$, there exists an r-round* IPP *for $\mathcal{L}$ with communication complexity $O\left((rk \log s) \cdot \varepsilon^{-1}\right)$ and query complexity $O\left(\frac{n}{k^r} \cdot \varepsilon^{-1}\right)$. Furthermore, the* IPP *is public-coin and has one-sided error.*

[6] See Section 3.5.2 for a discussion on why languages accepted by ROBPs can be computed in small depth.

[7] A sufficient assumption for [KR14] is the existence of (length-doubling) PRGs that can be computed in NC_1 and whose output cannot be distinguished from random by circuits of size $2^{o(n)}$.

(Interestingly, and in contrast to Theorems 3.1 and 3.2, here the communication complexity also depends on the proximity parameter ε.) In particular, by setting $k = \log n$ and $r = \frac{\log n}{\log \log n}$, we obtain IPPs for context-free languages and size $2^{\mathsf{polylog}(n)}$ ROBPs, with a sub-logarithmic number of rounds, constant query complexity, and poly-logarithmic communication complexity (w.r.t. constant proximity parameter).

A Remark on Computational Complexity. Following the property testing literature, we view the query complexity and the proof complexity (resp., communication complexity) as the primary resources of an MAP (resp., IPP). Still, the running time of the verifier and of the prover are also important resources. The proofs/provers in our MAPs and IPPs are indeed efficient; that is, polynomial in the main input x (and in the case of ROBPs also in the size of the ROBP).

As for our verifiers, those in Theorems 3.1 and 3.3 run in polynomial time (i.e., $\mathsf{poly}(|x|)$ time) rather than in *sub-linear* time as one might hope. However, by increasing the round complexity in Theorem 3.3 by a poly-logarithmic factor, we can obtain an IPP with sub-linear time verification. Constructing an MAP for context-free languages with sub-linear time verification remains an interesting open question. The verifiers in Theorems 3.2 and 3.4 run in *sub-linear time* if they are given a *suitable* (natural) representation of the ROBP.[8] See the technical sections (specifically Remark 3.7 and Remark 3.16) for further details.

Improved Results for Specific Languages. The paradigm used for the general results in Theorems 3.1-3.4 can be extended to yield better results for specific languages. A notable class of languages for which we obtain such an improvement is the class of languages of balanced parentheses expressions (a.k.a the Dyck languages), which are context-free languages, for which Parnas *et al.* [PRR01] showed a lower bound of $\widetilde{\Omega}(n^{1/11})$ for ordinary testers. Using special properties of the Dyck languages, we can improve on the general result in Theorem 3.1 in this special case and obtain a somewhat more efficient MAP for the Dyck languages. See details in Section 3.4.3.

3.1.2 Proof Overview

The proofs of Theorems 3.1 and 3.2 (i.e., the MAP results) will follow (roughly) as special cases of the proofs of Theorems 3.3 and 3.4 (i.e., the IPP results), respectively. Hence, in this overview we focus on the proofs of Theorems 3.3 and 3.4, while explaining how to derive Theorems 3.1 and 3.2 as special cases.

The proofs of Theorems 3.3 and 3.4 share a common theme: For $\mathcal{L}$ that is either a context-free language or is accepted by a ROBP, we show that every input $x \in \mathcal{L}$ can be broken-down into k sub-problems (related to $\mathcal{L}$) such that the following holds:

[8] Indeed, the running time of the verifier crucially relies on the specific representation of the ROBP. We remark that there are other natural representations of ROBPs than the one we use, and for some of these representations obtaining sub-linear running time may not be feasible.

1. On the one hand, if $x \in \mathcal{L}$, then there exists (1) a partition of $[n]$ into sets $S_1, \ldots, S_k$ (each of size roughly n/k); and (2) languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ such that both (1) and (2) have a concise representation, and, for every $i \in [k]$, the projection of x on S_i, denoted $x[S_i]$, is in the language $\mathcal{L}_i$. Furthermore, if $\mathcal{L}$ is a context-free language (resp., accepted by an ROBP), then the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are all "variants" of context-free languages[9] (resp., accepted by ROBPs).

2. On the other hand, if x is "far" from $\mathcal{L}$, then for every concise representation of a partition $S_1, \ldots, S_k$ of $[n]$ and languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ (of the type used in 1), for an average $i \in [k]$, it holds that $x[S_i]$ is proportionally "far" from $\mathcal{L}_i$.

By design, the partition $S_1, \ldots, S_k$ as well as the corresponding languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ depend on the entire input x, and so the verifier (who only has query access to x) cannot generate them by itself. Instead, the concise representation of $S_1, \ldots, S_k$ and $\mathcal{L}_1, \ldots, \mathcal{L}_k$ will be specified by the prover (as a single message in the case of an IPP, or as the entire proof string in the case of an MAP).

Given the latter, we construct an MAP as follows. The MAP verifier selects at random a small subset $I \subseteq [k]$ and, for every $i \in I$, reads *all* of $x[S_i]$ (which is of length roughly n/k) and checks that $x[S_i] \in \mathcal{L}_i$. Indeed, by the two foregoing conditions, if $x \in \mathcal{L}$, then $x[S_i] \in \mathcal{L}_i$ for every $i \in [k]$, whereas if x is "far" from $\mathcal{L}$, then, by an averaging argument, for many $i \in [k]$, it holds that $x[S_i]$ is proportionally "far" from $\mathcal{L}_i$ (and in particular $x[S_i] \notin \mathcal{L}_i$), and the verifier will reject.

A natural approach for extending the foregoing MAP to an IPP is to have the verifier send the set I (where I is chosen at random as in the MAP) to the prover, and then *recursively* run $|I|$ IPP protocols to check that $x[S_i]$ is close to $\mathcal{L}_i$, for every $i \in I$. In each recursive call the input shrinks by (roughly) a factor of k. After the recursion reaches depth r, where r is a predetermined bound on the number of rounds, the verifier can simply read its entire current input (of length $O(n/k^r)$) and decide whether to accept or reject.

The foregoing approach indeed works, but because there is more than one recursive call in each round, the complexity of the resulting IPP depends *exponentially* on the number of rounds r. Instead, we use a more economical approach, which avoids the exponential dependence on r, based on the notion of a *proximity oblivious tester* [GR11]. Recall that a proximity oblivious tester for a property Π is a tester that does not receive the proximity parameter ε as input and is only required to reject inputs that are ε-far from Π with probability proportional to ε (rather than probability 2/3). To present a more economical recursion, the IPP that we design is similarly "proximity oblivious". The idea is to have the verifier select at random only a single index $i \in [k]$, send i to the prover, and then have the two parties recursively run an IPP protocol for verifying that $x[S_i]$ is close to $\mathcal{L}_i$. Indeed, if $x \in \mathcal{L}$ then $x[S_i] \in \mathcal{L}_i$, whereas if x is ε-far from $\mathcal{L}$, then, since i was chosen at random, on the average $x[S_i]$ is ε-far from $\mathcal{L}_i$, and therefore, by inductive

[9] If $\mathcal{L}$ is a context-free language, then the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ will be variants of context-free languages, which we call "partial derivation languages". However, if $\mathcal{L}$ is accepted by an ROBP, then the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are also accepted by (different) ROBPs.

reasoning, the verifier will reject with probability ε. To obtain constant soundness we can just repeat[10] the entire proximity oblivious protocol $O(1/\varepsilon)$ times in parallel.

This concludes the high-level description of our MAPs and IPPs. Of course, the way in which the partition is generated is quite different in the case of context-free languages and in the case of ROBP, and different technical problems arise in each case. In the following subsections we discuss the specific details. In Section 3.1.2.1 we give an overview of how to partition read-once branching programs. Partitioning context-free languages is more involved, and so, in Section 3.1.2.2, as a warm-up, we first consider partitioning into *two parts* (i.e., $k = 2$). Then, in Section 3.1.2.3 we show how to extend the technique to *multiple parts* (i.e., general $k \geq 2$).

3.1.2.1 Partitioning ROBPs

Recall that a branching program on n variables is a directed acyclic graph with a unique source vertex with in-degree 0 and (possibly) multiple sink vertices with out-degree 0. Each sink vertex is labeled with either 0 (i.e., *reject*) or 1 (i.e., *accept*). Each non-sink vertex is labeled by an index $i \in [n]$ and has exactly 2 outgoing edges, which are labeled by 0 and 1. The output of the branching program B on input $x \in \{0,1\}^n$, denoted $B(x)$, is computed in a natural way by starting at the source vertex and taking a walk such that at a vertex labeled by $i \in [n]$, we traverse the outgoing edge labeled by x_i. Once a sink is reached, we output its label. The branching program is *read-once* (ROBP for short) if along every path from source to sink, every index ($i \in [n]$) appears at most once. The *size* of a branching program B, denoted $|B|$, is the number of vertices in it.

For any fixed ROBP B, we construct an IPP (and an MAP, which is a special case of the IPP) for the language accepted by B, denoted $\mathcal{L}_B \stackrel{\text{def}}{=} \{x \in \{0,1\}^n : \ B(x) = 1\}$. In this overview, we make a simplifying assumption that B is both *layered* and *ordered* (a.k.a., an *ordered binary decision diagram* or OBDD). That is, we assume that the vertices of B are partitioned into $n+1$ layers such that, for every $i \in [n]$, edges only go from layer i to layer $i+1$; and vertices in layer i are labeled by the index i (i.e., the ROBP reads its input "in order").

The key idea, which enables the IPP verifier to generate the aforementioned partition $S_1, \ldots, S_k$ (together with the corresponding languages), is to have the prover specify k evenly-spaced vertices along the accepting path corresponding to the input $x \in \mathcal{L}_B$. More specifically, observe that x induces a path $\varphi_0 \to \varphi_1 \to \cdots \to \varphi_n$ from the start vertex φ_0 to some accepting sink φ_n. The prover sends to the verifier a subsequence of this walk, specifically the subsequence $\varphi_{n/k}, \ldots, \varphi_{i \cdot n/k}, \ldots, \varphi_n$.

Given the subsequence, we can reduce the problem of verifying that there exists a path of length n from φ_0 to φ_n to verifying that there exists a path of length n/k between each pair of consecutive vertices in the sequence $\varphi_0, \varphi_{n/k}, \ldots, \varphi_{i \cdot n/k}, \ldots, \varphi_n$. In other words, for every $i \in [k]$ we consider the ROBP B_i that consists only of layers $(i-1) \cdot n/k$ up to $i \cdot n/k$ of B, with the starting state $\varphi_{(i-1) \cdot n/k}$ and the (only) accepting state $\varphi_{i \cdot n/k}$. Verifying

[10] As expected, parallel repetition reduces the soundness error of IPPs at an exponential rate. See Section 3.5.1 for details.

that $x \in \mathcal{L}_B$ can be reduced to verifying that $x[S_i] \in \mathcal{L}_{B_i}$, for every $i \in [k]$, where $S_i \subseteq [n]$ is the set of coordinates of x that are read by B_i and $\mathcal{L}_{B_i} \stackrel{\text{def}}{=} \{z \in \{0,1\}^{n/k} : B_i(z) = 1\}$. Moreover, since $S_1, \ldots, S_k$ is a partition of $[n]$, if x is ε-far from $\mathcal{L}_B$, then $x[S_i]$ is ε-far from $\mathcal{L}_{B_i}$, for an average $i \in [k]$. Hence, we can follow the high-level outline that was suggested in Section 3.1.2; that is, the IPP verifier selects $i \in [k]$ at random, sends i to the prover, and then the two parties recursively run an IPP protocol to verify that $x[S_i]$ is close to the $\mathcal{L}_{B_i}$.

The foregoing intuition almost works but there is a subtle problem: What if the message sent by a *cheating* prover is such that $\mathcal{L}_{B_{i^*}}$ is empty, for some $i^* \in [k]$. This corresponds to a situation in which the branching program B contains no path from $\varphi_{(i^*-1)\cdot n/k}$ to $\varphi_{i^* \cdot n/k}$. In such case, with high probability (i.e., if the verifier chooses i such that $i \neq i^*$) the verifier, as described so far, will not notice this fact and may accept inputs that are far from $\mathcal{L}_B$.

We overcome this difficulty by observing that when the verifier interacts with the honest prover, it holds that $x[S_i] \in \mathcal{L}_{B_i}$ for every $i \in [k]$, and therefore $\mathcal{L}_{B_i} \neq \emptyset$. Hence, we can have the verifier explicitly check that $\mathcal{L}_{B_i} \neq \emptyset$ for *every* $i \in [k]$ (i.e., that there exists *some* input that leads from $\varphi_{(i-1)\cdot n/k}$ to $\varphi_{i \cdot n/k}$ in B). This check requires direct and full access to the branching program B (which is fixed) but does *not* require any queries to the input x, and so we can perform it for *every*[11] $i \in [k]$.

Given this additional check, we can show that the foregoing IPP works. To do so, we argue by induction on the number of rounds that if the input x is ε-far from $\mathcal{L}$ then the verifier rejects with probability at least ε. Indeed, if x is ε-far from $\mathcal{L}_B$, then in the first round we have that:

$$\begin{aligned}
\Pr\left[\text{Verifier for } \mathcal{L}_B \text{ rejects } x\right] &= \mathop{\mathbf{E}}_{i}\left[\Pr\left[\text{Verifier for } \mathcal{L}_{B_i} \text{ rejects } x[S_i]\right]\right] \\
&\geq \mathop{\mathbf{E}}_{i}\left[\varepsilon_i\right] \\
&\geq \varepsilon,
\end{aligned}$$

where ε_i denotes the relative distance of $x[S_i]$ from $\mathcal{L}_{B_i}$, for every $i \in [k]$, and the first inequality follows from the induction hypothesis.

We remark that when dealing with general ROBPs, rather than OBDDs, there are several additional technical difficulties. In particular, since B is not layered, we have to modify our definition of B_i (which previously consisted of layers $(i-1) \cdot n/k$ to $i \cdot n/k$ of B). A natural approach is to define B_i to consist of all paths (in B) of length n/k starting at $\varphi_{(i-1)\cdot n/k}$.[12] The difficulty is that B_i may depend on many, possibly even all, of the bits of x (since different paths may look at different bits), rather than just n/k bits (as was the case for OBDDs). Hence, the input does not necessarily shrink in the recursive step. Nevertheless, we resolve this issue by showing that the *effective length* of

[11] However, this check does increase the running time of the verifier (which we view as a secondary resource) to $\mathsf{poly}(|B|)$. This computation can be minimized by using a pre-processing step in which we compute a $|B| \times |B|$-sized table whose $(v,u)^{\text{th}}$ entry says whether the vertices v and u are connected in B.

[12] The actual definition of B_i that we use is different. See Section 3.3 (in particular Footnote 18).

the input, which is the number of bits that need to be read in order to determine whether the ROBP accepts, does shrink, and this suffices to make progress in the recursion. For further details, see Section 3.3.

3.1.2.2 Partitioning Context-Free Languages into Two Parts

Recall that a *context-free grammar* is a tuple $G = (V, \Sigma, R, A_{\mathsf{start}})$, where $V = \{A_1, A_2, \dots\}$ denotes a (finite) set of variables, $\Sigma = \{\sigma_1, \sigma_2, \dots\}$ denotes a (finite) set of terminal symbols (i.e., the alphabet), R is a set of production rules (e.g., rules of the form $A_7 \to \sigma_5 A_3 A_9 \sigma_8 A_2$) and $A_{\mathsf{start}} \in V$ denotes a special "start" variable. We say that a string $\alpha \in (\Sigma \cup V)^*$ is *derived* from a variable A_j, denoted by $A_j \stackrel{*}{\Rightarrow} \alpha$, if α can be obtained from A_j by iteratively applying production rules in R. Each such derivation can be described by a *derivation tree*, which is a rooted, directed, ordered, and labeled tree (with edges oriented away from the root), where the root is labeled by A_j, the leaves are labeled by the sybmols of α (in order), and the children of each vertex in the tree correspond to an application of a production rule in G. The language $\mathcal{L} \subseteq \Sigma^*$ *generated* by G consists of all strings that can be derived from A_{start} using the production rules in R.

Let $\mathcal{L}$ be a context-free language and let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be the context-free grammar that generates $\mathcal{L}$. In this section we show how to partition $x \in \mathcal{L}$ into two parts. Next, in Section 3.1.2.3, we show how to extend this technique to multiple parts.

For $x \in \mathcal{L}$ (i.e., $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x$), there exists a derivation tree T corresponding to the derivation $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x$. For simplicity, let us assume that T is a *binary* tree. The root of T is labeled by A_{start} and the leaves are labeled, in order, by $x_1, \dots, x_n$, where $n \stackrel{\text{def}}{=} |x|$. Recall that the Lewis-Stearns-Harmanis Lemma [LSH65] states that every binary tree on n leaves has a subtree[13] with a number of leaves between $n/3$ and $2n/3$. Applying this lemma to T, we can find such a subtree T' of T. Observe that T' induces a partition of $[n]$ into two parts $S_1, S_2 \subseteq [n]$, where S_1 (which is actually an interval) contains all the leaves of T that belong to T' and $S_2 \stackrel{\text{def}}{=} [n] \backslash S_1$ contains all other leaves. The IPP prover finds T' and sends S_1 and A_1 to the verifier, where A_1 is the label of the root of T'. Since S_1 is an interval, the latter requires only $O(\log n)$ communication.

Given (S_1, A_1), the verifier can construct the partition and the corresponding languages, where the partition is simply (S_1, S_2) and the languages are

$$\mathcal{L}_1 \stackrel{\text{def}}{=} \left\{ w \in \Sigma^{|S_1|} \,:\, A_1 \stackrel{*}{\Rightarrow} w \right\}$$

and

$$\mathcal{L}_2 \stackrel{\text{def}}{=} \left\{ w \in \Sigma^{|S_2|} \,:\, A_2 \stackrel{*}{\Rightarrow} w[1, \dots, s-1] \circ A_1 \circ w[s, \dots, |S_2|] \right\},$$

where $A_2 \stackrel{\text{def}}{=} A_{\mathsf{start}}$ and $s \in [n]$ is the starting position of the interval S_1 in $[n]$.

[13] Here and throughout this work, by a subtree, we mean a node of the tree together with *all* of its descendants, see also Section 3.2.3.

Note that $\mathcal{L}_2$ is not quite a context-free language (although $\mathcal{L}_1$ is). Rather, $\mathcal{L}_2$ consists of strings that correspond to *partial derivations* (i.e., derivation processes that end before all symbols are terminals) starting from A_{start} that produce strings that have the variable A_1 in their s^{th} coordinate. We refer to such languages, which we view as generalization of context-free languages, as *partial derivation languages*, and for the recursion to go through, we actually design the original protocol to handle not only context-free languages but also partial derivation languages.

Observe that if $x \in \mathcal{L}$, then clearly $x[S_1] \in \mathcal{L}_1$ and $x[S_2] \in \mathcal{L}_2$. On the other hand, suppose that $x[S_1]$ is ε_1-close to a string $z_1 \in \mathcal{L}_1$ and $x[S_2]$ is ε_2-close to a string $z_2 \in \mathcal{L}_2$. If we choose $i \in \{1,2\}$ at random, such that $\Pr[i=1] = |S_1|/n$ and $\Pr[i=2] = |S_2|/n$, then x is $\mathbf{E}_i[\varepsilon_i]$-close to the string $z = z_2[1,\ldots,s-1] \circ z_1 \circ z_2[s,|S_2|]$. Since $A_1 \overset{*}{\Rightarrow} z_1$ and $A_{\mathsf{start}} \overset{*}{\Rightarrow} z_2[1,\ldots,s-1] \circ A_1 \circ z_2[s,\ldots,|S_2|]$ (because $z_1 \in \mathcal{L}_1$ and $z_2 \in \mathcal{L}_2$), we deduce that $A_{\mathsf{start}} \overset{*}{\Rightarrow} z$, and therefore $z \in \mathcal{L}$. Hence, x is $\mathbf{E}_i[\varepsilon_i]$-close to $\mathcal{L}$.

Given the above, we can design an IPP for $\mathcal{L}$ similarly to the IPP for ROBP that was described in Section 3.1.2.1. Specifically, given (S_1, A_1), the verifier chooses at random $i \in \{1,2\}$ according to the distribution above, sends i to the prover, and both parties run the protocol recursively, with respect to the language $\mathcal{L}_i$ and the input $x[S_i]$.

3.1.2.3 Partitioning Context-Free Languages into Multiple Parts

The first step in partitioning context-free languages into *multiple* parts is a generalization of the Lewis-Stearns-Hartmanis lemma that shows that, for every desired parameter $t \in [n]$, every (constant degree) tree T with n leaves has a subtree with roughly t leaves. The precise statement of the lemma and its proof are given in Lemma 3.5 below.

Using Lemma 3.5, we can partition an input $x \in \mathcal{L}$ into k parts of (roughly) the same size in the following way. As before, we construct a derivation tree T corresponding to the derivation $A_{\mathsf{start}} \overset{*}{\Rightarrow} x$. However, this time we use Lemma 3.5 to find a subtree T_1 with roughly n/k leaves. The coordinates of the leaves of T_1 constitute the first part of the partition (denoted by S_1). To find the second subtree, we remove the entire subtree T_1 from T, *except for its root*. We obtain a new tree T' with (roughly) $n - \frac{n}{k}$ leaves, where one of the leaves of T' is labeled by a variable rather than a terminal. By applying Lemma 3.5 again on the new tree T', we can find a subtree T_2 of T' with roughly n/k leaves. The second part (denoted by S_2) of our partition will consist of the coordinates of all the leaves of T_2 that are labeled by terminals (i.e., are also leaves of the original tree T). We stress that S_2 may not be an interval (but rather two intervals separated by S_1).

We proceed similarly, where in each iteration we remove the subtree that was found in the previous iteration (except for its root) and find a new subtree T_i of T with roughly n/k leaves. The subtrees $T_1, T_2, \ldots, T_k$ induce a partition of $[n]$ where the i^{th} part, denoted S_i (of size roughly n/k), consists of all leaves of T_i that are labeled by terminals (i.e., are leaves of the original tree T) but do not belong to $S_1 \cup \cdots \cup S_{i-1}$.

While the representation of a general partition of $[n]$ into k parts requires $n \cdot \log_2(k)$ bits, we show that the partition $S_1, \ldots, S_\ell$ actually has a concise representation. Indeed,

each subtree T_i induces an interval $I_i \subseteq [n]$, which contains all of its leaves (but potentially also coordinates of other parts in the partition). Given $I_1, \ldots, I_\ell$, the partition $S_1, \ldots, S_\ell$ is uniquely determined (by setting $S_i = I_i \setminus (I_1 \cup \cdots \cup I_{i-1})$). We remark that each pair of *intervals* can be either disjoint or nested (i.e., either $I_i \cap I_j = \emptyset$ or $I_i \subsetneq I_j$).

In light of the foregoing discussion, the prover can send to the verifier the intervals $I_1, \ldots, I_k$ and the variables $A_1, \ldots, A_\ell$ of the roots of the subtrees $T_1, \ldots, T_k$ (respectively). Note that the root of the last subtree T_k is in fact the root of the original derivation tree T (and thus $A_k = A_{\mathsf{start}}$) and that its corresponding interval I_k is $[n]$.

Let $I_{i_1}, \ldots, I_{i_k}$ be the ordered (from left to right) maximal intervals of $I_k = [n]$. That is, the (disjoint) intervals that are contained in I_k but are not contained in any of the other intervals. Observe that if the intervals were generated as prescribed, then A_{start} yields a string x' (composed of terminals and variables) that results from x by replacing the substring $x[I_{i_j}]$ with the variable A_{i_j}, for every $j \in [k]$. Denote the language that contains all such strings by $\mathcal{L}_k$. Similarly, for any interval $I_{i_j} \in \{I_{i_1}, \ldots, I_{i_k}\}$, observe that A_{i_j} yields the string that results from $x[I_{i_j}]$ by replacing coordinates in the maximal intervals that I_{i_j} contains with the corresponding variables. Denote the language of all such strings by $\mathcal{L}_{i_j}$. We show that by applying this idea iteratively we obtain languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ such that (1) if $x \in \mathcal{L}$, then $x[S_i] \in \mathcal{L}_i$ for every $i \in [k]$; and (2) if x is ε-far from $\mathcal{L}$, then $x[S_i]$ is ε-far from $\mathcal{L}_i$, for an average $i \in [k]$, where the average is weighted proportionally to the sizes of $S_1, \ldots, S_k$.

Given the partition above, verifying that $x \in \mathcal{L}$ is reduced to testing that the sub-input $x[S_i]$ is close to $\mathcal{L}_i$, for $i \in [k]$ distributed as above. Hence, as before, the verifier chooses i at random, sends i to the prover and the two parties recursively run an IPP for verifying that $x[S_i]$ is ε-close to $\mathcal{L}_i$.

We emphasize that, as was the case for $k = 2$, the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are not necessarily *context-free languages* but are rather "partial derivation languages". Indeed, for the recursion to go through, we design the IPP to work for such languages (rather than just context-free languages).

3.1.2.4 Digest and Relation to Concatenation Problems

The proofs of Theorems 3.1-3.4 are based on a natural paradigm for designing proofs of proximity. This paradigm consists of two steps: (1) partition the problem into smaller related sub-problems, and (2) verifying a small random sample of the sub-problems. This basic approach was taken by [RVW13] in their construction of an IPP for the Hamming weight problem (i.e., approximating whether a given string has Hamming weight $n/2$). The partitioning in this case is into several intervals of equal length and the IPP prover specifies the Hamming weight of each substring. A more general instantiation of this paradigm was used in [GR13b] to construct MAPs for *parameterized concatenation problems*. Loosely speaking, a language $\mathcal{L}$ is a parameterized concatenation problem if $\mathcal{L} = \mathcal{L}_{\alpha_1} \times \cdots \times \mathcal{L}_{\alpha_k}$, for some integer k, where each language $\mathcal{L}_{\alpha_i}$ is a language parameterized by α_i; thus, the partitioning is done by providing the parameters $\alpha_1, \ldots, \alpha_k$.

In this work we significantly extend the foregoing framework in several aspects: The

partition is not restricted to contiguous intervals, but is rather more involved and depends more dramatically on the structure of the specific language and, moreover, also on the specific input. Furthermore, whereas for concatenation problems the parameterization of each problem is "light" (typically having a logarithmic description length), in our settings the parameterization can be quite extensive, as in *massively parameterized problems* (see survey by Newman [New10]).

3.1.3 Organization

In Section 3.2 we provide the necessarily preliminaries regarding proofs of proximity, context-free languages, and branching programs. In Section 3.3 we construct MAPs and IPPs for languages accepted by ROBPs (with additional discussion on testing affine subspaces in Section 3.3.3). In Section 3.4 we construct MAPs and IPPs for context-free languages (with additional discussion on the Dyck languages in Section 3.4.3). Sections 3.3 and 3.4 can be read independently of each other. We note that the implementation of the outline provided in Section 3.1.2 is far more involved in the case of context-free languages.

3.2 Preliminaries

We begin with some standard notations:

- We denote the concatenation of two strings $x \in \Sigma^n$ and $y \in \Sigma^m$ (over a common alphabet Σ) by $x \circ y \in \Sigma^{n+m}$.

- We denote the *absolute distance* between two (equal length) strings $x \in \Sigma^n$ and $y \in \Sigma^n$ by $\overline{\Delta}(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|$, and their *relative distance* by $\Delta_{\mathsf{REL}}(x,y) \stackrel{\text{def}}{=} \frac{\overline{\Delta}(x,y)}{n}$. If $\Delta_{\mathsf{REL}}(x,y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we denote the *absolute distance* of x from a non-empty set $S \subseteq \Sigma^n$ by $\overline{\Delta}(x,S) \stackrel{\text{def}}{=} \min_{y \in S} \overline{\Delta}(x,y)$ and the *relative distance* of x from S by $\Delta_{\mathsf{REL}}(x,S) \stackrel{\text{def}}{=} \min_{y \in S} \Delta_{\mathsf{REL}}(x,y)$. If $\Delta_{\mathsf{REL}}(x,S) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S.

- We denote the projection of a string $x \in \Sigma^n$ to a subset of coordinates $S \subseteq [n]$ by $x[S]$. For every $i,j \in [n]$, we denote by $x[i,j]$ the projection of x to the interval $[i,j]$ (if $i > j$ then the interval is empty).

- We denote by $A^x(y)$ the output of algorithm A, given direct access to input y and query (i.e., oracle) access to the string x. Given two interactive machines A and B, we denote by $(A^x, B(y))(z)$ the output of A when interacting with B, where A (resp., B) is given oracle access to x (resp., direct access to y) and both parties have direct access to z.

Integrality. Throughout this work, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the closest integer.

3.2.1 Property Testing, MAPs and IPPs

In this section we define testers, MAPs and IPPs. Actually, testers and MAPs will be defined as restrictions of IPPs.

3.2.1.1 IPP

We define a language, over an alphabet Σ, as an ensemble $\mathcal{L} \stackrel{\text{def}}{=} \cup_{n\in\mathbb{N}}\mathcal{L}_n$, where $\mathcal{L}_n \subseteq \Sigma^n$ for every $n \in \mathbb{N}$. The definition of an IPP is a natural extension of the standard definition of IP (interactive proof) where the main distinction is that the verifier only has oracle access to the input. Also, since our focus is on the query and communication complexities, we do not restrict the computational complexity of the verifier (see discussion at the end of Section 3.1).

Definition 3.1 (Interactive Proof of Proximity (IPP) [EKR04, RVW13])**.** *An* interactive proof of proximity *(*IPP*) for the language $\mathcal{L} = \cup_{n\in\mathbb{N}}\mathcal{L}_n$ is an interactive protocol with two parties: a (computationally unbounded) prover $\mathcal{P}$, which has* free *access to input x, and a verifier $\mathcal{V}$, which is a probabilistic computationally unbounded algorithm which has oracle access to x. The parties send messages to each other, and at the end of the communication, the following two conditions are satisfied:*

1. **Completeness:** *For every $n \in \mathbb{N}$, proximity parameter $\varepsilon > 0$ and $x \in \mathcal{L}_n$ it holds that*
$$\Pr\left[(\mathcal{V}^x, \mathcal{P}(x))(n, \varepsilon) = 1\right] \geq 2/3.$$
where the probability is over the coin tosses of $\mathcal{V}$.

2. **Soundness:** *For every $n \in \mathbb{N}$, $\varepsilon > 0$, and $x \in \{0,1\}^n$ that is ε-far from $\mathcal{L}_n$ and for every computationally unbounded (cheating) prover $\mathcal{P}^*$ it holds that*
$$\Pr\left[(\mathcal{V}^x, \mathcal{P}^*)(n, \varepsilon) = 0\right] \geq 2/3.$$
where the probability is over the coin tosses of $\mathcal{V}$.

If the completeness condition holds with probability 1, then we say that the IPP *has a* one-sided error *and otherwise the* IPP *is said to have a* two-sided error*. If all of the verifier's messages are uniformly distributed and independent random strings then the* IPP *is said to be* public-coin*.*

An IPP for $\mathcal{L} = \cup_{n\in\mathbb{N}}\mathcal{L}_n$ is said to have query complexity $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if, for every[14] $n \in \mathbb{N}$, $\varepsilon > 0$ and $x \in \mathcal{L}_n$, the verifier $\mathcal{V}$ makes at most $q(n, \varepsilon)$ queries to x when

[14]We measure the resources used in the protocol only when the verifier interacts with the *honest* prover. However, in the general case, the verifier can simply halt once one of its resources exceeds the corresponding bound (since it knows that it must be interacting with a *cheating* prover).

interacting with $\mathcal{P}$. The IPP is said to have communication complexity $c : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if, for every $n \in \mathbb{N}$, $\varepsilon > 0$ and $x \in \mathcal{L}_n$, the communication between $\mathcal{V}$ and $\mathcal{P}$ consists of at most $c(|x|, \varepsilon)$ bits. A round of communication consists of a single message sent from the verifier to the prover followed by a single message sent from the prover to the verifier. The IPP is said to have r rounds (sometimes called an r-round IPP), for $r : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if, for every $n \in \mathbb{N}$, $\varepsilon > 0$ and $x \in \mathcal{L}_n$, if the number of rounds in the interaction between $\mathcal{V}$ and $\mathcal{P}$ on input x is at most $r(|x|, \varepsilon)$.

The standard definition of a property tester may be derived from Definition 3.1 by restricting the communication complexity to 0. The definition of an MAP can be derived by restricting the communication to be only from the prover to the verifier (see [GR13b] for further details on MAPs).

Non-uniform IPPs. While Definition 3.1 refers to a *uniform* definition of IPP, throughout this work it will be convenient for us to use a *non-uniform* definition. That is, we fix an integer $n \in \mathbb{N}$, which we think of as a variable parameter, and restrict Definition 3.1 to inputs of length n. Hence, unless stated otherwise, by an IPP we actually mean the *non-uniform* variant. Despite the fact that the integer n is fixed, we view it as a generic parameter and allow ourselves to write asymptotic expressions such as $O(n)$. We also note that while our results are proved in terms of non-uniform IPP, they can be extended to the uniform setting in a straightforward manner.

3.2.1.2 Proximity Oblivious IPP

Extending the notion of proximity oblivious testers [GR11], we define a *proximity oblivious* IPP, as a variant of an IPP in which neither party receives the proximity parameter as input and for every $\varepsilon > 0$, the verifier is required to reject inputs that are ε-far from the language with some probability $\rho(\varepsilon)$.

Definition 3.2 (Proximity Oblivious IPP)**.** *Let $\rho : (0, 1] \to (0, 1]$ be a monotone function. A* proximity oblivious IPP with detection probability ρ*, for the language $\mathcal{L} = \cup_{n \in \mathbb{N}} \mathcal{L}_n$ is similar to Definition 3.1, except that the neither the verifier nor the prover*[15] *receive the proximity parameter as input, and the completeness and soundness conditions are modified as follows:*

1. **Completeness:** *For every $n \in \mathbb{N}$, and $x \in \mathcal{L}_n$ it holds that*[16]

$$\Pr\left[(\mathcal{V}^x, \mathcal{P}(x))(n) = 1\right] = 1.$$

[15] Since we do not bound the computational resources of the prover, it can simply deduce the proximity parameter from the input.

[16] Note that we require the verifier to accept inputs $x \in \mathcal{L}$ with probability 1. A more general definition could allow this probability to be some smaller constant or even a function of ε (see [GS12]). For simplicity (and since it suffices for our purposes), in this work we only consider proximity oblivious IPPs with perfect completeness.

2. **Soundness:** *For every $n \in \mathbb{N}$, every $x \in \Sigma^n$, and for every computationally unbounded (cheating) prover $\mathcal{P}^*$ it holds that*

$$\Pr\left[(\mathcal{V}^x, \mathcal{P}^*)(n) = 0\right] \geq \rho\big(\Delta(x, \mathcal{L}_n)\big)$$

In both condiditons the probability is over the coin tosses of $\mathcal{V}$.

Note that any proximity oblivious IPP with detection probability $\rho(\cdot)$, can be transformed into a standard IPP (as in Definition 3.1) by repeating the proximity oblivious IPP $O(1/\rho(\varepsilon))$ times in parallel (see Section 3.5.1 for details on parallel repetition for IPPs).

3.2.2 Read-Once Branching Programs (ROBPs)

In this section we provide the necessary background on ROBPs (needed only for Section 3.3). An ROBP is defined as follows

Definition 3.3 (ROBP)**.** *A* branching program *on n variables is a directed acyclic graph that has a unique source vertex with in-degree 0 and (possibly) multiple sink vertices with out-degree 0. Each sink vertex is labeled either with 0 (i.e.,* reject*) or 1 (i.e.,* accept*). Each non-sink vertex is labeled by an index $i \in [n]$ and has exactly 2 outgoing edges, which are labeled by 0 and 1.*

The output of the branching program B on input $x \in \{0,1\}^n$, denoted $B(x)$, is the label of the sink vertex reached by taking a walk, starting at the source vertex such that at every vertex labeled by $i \in [n]$, the step taken is on the edge labeled by x_i.

The branching program is said to be read-once *(or ROBP for short) if along every path from source to sink, every index $i \in [n]$ appears at most once. The* size *of a branching program B, denoted $|B|$, is the number of vertices in its graph.*

Let φ and ψ be vertices in the branching program B on n variables and let $x \in \{0,1\}^n$. Loosely speaking, we write $\varphi \overset{x,k}{\rightsquigarrow} \psi$ if the walk of length k corresponding to x that starts at φ ends at ψ. Note that only k coordinates of x are read (adaptively) in this walk and that φ itself only determines the first variable read. Formally, we write $\varphi \overset{x,1}{\rightsquigarrow} \psi$ if the edge (φ, ψ) appears in B and is labeled by x_i, where i is the label of φ, and we (inductively) write $\varphi \overset{x,k}{\rightsquigarrow} \psi$ if there exists a vertex ζ in B such that $\varphi \overset{x,1}{\rightsquigarrow} \zeta$ and $\zeta \overset{x,k-1}{\rightsquigarrow} \psi$.

3.2.3 Context-Free Languages

In this section we provide the necessary background on context-free languages (needed only for Section 3.4). To define *context-free languages*, we first define context-free grammars (see [HMU06] for more details).

Definition 3.4 (Context-free grammar)**.** *A* context-free grammar *is a tuple $G = (V, \Sigma, R, A_{\mathsf{start}})$ such that V is a (finite) set of symbols, referred to as* variables*; Σ is a (finite) set of symbols, referred to as* terminals*; $R \subseteq V \times (V \cup \Sigma)^*$ is a (finite) relation, where each $(A, \alpha) \in R$ is referred to as a* production rule *and is denoted by $A \to \alpha$; $A_{\mathsf{start}} \in V$ is a variable that is referred to as the* start variable*.*

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar, and let $\alpha, \beta \in (V \cup \Sigma)^*$ be strings of terminals and variables. We say that α directly yields β, denoted by $\alpha \Rightarrow \beta$, if there exists a production rule $A \to \gamma$ in R such that β is obtained from α by replacing exactly one occurrence of the variable A in α with the string $\gamma \in (V \cup \Sigma)^*$. We say that α yields β, denoted $\alpha \overset{*}{\Rightarrow} \beta$ if there exists a finite sequence of strings $\alpha_0, \ldots, \alpha_k \in (V \cup \Sigma)^*$ such that $\alpha_0 = \alpha$, $\alpha_k = \beta$, and $\alpha_0 \Rightarrow \ldots \Rightarrow \alpha_k$. The language $\mathcal{L}_n \subseteq \Sigma^n$ is a context-free language if there exists a grammar $G = (V, \Sigma, R, A_{\mathsf{start}})$ such that $\mathcal{L}_n = \{x \in \Sigma^n : A_{\mathsf{start}} \overset{*}{\Rightarrow} x\}$, where the derivation is with respect to the rules in R.

Derivation Tree. Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar. For $A \in V$ and $x \in \Sigma^*$, a derivation tree, corresponding to the derivation $A \overset{*}{\Rightarrow} x$, is a rooted, directed, ordered, and labeled tree T (with edges oriented away from the root) that satisfies the following properties:

- Each internal vertex is labeled by some variable, and the root is labeled by the variable A.
- Each leaf is labeled by a terminal symbol, where the i^{th} leaf is labeled by the i^{th} symbol of x.
- For every internal vertex v, if v is labeled by the variable $A' \in V$ and for every $i \in [k]$ (where k is the number of children of v) the i^{th} child of v is labeled by $\alpha_i \in V \cup \Sigma$, then the production rule $A' \to \alpha_0 \circ \cdots \circ \alpha_k$ must belong to R.

For every derivation $A \overset{*}{\Rightarrow} x$ there exists a corresponding derivation tree.

Trees and the Lewis-Stearns-Hartmanis Lemma. In this work we only consider trees that are rooted, directed, and ordered (e.g., derivation trees as above). Thus, throughout this work, whenever we say tree we mean a rooted, directed, and ordered tree (with edges oriented away from the root). Note that the fact that the tree is ordered induces an ordering of its leaves. We define the arity of a tree to be the maximal number of children of any vertex in the tree. We follow the data-structure literature and define a subtree of a tree T as a tree consisting of a node in T and all of its descendants in T.[17]

For a tree T, we denote by $L(T)$ the number of leaves of T. We will use the following straightforward generalization of the Lewis-Stearns-Hartmanis Lemma [LSH65]:

Lemma 3.5. *Let T be a tree with arity d and let $t \in [L(T)]$. Then, there exists a subtree T' of T with $L(T') \in [t/d, t]$ leaves.*

The Lewis-Stearns-Hartmanis lemma corresponds to the special case of Lemma 3.5 in which $d = 2$ (i.e., a binary tree) and $t = 2n/3$.

[17] This definition differs from the graph-theoretic definition that defines a subtree as any connected subgraph of a tree. For example, the root of a tree is a subtree in the graph theoretic sense but not according to our definition (unless the tree has exactly one vertex).

Proof of Lemma 3.5. We prove by induction on the *size* of the tree (not on the number of leaves), noting that the base case holds trivially. Suppose that the lemma holds for all trees of size less than n. Let T be a tree of size n and let $t \in [L(T)]$.

If $t = L(T)$, then we are done (since $L(T) \in [t/d, t]$ and so T itself has the desired property). Otherwise, if $t < L(T)$, then T has a subtree T' (rooted at one of its children) such that $L(T') \geq t/d$ (since otherwise T has a total of less than $d \cdot t/d = t < L(T)$ leaves). If $L(T') \leq t$, then we are also done (since $L(T') \in [t/d, t]$ and so T' satisfies the desired property). Otherwise, $t \in [L(T')]$ and the lemma follows by applying the induction hypothesis on T'. □

3.3 MAPs and IPPs for Read-Once Branching Programs

In this section we prove Theorem 3.4 (and Theorem 3.2 will follow as a special case of the proof) by constructing an IPP for every language that is accepted by an ROBP.

3.3.1 IPPs for ROBPs

Before presenting the IPP formally, we provide a short overview of the protocol (for a more detailed overview, see Section 3.1.2.1). Let B be an ROBP on n variables. We construct an r-round public-coin IPP for the language that is accepted by B. The IPP runs recursively, where each round of communication is as follows. Given an input x that is accepted by B within $n' \leq n$ steps, the prover finds the accepting path $\varphi_0 \to \varphi_1 \to \cdots \to \varphi_{n'}$ and sends to the verifier the subsequence $\varphi_{n'/k}, \ldots, \varphi_{i \cdot n'/k}, \ldots, \varphi_{n'}$ that contains every $(n'/k)^{\text{th}}$ vertex along this path. The verifier checks that every two consecutive vertices in the prover's message are connected (this can be done *without* making queries to the input x), then selects uniformly at random $i \in [k]$, sends i to the prover, and defines a new ROBP B_i that has the same graph as B, but its source vertex is $\varphi_{(i-1) \cdot n'/k}$ and its *unique* accepting sink is $\varphi_{i \cdot n'/k}$.[18] Subsequently, both parties (recursively) invoke the IPP protocol on input x and the ROBP B_i.

A crucial point is that although the input x does not not shrink in each recursive call, the *effective length of the input*, n', which is the alleged number of bits of x that need to be read in order to get from the source to the accepting sink, does shrink (by a factor of k). Hence, after r such rounds, the verifier can read all bits of x that are required to verify the current statement (which refers to a path of length n/k^r). Interestingly, and in contrast to the simpler case of OBDDs, in this last step the verifier reads the n/k^r bits of the input *adaptively*, based on the steps taken by the ROBP.

[18] One could alternatively define B_i to consist only of vertices at distance at most n'/k from $\varphi_{(i-1) \cdot n'/k}$. We refrain from taking this approach due to technical reasons. Note that also when using this alternate definition, when considering general ROBPs (rather than OBDDs), B_i could potentially look at all of the n bits of the input (see discussion at the end of Section 3.1.2.1).

In the IPP that we construct, we assume for simplicity that the verifier is given an integer $n' \leq n$, and the claim (which the verifier is trying to validate) is that $B(x) = 1$ after reading *exactly* n' bits of the input x. Furthermore, we assume that the ROBP B is such that there exists *some* accepting path (i.e., from the source to some accepting sink) of length n'. We can reduce the general case to this restricted setting by having the prover send n' as part of its first message and having the verifier explicitly check that there is some accepting path of length n' (this check requires no queries to the main input x).

Recall that, as noted in Section 3.1.2, to facilitate the recursion we use the notion of a *proximity oblivious* IPP (see Section 3.2.1.2). When handling general ROBPs (rather than OBDDs), we follow this approach in *spirit* but, due to technical reasons, the construction will not exactly fit Definition 3.2. More specifically, since in each step of the recursion only the *effective* length of the input shrinks (but the actual length of the input stays the same), throughout the proof it will be more convenient for us to use *absolute* distances, denoted by $\overline{\Delta}$ (see Section 3.2) rather than with distances that are relative to n. Hence, the detection probability $\overline{\rho}$ of the verifier will be a function of the *absolute distance* (rather than of the relative distance) of the input from the language. That is, we will construct an absolute proximity oblivious IPP with detection probability $\left\{\overline{\rho}_n : \{0,\ldots,n\} \to (0,1]\right\}_{n\in\mathbb{N}}$, which is the same as Definition 3.2 except that we modify the soundness condition as follows:

- ***Soundness:*** For every $n \in \mathbb{N}$, $x \in \Sigma^n$, and for every computationally unbounded (cheating) prover $\mathcal{P}^*$ it holds that

$$\Pr\left[(\mathcal{V}^x, \mathcal{P}^*)(n) = 0\right] \geq \overline{\rho}_n\big(\overline{\Delta}\left(x, \mathcal{L}_n\right)\big) \tag{3.1}$$

Again, we can transform an *absolute* proximity oblivious IPP with detection probability $\left\{\overline{\rho}_n : \{0,\ldots,n\} \to (0,1]\right\}_{n\in\mathbb{N}}$ into a standard IPP (as in Definition 3.1) by repeating the base protocol $O(1/\overline{\rho}_n(\varepsilon \cdot n))$ times in parallel.

The absolute proximity oblivious IPP for ROBPs, denoted ROBP-IPP, is presented in Fig. 3.1 (recall that the notation $\varphi \overset{x,m}{\rightsquigarrow} \psi$, which is used in Fig. 3.1 means that, given input x, the ROBP walks from φ to ψ in m steps, see Section 3.2.2 for details).

It can be easily verified that the round complexity is r, the communication complexity is $O(rk \cdot \log(|B|))$ and the query complexity is $O(n/k^r)$. We proceed to show that completeness and soundness hold.

Completeness. Let B be a ROBP on n variables, let $r \geq 0$, $n' \in [n]$, and let $x \in \{0,1\}^n$ such that $B(x) = 1$ after reading exactly n' bits of the input. (Perfect) completeness follows by induction on r as follows.

For $r = 0$, the verifier just reads the appropriate n' bits of the input and accepts with probability 1. For $r \geq 1$, let $(\varphi_0, \varphi_1, \ldots, \varphi_{n'})$ be the accepting path corresponding to x. The checks that the verifier performs in the current round pass, since $\varphi_{n'}$ is indeed an accepting sink and $\varphi_{(i-1)\cdot n'/k^r} \overset{x,n'/k^r}{\rightsquigarrow} \varphi_{i\cdot n'/k^r}$, for every $i \in [p^r]$. Furthermore, since for

The Protocol $\mathsf{ROBP\text{-}IPP}^{B}_{n,n',r}$**:**

Common Input: Integers $n, n' \in \mathbb{N}$, a ROBP B on n variables such that there exists some accepting path of length n' in B, and a parameter $r \in \mathbb{N}$.

Prover's Input: Direct access to $x \in \{0,1\}^n$ such that $B(x) = 1$ in exactly n' steps.

Verifier's Input: Oracle access to the same x.

1. If $r = 0$, the verifier $\mathcal{V}$ checks whether $B(x) = 1$ after exactly n' steps by (adaptively) reading the appropriate n' bits of x. If $B(x) = 1$, then $\mathcal{V}$ accepts, otherwise it rejects, and in either case both parties terminate the protocol.
2. The Prover $\mathcal{P}$:
   (a) Let $\varphi = (\varphi_0, \ldots, \varphi_{n'}) \in [|B|]^{n'}$ be the sequence of vertices in the accepting path (of length $n' \leq n$) in B that corresponds to the evaluation of B on input x.
   (b) Send $(\varphi_{n'/k}, \varphi_{2n'/k}, \ldots, \varphi_{n'})$ to $\mathcal{V}$.[a]
3. The Verifier $\mathcal{V}$:
   (a) Check that $\varphi_{n'}$ is an accepting sink of B.
   (b) Let φ_0 be the source of B.
   (c) For every $i \in [k]$, check that there exists some input $x^{(i)} \in \{0,1\}^n$ such that $\varphi_{(i-1)\cdot n'/k} \overset{x^{(i)}, n'/k}{\rightsquigarrow} \varphi_{i\cdot n'/k}$.[b]
   (d) Select uniformly at random an index i in $[k]$, and send i to $\mathcal{P}$.
4. Denote by B_i the ROBP that has the same graph as B, except that its source is $\varphi_{(i-1)\cdot n'/k}$, and its *unique* accepting vertex is $\varphi_{i\cdot n'/k}$.
5. Both parties (recursively) invoke $\mathsf{ROBP\text{-}IPP}^{B_i}_{n,n'',r-1}$, where $n'' = n'/k$, on input x.

[a] The accepting sink $\varphi_{n'}$ is also sent since (in the first step of the recursion) there could be multiple accepting sinks.

[b] This check is performed without making any queries to the main input x. Although our focus is not on *computational* complexity, we note that this can be done in $\mathsf{poly}(s)$ time.

Figure 3.1: IPP for ROBPs

every choice of $i \in [k]$ (made by the verifier) it holds that $\varphi_{(i-1)\cdot n'/k} \overset{x,n'/k}{\rightsquigarrow} \varphi_{i\cdot n'/k}$, the two parties recursively run the $r-1$ round protocol on a branching program B_i such that $B_i(x) = 1$ after reading exactly n'/k bits of x. Hence, by the inductive hypothesis the verifier accepts with probability 1.

Soundness. Soundness follows directly from the following lemma, which is proved by induction on the number of rounds r. We suggest to the reader to first consider the case that $n' = n$ in both the lemma statement and its proof. Nevertheless, we stress that in lower levels of the recursion, the parameter n' becomes much smaller than n.

Lemma 3.6. *Let $n \in \mathbb{N}$, $n' \in [n]$ and $r \geq 0$. For every ROBP B of size s on n variables that has an accepting path of length n', every x that is in* absolute *distance $\varepsilon \cdot n'$ from $\{z \in \{0,1\}^n : B(z) = 1\}$, and for every cheating prover strategy $\mathcal{P}^*$ it holds that:*

$$\Pr[(\mathcal{V}^x, \mathcal{P}^*)(n, n', B, r) = 0] \geq \varepsilon,$$

where $\mathcal{V}$ is the r-round verifier of ROBP-IPP *(of Fig. 3.1).*

Proof. We prove Lemma 3.6 by induction on the number of rounds $r \geq 0$. In the base case, corresponding to $r = 0$, the verifier simply ignores the prover and reads the appropriate n' bits of x. Hence, if $B(x) \neq 1$, then the verifier rejects with probability 1.[19]

In the inductive step, for $r \geq 1$, let $x \in \{0,1\}^n$ that is at absolute distance $\varepsilon \cdot n'$ from $\{z \in \{0,1\}^n : B(z) = 1\}$ and let P^* be a (deterministic) cheating prover strategy for the protocol ROBP-IPP of Fig. 3.1 (with r rounds). Let $(\varphi_{n'/k}, \varphi_{2n'/k}, \ldots, \varphi_{n'})$ be the first message sent by P^* to $\mathcal{V}$ and let φ_0 be the source. Since the verifier explicitly checks these conditions, it must be the case that $\varphi_{n'}$ is an accepting sink, and that for every $i \in [k]$, there exists some $x^{(i)} \in \{0,1\}^n$ such that $\varphi_{(i-1)\cdot n'/k} \overset{x^{(i)},n'/k}{\rightsquigarrow} \varphi_{i\cdot n'/k}$. Furthermore, for every $i \in [k]$, we assume without loss of generality that $x^{(i)}$ is the string z that minimizes the distance of x to the set $\left\{z \in \{0,1\}^n : \varphi_{(i-1)\cdot n'/k} \overset{z,n'/k}{\rightsquigarrow} \varphi_{i\cdot n'/k}\right\}$.

Recall that $\overline{\Delta}$ denotes absolute distance (see Section 3.2) and let $\varepsilon_i \overset{\text{def}}{=} \frac{\overline{\Delta}(x,x^{(i)})}{n'/k}$. The following claim, which crucially uses the fact that B is *read-once*, shows that the average of the ε_i's (which will later be shown to lower bound the rejection probability of $\mathcal{V}$) is at least ε.

Claim 3.6.1. $\mathbf{E}_i[\varepsilon_i] \geq \varepsilon$.

Proof. For every $i \in [k]$, let $J_i \subseteq [n]$ be the set of n'/k variables that are read when going from $\varphi_{(i-1)\cdot n'/k}$ to $\varphi_{i\cdot n'/k}$ on input $x^{(i)}$. We first prove that the sets J_i's are disjoint. Assume otherwise; that is, that there exists $j \in J_{i_1} \cap J_{i_2}$ for some $i_1, i_2 \in [k]$. For every $i \in [k]$, denote the path from $\varphi_{(i-1)\cdot n'/k}$ to $\varphi_{i\cdot n'/k}$ (in B) on input $x^{(i)}$ by P_i. Consider

[19] In the trivial case that $\varepsilon = 0$ (i.e., $B(x) = 1$), the verifier also satisfies the requirement, since it rejects with probability at least $0 = \varepsilon$. It may also be worth mentioning that it always holds that $\varepsilon \leq 1$, since B has an accepting path of length n'.

the concatenated path $P_1 \circ \cdots \circ P_k$. This is a path in B in which the label j appears twice (both in P_{i_1} and in P_{i_2}) in contradiction to our assumption that B is a *read-once* branching program.

Define $x' \in \{0,1\}^n$ as follows. For every $j \in [n]$, if $j \in J_i$ for some $i \in [k]$ (which must be unique as just shown), then $x'[j] \stackrel{\text{def}}{=} x^{(i)}[j]$, and otherwise (i.e., if $j \notin J_1 \cup \cdots \cup J_p$) we set $x'[j] \stackrel{\text{def}}{=} x[j]$. Note that

$$\varphi_0 \overset{x',n'/k}{\rightsquigarrow} \varphi_{n'/k} \overset{x',n'/k}{\rightsquigarrow} \cdots \overset{x',n'/k}{\rightsquigarrow} \varphi_{n'}$$

and therefore $B(x') = 1$. The claim follows by noting that

$$\varepsilon \cdot n' \leq \overline{\Delta}(x, x') = \sum_{i \in [k]} \overline{\Delta}(x[J_i], x'[J_i]) = \sum_{i \in [k]} \overline{\Delta}\left(x[J_i], x^{(i)}[J_i]\right) \leq \sum_{i \in [k]} \overline{\Delta}\left(x, x^{(i)}\right) = \sum_{i \in [k]} \varepsilon_i \cdot n'/k,$$

where the first inequality follows from the fact that x is in absolute distance $\varepsilon \cdot n'$ from $\{z \in \{0,1\}^n : B(z) = 1\}$ combined with the fact that $B(x') = 1$, and the first and second equality follow from the definition of x' (the first equality also uses the fact that the J_i's are disjoint). The claim follows. □

For every $i \in [k]$, let B_i be the ROBP that has the same graph as B, but its source is $\varphi_{(i-1)\cdot n'/k}$, and its *unique* accepting vertex is $\varphi_{i \cdot n'/k}$. Let P_i^* be the residual $r-1$ round strategy of P^* after receiving the message i from $\mathcal{V}$ in the first round, and let $\mathcal{V}_i$ be the residual strategy of $\mathcal{V}$ after fixing its first message to i. Note that $\mathcal{V}_i$ is exactly the strategy of the verifier in $\mathsf{ROBP\text{-}IPP}^{B_i}_{n,n',r-1}$.

Claim 3.6.2. *For every $i \in [k]$, it holds that*

$$\Pr[(\mathcal{V}_i^x, P_i^*)(n, n'', B_i, r-1) = 0] \geq \varepsilon_i,$$

where $n'' = n'/k$.

Proof. Let $i \in [k]$. Recall that $x^{(i)}$ was chosen as $z \in \{0,1\}^n$ that minimizes the distance of x to the set $S_i \stackrel{\text{def}}{=} \left\{z \in \{0,1\}^n : B_i(z) = 1 \text{ using exactly } n'/k \text{ steps}\right\}$. Hence,

$$\overline{\Delta}(x, S_i) = \overline{\Delta}\left(x, x^{(i)}\right) = \varepsilon_i \cdot n'/k.$$

Hence, $(\mathcal{V}_i^x, P_i^*)(n, n'', B_i, r-1)$ corresponds to an invocation of the $r-1$ round version of the protocol on an input x that is in absolute distance $\varepsilon_i \cdot n'/k$ from S_i. Therefore, by the inductive hypothesis, the verifier $\mathcal{V}_i$ rejects with probability at least ε_i. □

Using Claim 3.6.1 and Claim 3.6.2 we obtain that

$$\Pr[(\mathcal{V}^x, P^*)(n, n', B, r) = 0] = \mathop{\mathbf{E}}_{i \in [k]}\left[\Pr[(\mathcal{V}_i^x, P_i^*)(n, n'/k, B_i, r-1) = 0]\right] \geq \mathop{\mathbf{E}}_{i \in [k]}[\varepsilon_i] \geq \varepsilon, \tag{3.2}$$

and the lemma follows. □

This concludes the proof of Theorem 3.4.

Remark 3.7 (Computational Complexity)**.** *The running time of the* IPP *prover in Fig. 3.1 is polynomial in its input (i.e.,* $\mathsf{poly}(|B|, n, k, r, 1/\varepsilon)$*). As for the* IPP *verifier, if the representation of the ROBP* B *allows one to check if two vertices in the graph of* B *are connected in* $\mathsf{polylog}(|B|)$ *time, then the verifier runs in time* $\mathsf{poly}(\log n, k, r, \log(|B|), 1/\varepsilon)$*. If such a representation is not available, then it can be generated in a relatively expensive (i.e.,* $\mathsf{poly}(|B|)$ *time) pre-processing step, which does not depend on the input* x *and can be re-used for multiple inputs.*

Alternatively, for some other natural representations, the verifier can employ the prover to efficiently check if two vertices in B *are connected. Consider for example a natural representation in which there exists a polynomial (i.e.,* $\mathsf{poly}(\log(|B|))$*) size circuit* C *that on input a vertex* v *(in the graph of* B*) and a bit* $\sigma \in \{0,1\}$ *outputs the neighbor* u *of* v *that* σ *leads to (i.e., the edge* (v,u) *is labeled by* σ*). Suppose further that* C *is* $O(\log(|B|))$*-space uniform (i.e., can be generated by an* $O(\log(|B|))$*-space Turing machine). In such case we can use the prover to check connectivity, as described next, and so we obtain sub-linear verification.*

In order to verify connectivity efficiently, we first observe that there exists an ($O(\log(|B|))$*-space uniform) circuit, of* $\mathsf{polylog}(|B|)$*-depth and* $\mathsf{poly}(|B|)$*-size, that on input two vertices* v *and* u *outputs* 1 *if and only if they are connected (possibly via a long path).*[20] *Now we can apply the efficient interactive proof-system for low-depth computation*[21] *of Goldwasser et al. [GKR08, Theorem 1] to obtain an interactive proof-system that verifies that two given vertices are connected, where the verifier runs in time* $\mathsf{polylog}(|B|)$ *and the prover runs in time* $\mathsf{poly}(|B|)$*.*[22] *We note that employing this proof-system inside our* IPP *increases the round complexity of the* IPP *by a* $\mathsf{polylog}(|B|)$ *factor.*

Remark 3.8 (IPPs for Ordered Binary Decision Diagrams)**.** *Recall that an ordered binary decision diagram (OBDD) is an ROBP that is both* layered *and* ordered *(see Section 3.1.2.1). We observe that the communication complexity in Theorem 3.4 can be slightly improved for OBDDs of width* w *and size* $s = O(nw)$ *from* $O\big((pr \log s) \cdot \varepsilon^{-1}\big)$ *to* $O\big((pr \log w) \cdot \varepsilon^{-1}\big)$*, by noting that the* i^{th} *vertex specified by the prover (say, in the first round)* must *be in layer* $i \cdot n/p$ *and therefore it can be specified using only* $\log_2 w$ *bits.*

[20] The circuit first uses C to generate the entire adjacency matrix of B and then checks whether v and u are connected by repeated squaring of the adjacency matrix. Note that all actions can be implemented in $\mathsf{polylog}(|B|)$-depth and $\mathsf{poly}(|B|)$-size.

[21] Goldwasser *et al.* show that any language that is accepted by a ($O(\log(S(n)))$-space uniform) circuit of depth $D(n)$ and size $S(n)$, has an interactive proof-system, where the verifier runs in time $(n + D(n)) \cdot \mathsf{polylog}(S(n))$ and the prover runs in time $\mathsf{poly}(S(n))$.

[22] We stress that the interactive proof-system for verifying connectivity is a standard interactive proof-system and not a "proof of proximity" (i.e., not an IPP). Indeed, this is crucial for our application since we use the interactive proof-system for connectivity as a subroutine within our IPP, and the IPP verifier should reject if at any point it encounters a pair of vertices that are disconnected (even if the pair is "close" to being connected).

3.3.2 MAPs for ROBPs

We observe that Theorem 3.2 follows almost directly from the proof of Theorem 3.4, when restricted to the case $r = 1$. Indeed, the only two gaps (which are easily resolved) are:

1. Interaction: Theorem 3.4 (restricted to $r = 1$) guarantees a 1-round IPP for languages recoginzed by ROBPs. In general, a 1-round IPP is not necessarily an MAP, since it may include a message sent from the verifier to the prover. Nevertheless, the order of the messages in our protocol is such that first the prover sends a message to the verifier and then the verifier responds. The last message can clearly be avoided and so we obtain an MAP.

2. Dependence on the Proximity Parameter in the Proof Length: Recall that there is a linear dependence on $1/\varepsilon$ in the communication complexity in Theorem 3.4, due to the $O(1/\varepsilon)$ parallel repetitions that were used. However, for MAPs, parallel repetition can be performed without increasing the proof length, since the proof is a deterministic function of the input. Hence, we can save the additional $O(1/\varepsilon)$ factor that is used for general IPPs.

3.3.3 MAPs and IPPs for Affine Spaces

In this section, as an example, we show how Theorems 3.2 and 3.4 can yield MAPs and IPPs for any *affine space*.

Before proceeding to the proof, we remark that Rothblum *et al.* [RVW13] identified a specific affine space, called PVAL, as being "complete" for the construction of IPPs for the class NC.[23] They constructed an IPP for PVAL and thereby obtain IPPs for all of NC. Interestingly, PVAL is an affine space and so the results of this section yield an alternative IPP for it. Unfortunately though, the parameters obtained by our IPP are inferior[24] to those of [RVW13] and do not yield an alternative IPP for NC.

Definition 3.9. *Let $\mathbb{F}$ be a finite field, $n \in \mathbb{N}$ and $t \in [n]$. An* affine subspace *of the vector space $\mathbb{F}^n$, denoted* $\mathsf{AffineSpace}_{A,b}$*, is parametrized by a matrix $A \in \mathbb{F}^{t \times n}$ and a vector $b \in \mathbb{F}^t$ and consists of all strings $x \in \mathbb{F}^n$ such that $Ax = b$.*

Our construction of an IPP for every affine space follows directly from Theorem 3.4 by showing that membership in an affine subspaces can be recognized by a small-width OBDD.

Proposition 3.10. *Let $\mathbb{F}$ be a finite field, $n \in \mathbb{N}$ and $t \in [n]$. For every $A \in \mathbb{F}^{t \times n}$ and $b \in \mathbb{F}^t$, there exists a width $|\mathbb{F}|^{O(t)}$ OBDD that accepts* $\mathsf{AffineSpace}_{A,b}$.

[23] The language PVAL is parameterized by a sequence of points in a finite vector space and a sequence of values, and consists of all strings x whose low degree extension $\mathsf{LDE}(x)$ is equal to the given sequence of values at the corresponding sequence of points

[24] More specifically, for a PVAL instance parameterized by t points, the communication complexity in our protocol is $O(t \cdot 1/\varepsilon \cdot \mathsf{polylog}(n))$, whereas in [RVW13] it is $O(t \cdot (1/\varepsilon)^{o(1)} \cdot \mathsf{polylog}(n))$. Our result is insufficient since in the context of the proof of IPPs for NC, $t = \sqrt{n}$ and $\varepsilon = 1/\sqrt{n}$.

Proof. We describe a deterministic streaming algorithm for deciding membership in $\mathsf{AffineSpace}_{A,b}$. The algorithm gets access to a stream of n fields elements, reads the input element-by-element (in order) and stores a total of t field elements at any given time. Transforming the latter into an OBDD, as required, is straightforward.[25]

Denote the columns of A by $a_1, \ldots, a_n \in \mathbb{F}^t$. The algorithm maintains a vector $c \in \mathbb{F}^t$ which is initialized to 0. The streaming algorithm reads the input $x \in \mathbb{F}^n$ element-by-element and after reading the i^{th} element, the algorithm sets $c \leftarrow c + x_i a_i$ (where the addition is over $\mathbb{F}$). In the end, it holds that

$$c = \sum_{i=1}^{n} x_i a_i = Ax$$

and therefore it suffices for the algorithm to accept if $c = b$ and reject otherwise.

□

By applying Theorem 3.4, we obtain the following corollary.

Corollary 3.5. *Let $\mathbb{F}$ be a finite field, $n \in \mathbb{N}$ and $t \in [n]$. For every $A \in \mathbb{F}^{t \times n}$, $b \in \mathbb{F}^t$ and for every $k = k(n) \geq 2$ and $r = r(n) \geq 1$ such that $k^r \leq n$, there exists an r-round IPP for $\mathsf{AffineSpace}_{A,b}$ with communication complexity $O\big((rk \cdot t \log |\mathbb{F}|) \cdot \varepsilon^{-1}\big)$ and query complexity $O\left(\frac{n}{k^r} \cdot \varepsilon^{-1}\right)$. Furthermore, the IPP is public-coin and has one-sided error.*

3.4 MAPs and IPPs for Context-Free Languages

In this section we prove Theorem 3.3 by constructing an IPP for any context-free language. As noted in the introduction, the proof of Theorem 3.1 will follow as a special case of this IPP.

The proof of Theorem 3.3 extensively uses the notions of a *partial derivation* and a *partial derivation language*. Recall that a partial derivation of a grammar G is a derivation, according to the production rules of G, in which not all variables are expanded. Our notion of a *partial derivation language* is more complex. In particular, it does *not* refer to the language that consist of all possible partial derivations of the grammar (i.e., $\{x \in (\Sigma \cup V)^* : A_{\mathsf{start}} \overset{*}{\Rightarrow} x\}$). Rather, we define a partial derivation language as a language that consists of the subsequence of *terminal symbols* that correspond to partial derivations that start at some fixed variable. Furthermore, we consider only partial derivations in which the subsequence of variables in the partial derivation occur in specific locations. More concretely, a partial derivation language is parameterized by (1) a start variable A_0; (2) the number of terminals m; (3) a sequence of ℓ locations $i_1, \ldots, i_\ell$; and (4) a corresponding sequence of variables $A_1, \ldots, A_\ell$. The language consists of strings z of length m such that the string $z' = z[1, i_1 - 1] \circ A_1 \circ z[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ z[i_\ell, m]$ can be derived from A_0. More formally,

[25] Loosely speaking, each layer of the OBDD will consist of the $2^{O(t \log |\mathbb{F}|)}$ possible configurations of the streaming algorithm (which include both its current state and possibly some of the bits of the element that is currently being read).

Definition 3.11 (Partial Derivation Language)**.** *A* partial derivation language *of the grammar $G = (V, \Sigma, R, A_{\mathsf{start}})$ is a language $\mathcal{L} \subseteq \Sigma^m$, parameterized by indices $1 \leq i_1 \leq \ldots \leq i_\ell \leq m$ and variables $A_0, \ldots, A_\ell \in V$ such that*

$$\mathcal{L} \stackrel{\text{def}}{=} \left\{ z \in \Sigma^m \;:\; A_0 \stackrel{*}{\Rightarrow} z[1, i_1 - 1] \circ A_1 \circ z[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ z[i_\ell, m] \right\}.$$

The concise description of a partial derivation language $\mathcal{L} \subseteq \Sigma^m$, parameterized by $\overline{i} = (i_1, \ldots, i_\ell)$ and $\overline{A} = (A_0, \ldots, A_\ell)$, is denoted by $\langle \mathcal{L} \rangle \stackrel{\text{def}}{=} \left(m, \overline{i}, \overline{A}\right)$.

We stress that $z \in \mathcal{L}$, where $\mathcal{L}$ is a partial derivation language such that $\langle \mathcal{L} \rangle = \big(m, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$, means that z is a string of *terminal* symbols such that $A_0 \stackrel{*}{\Rightarrow} z'$, where z' is an interleaving of z and $A_1, \ldots, A_\ell$, in which A_j appears in coordinate $i_j + j - 1$. Indeed, there is a natural 1-1 correspondence between the indices $i_j \in [m]$ that are the locations in z in which the variables should be inserted, and the indices $i'_j \in [m + \ell]$, where $i'_j \stackrel{\text{def}}{=} i_j + j - 1$, that are the locations in the string $z' = z[1, i_1 - 1] \circ A_1 \circ z[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ z[i_\ell, m]$ in which the fixed variables appear.

Our construction of an IPP is recursive, and to facilitate the recursion, as discussed in Section 3.1.2.2, it will be useful for us to construct an IPP for *partial derivation languages* rather than just *context-free languages*. Additionally, as discussed in Section 3.1.2, the IPP will be proximity oblivious[26] (see Section 3.2.1.2). That is, we prove the following (more general) lemma:

Lemma 3.12. *Let G be a context-free grammar, let $\mathcal{L}$ be a* partial derivation language *corresponding to G, parameterized by $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$. For every integers $k \geq 2$ and $r \geq 1$ such that $k^r \leq n$, there exists an r-round proximity oblivious* IPP *for $\mathcal{L}$ with detection probability $\rho(\varepsilon) = \varepsilon$, communication complexity $O(rk \log(n + \ell))$ and query complexity $O\left(\frac{n+\ell}{k^r}\right)$. Furthermore, the proximity oblivious* IPP *is public-coin.*

Theorem 3.3 follows directly from Lemma 3.12 by observing that (1) every context-free language is a partial derivation language, without any fixed variables (i.e., $\ell = 0$), and (2) we can transform any proximity oblivious IPP into a standard IPP (by repeating the former $O(1/\varepsilon)$ times in parallel).

Lemma 3.12 is proved in Sections 3.4.1 and 3.4.2. Specifically, in Section 3.4.1, which contains the more involved (and interesting) part of the proof, we show a scheme for partitioning partial derivation languages into several smaller partial derivation languages. Then, in Section 3.4.2 we use this partition to construct an IPP for partial derivation languages (which is a fairly straightforward implementation of the outline presented in Sections 3.1.2.2 and 3.1.2.3), as well as describe the steps required to derive an MAP (thereby proving Theorem 3.1). Finally, in Section 3.4.3 we show how to improve the efficiency of the foregoing MAP for the Dyck languages (i.e., the languages of balanced parentheses expressions).

[26] In contrast to the case of ROBPS (see Section 3.3), here we can directly use Definition 3.2 without any modifications.

3.4.1 Partitioning Partial Derivation Languages

Let $\mathcal{L} \subseteq \Sigma^n$ be a partial derivation language[27] of a context-free grammar $G = (V, \Sigma, R, A_{\mathsf{start}})$, parameterized by $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$, and let $d = O(1)$ be the length of the longest production rule in R (so that every $x \in \mathcal{L}$ has a derivation tree with arity at most d).

In this section we describe a technique for partitioning $\mathcal{L}$ into several partial derivation languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ (of shorter strings), while preserving distances. That is, inputs x that belongs to $\mathcal{L}$ will be partitioned into k parts such that for every $j \in [k]$, the j^{th} part of x belongs to $\mathcal{L}_j$, whereas, for inputs x that are far from $\mathcal{L}$, the j^{th} part of x will be far from $\mathcal{L}_j$, for an average j. Later, in Section 3.4.2, we use this partition to construct an IPP for $\mathcal{L}$. (See Sections 3.1.2.2 and 3.1.2.3 for a high-level overview.)

The partition, which will be constructed jointly by the IPP prover and verifier, has two different representations. The first representation, which we call the *interval representation*, is a concise representation that is generated by the prover and sent to the verifier. The advantage of this representation is has a simple *syntactic* structure. The second representation, which is the actual partition, will be derived by the verifier from the interval representation. The main advantage of the latter representation is that it facilitates the verification of the *semantic* relation of the partition to the main input x.

We begin by describing the procedure that is used to generate the *interval representation* of the partition. The procedure, called `Generate-Intervals`(x, t), is given as input $x \in \mathcal{L}$ (recall that $\mathcal{L}$ is parameterized by $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$) and a parameter $t \in [n']$, where $n' \stackrel{\text{def}}{=} n + \ell$ and t specifies the desired size of each part in the partition. We assume for simplicity that $t \geq 2d$, and the case that $t < 2d = O(1)$ will be handled separately (and trivially) in Section 3.4.2. First, the procedure constructs[28] a derivation tree T corresponding to the derivation $A_0 \stackrel{*}{\Rightarrow} x'$, where $x' \stackrel{\text{def}}{=} x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$ ($A_0 \stackrel{*}{\Rightarrow} x'$ follows from the fact that $x \in \mathcal{L}$). Next, using Lemma 3.5, the procedure finds $k = O(n'/t)$ rooted subtrees[29] $T_1, \ldots, T_k$ of T such that (1) every vertex of T belongs to at least one of the subtrees, and (2) for each $i < j$ either T_i and T_j are disjoint or T_i is a subtree of T_j. The procedure outputs $\overline{I} = (I_1, \ldots, I_k) \in ([n']^2)^k$ and $\overline{B} = (B_1, \ldots, B_k) \in V^k$ where B_j is the label of the root of T_j and $I_j \subseteq [n']$ is the minimal interval that contains all the leaves of T_j, for every $j \in [k]$. Each pair of intervals is either disjoint or contained in one other. The `Generate-Intervals` procedure is detailed in Fig. 3.2.

To see that `Generate-Intervals` halts with $k \leq \frac{n'}{t/d-1} \leq 2d \cdot \frac{n'}{t}$ intervals, observe that in each iteration the number of leaves of the tree T' (defined in Step 3a) decreases

[27] We suggest to the reader to consider the case that $\mathcal{L}$ is a context-free language (i.e., no variables are fixed) at first reading, since it is somewhat simpler. However, we stress that we have to handle general partial derivation languages for the recursion to go through.

[28] Although our focus is not on computational complexity, we remark that such a derivation tree can be constructed in time $\mathsf{poly}(n')$, see [HMU06] for details.

[29] Recall that we define a *subtree* of a tree T as a tree consisting of a node in T together with *all* of its descendants, see Section 3.2.3.

`Generate-Intervals`(x,t)

Input: $x \in \mathcal{L}$ (where $\mathcal{L}$ is a partial derivation language parameterized by $\big(n,(i_1,\ldots,i_\ell),(A_0,\ldots,A_\ell)\big)$) and $t \in [2d,n']$, where $n' = n+\ell$.

1. Construct a derivation tree T of arity d, with n' leaves, corresponding to the derivation $A_0 \overset{*}{\Rightarrow} x[1,i_1-1] \circ A_1 \circ x[i_1,i_2-1] \circ \cdots \circ A_\ell \circ x[i_\ell,n]$ (according to the grammar G).
2. Set $j = 1$.
3. Repeat: (prior to the j^{th} iteration, we have already constructed subtrees $T_1,\ldots,T_{j-1}$ of T).
   (a) Construct a tree T' from T by removing all the vertices of $T_{j'}$ except for the root of $T_{j'}$, for every $j' \in [j-1]$. Note that there is a natural correspondence between the vertices of T' and the vertices of T from which they were copied.
   (b) If the number of leaves of T' is less than t, then exit the loop.
   (c) Applying Lemma 3.5 to T', with size parameter t, find a subtree of T' with t' leaves such that $t' \in [t/d,t]$. Denote the root of this subtree by v'. Let v be the vertex in T that corresponds to v', and define T_i as the subtree of T rooted at v.
   (d) Increment j by 1.
4. Set $k = j$ and $T_k = T$.
5. For every $j \in [k]$, let B_j be the label of (i.e., the variable associated with) the root of T_j, and let $I_j \subseteq [n']$ be the minimal interval that contains all the leaves of T_j in T.
6. Output $(\overline{I},\overline{B})$, where $\overline{I} = (I_1,\ldots,I_k)$ and $\overline{B} = (B_1,\ldots,B_k)$.

Figure 3.2: The `Generate-Intervals` Procedure for the Partial Derivation Language $\mathcal{L}$.

additively by at least $t/d-1$ and that we assumed that $t \geq 2d$.

As noted above, the output $(\overline{I},\overline{B})$ of `Generate-Intervals` is in the first representation of the partition, which we called the interval representation. Next, we show a transformation $\mathcal{T}$ (which will be applied by the IPP verifier) that transforms the interval representation of the partition into an actual partition of the main input x.

Actually, instead of partitioning the input x into parts $S_1,\ldots,S_k \subseteq [n]$, it will be more convenient to view the partition as a partition of the *terminal* coordinates of $x' = x[1,i_1-1] \circ A_1 \circ x[i_1,i_2-1] \circ \cdots \circ A_\ell \circ x[i_\ell,n]$.[30] That is, instead of a partition of $[n]$, we will find a partition of $[n']\backslash\{i'_1,\ldots,i'_\ell\}$, where $i'_j \overset{\text{def}}{=} i_j + j - 1$, for every $j \in [\ell]$ (indeed, the non-terminal coordinates of x' are precisely $\{i'_1,\ldots,i'_\ell\}$).

Our aim is to design a transformation $\mathcal{T}$ that maps $(\overline{I},\overline{B})$ into a partition $S_1,\ldots,S_k$ of $[n']\backslash\{i'_1,\ldots,i'_\ell\}$, where the parts have roughly the same length, together with (concise descriptions of) partial derivation languages $\mathcal{L}_1,\ldots,\mathcal{L}_k$ that satisfy the following

[30] Of course, the distinction disappears in the simpler case that $\mathcal{L}$ is a context-free language (i.e., $\ell = 0$).

conditions:

- **Completeness:** If $x \in \mathcal{L}$ and $(\overline{I}, \overline{B})$ is the output of `Generate-Intervals`(x, t), then $x'[S_j] \in \mathcal{L}_j$, for every $j \in [k]$.
- **Soundness:** If x is ε-far from $\mathcal{L}$, then for *every* $(\overline{I}, \overline{B}) \in ([n']^2)^k \times V^k$ it holds that $x'[S_j]$ is ε-far from $\mathcal{L}_j$, for an average $j \in [k]$ (where the average is weighted based on the lengths of the parts).

We begin with a high-level overview of the transformation $\mathcal{T}$ in the special and slightly simpler case that $\mathcal{L}$ is a context-free language (i.e., $\ell = 0$). In this case, given input $(\overline{I}, \overline{B})$, where $\overline{I} = (I_1, \ldots, I_k)$ and $\overline{B} = (B_1, \ldots, B_k)$, the transformation first constructs a partition of $[n]$ into k parts $S_1, \ldots, S_k$ by setting $S_j = I_j \setminus (I_1 \cup \cdots \cup I_{j-1})$, for every $j \in [k]$. The transformation outputs $S_1, \ldots, S_k$ as well as (concise) descriptions of k partial derivation languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ such that for every $j \in [k]$, the language $\mathcal{L}_j$ is a partial derivation language corresponding to a partial derivation starting from B_j into strings that have variables B_{j_i} at fixed coordinates corresponding to the relative position of all subintervals I_{j_i} of I_j. The transformation also checks that the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are non-empty so that the distance of $x'[S_j]$ from the corresponding language $\mathcal{L}_j$ is well defined (this check is indeed necessary — see discussion in Section 3.1.2).

The case that $\mathcal{L}$ is a partial derivation language (rather than a context-free language) is quite similar, where a fairly minor complication that arises is that we need to remove the non-terminal coordinates from the partition, and so we set $S_j = I_j \setminus (I_1 \cup \cdots \cup I_{j-1} \cup \{i'_1, \ldots, i'_\ell\})$. For technical reasons, it is more convenient for us to view each one of the non-terminal coordinates $i'_1, \ldots, i'_\ell$ as an additional *artificial* singleton interval. The transformation $\mathcal{T}$ is detailed in Fig. 3.3, and the *completeness* and *soundness* requirements (which were stated loosely above) are stated formally in the following two lemmas (Lemmas 3.13 and 3.14).

Lemma 3.13 (Completeness of $\mathcal{T}$)**.** *For every $x \in \mathcal{L}$ (where $\mathcal{L}$ is a partial derivation language parameterized by $\big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$) and parameter $t \in [2d, n']$, if $(\overline{I}, \overline{B}) \in ([n']^2)^k \times V^k$ is the output of* `Generate-Intervals`(x, t)*, then the transformation $\mathcal{T}(\overline{I}, \overline{B})$ does not reject, but rather outputs $\big((S_1, \langle \mathcal{L}_1 \rangle) \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$ such that for every $j \in [k]$:*

1. *$\mathcal{L}_j \subseteq \Sigma^{|S_j|}$ is a partial derivation language on strings of length $n_j = |S_j|$ with ℓ_j fixed variables such that $n_j + \ell_j \leq t$; and,*
2. *$x'[S_j] \in \mathcal{L}_j$, where $x' = x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$.*

Proof. Let $x \in \mathcal{L}$ and let $(\overline{I}, \overline{B})$ be the output of `Generate-Intervals`(x, t), where $\overline{I} = (I_1, \ldots, I_k)$ and $\overline{B} = (B_1, \ldots, B_k)$. Since $I_k = [n']$ and $B_k = A_0$, the transformation $\mathcal{T}(\overline{I}, \overline{B})$ does not reject, but rather outputs $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$. Let $I'_1, \ldots, I'_{\ell+k}$ be as defined in $\mathcal{T}$ (see Fig. 3.3).

The fact that, for every $j \in [k]$, it holds that $\mathcal{L}_j \subseteq \Sigma^{|S_j|}$ is a partial derivation language on strings of length n_j with ℓ_j fixed variables such that $n_j + \ell_j \leq t$ follows from the fact

The Transformation $\mathcal{T}(\overline{I}, \overline{B})$

Input: $\overline{I} = (I_1, \ldots, I_k) \in ([n']^2)^k$ and $\overline{B} = (B_1, \ldots, B_k) \in V^k$ (recall that $n' = n + \ell$ and that $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$).

1. Check that $(\overline{I}, \overline{B})$ is well formed: for every $j < i$ either $I_j \subsetneq I_i$ or $I_j \cap I_i = \emptyset$, and $I_k = [n']$ and $B_k = A_0$ (recall that $A_0 \in V$ is a variable such that all partial derivations in $\mathcal{L}$ start from A_0). If any test fails, then reject[a] and halt.
2. For $j \in [\ell]$, let $I'_j = \{i_j\}$.
3. For $j \in [k]$, let $I'_{\ell+j} = I_j$.
4. For every $j \in [k]$:
   (a) Let $I'_{j,1}, \ldots, I'_{j,\ell_j}$ be the ordered sequence (from left to right) of all maximal (strict) sub-intervals of $I_j = I'_{\ell+j}$ from the set of intervals $\{I'_1, \ldots, I'_{\ell+k}\}$. That is, all intervals (in order) from the set of intervals $\{I'_1, \ldots, I'_{\ell+k}\}$ that are strictly contained in I_j but are not contained in any other interval that is strictly contained in I'_j.[b]
   (b) Let $S_j = I_j \backslash (I'_{j,1} \cup \cdots \cup I'_{j,\ell_j})$.[c]
   (c) For every $s \in [\ell_j]$, let $i_{j,s} \in [|I_j|]$ be the *relative* starting position of the sub-interval $I'_{j,s}$ inside I_j, let $i'_{j,s} = i_{j,s} - \sum_{s'<s} |I'_{j,s'}|$, and let $B'_{j,s}$ be the label of the root of the subtree that corresponds to the interval $I'_{j,s}$. Define the following partial derivation language of G:
   $$\mathcal{L}_j \stackrel{\text{def}}{=} \left\{ w \in \Sigma^{|S_j|} \;:\; B_j \stackrel{*}{\Rightarrow} w[1, i'_{j,1}-1] \circ B'_{j,1} \circ w[i'_{j,1}, i'_{j,2}-1] \circ \cdots \circ B'_{j,\ell_j} \circ w[i'_{j,\ell_j}, |S'_j|] \right\}$$
   (see also Fig. 3.4). That is, $\langle \mathcal{L}_j \rangle = \big(|S_j|, (i'_{j,1}, \ldots, i'_{j,\ell_j}), (B'_{j,1}, \ldots, B'_{j,\ell_j})\big)$.
   (d) If $\mathcal{L}_j = \emptyset$, then reject and halt.[d]
5. Output $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$.

[a] In case the reader is bothered by the fact that the transformation may "reject", we can easily avoid rejecting by outputting instead some canonical representation of a "partition" that will always be rejected by the IPP verifier.

[b] In other words, an interval $I' \in \{I'_1, \ldots, I'_{\ell+k}\}$ is contained in the sequence if and only if $I' \subsetneq I_j$ and $I' \cap I'' \neq I'$, for every $I'' \in \{I'_1, \ldots, I'_{\ell+k}\} \backslash \{I'\}$ such that $I'' \subsetneq I_j$.

[c] Equivalently, $S_j = I_j \setminus (I'_1 \cup \cdots \cup I'_{\ell+j-1})$. We use the slightly more complicated definition to facilitate the proof.

[d] This check, which only requires access to $\langle L_j \rangle$ and the grammar G, can be done in $\mathsf{poly}(n')$ time.

Figure 3.3: The Transformation $\mathcal{T}$.

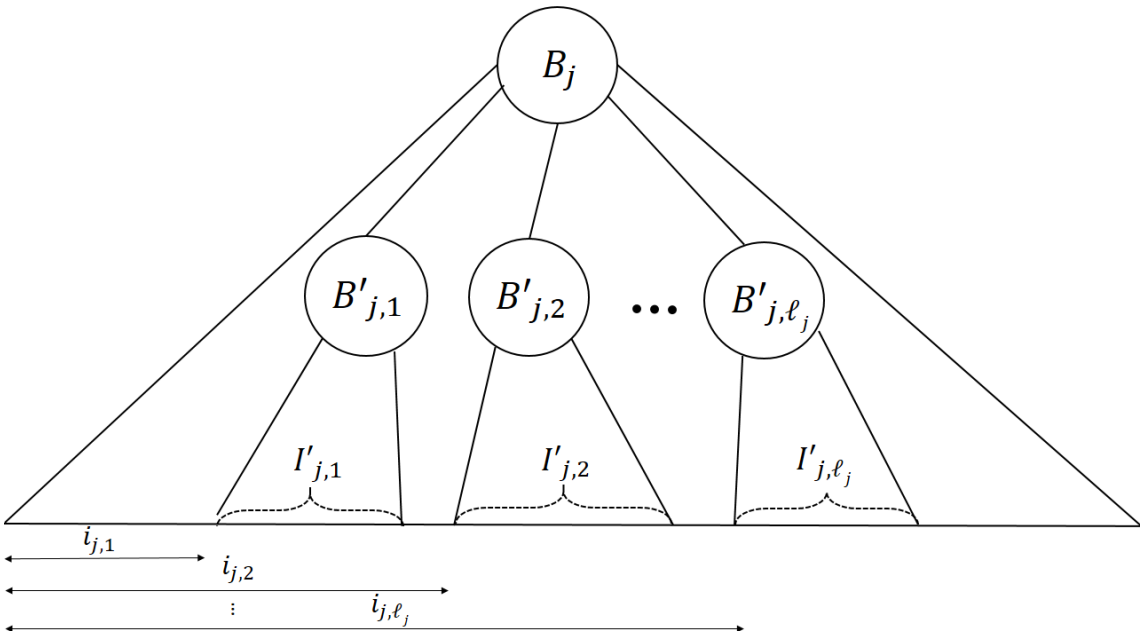

Figure 3.4: The partial derivation tree that describes the partial derivation $B_j \overset{*}{\Rightarrow} w[1, i'_{j,1} - 1] \circ B'_{j,1} \circ w[i'_{j,1}, i'_{j,2} - 1] \circ \cdots \circ B'_{j,\ell_j} \circ w[i'_{j,\ell_j}, |S'_j|]$.

that the quantity $n_j + \ell_j$ corresponds to the number of leaves of the subtree that was constructed in Item 3c in the `Generate-Intervals` procedure (recall that this subtree had at most t leaves).

To complete the proof of Lemma 3.13, we need to show that $x'[S_j] \in \mathcal{L}_j$, where $x' \overset{\text{def}}{=} x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$, for every $j \in [k]$. Let $j \in [k]$, and let ℓ_j, $i'_{j,1}, \ldots, i'_{j,\ell_j}, B'_{j,1}, \ldots, B'_{j,\ell_j}$ be as in Fig. 3.3. Let $w = x'[S_j]$, and observe that by construction,

$$B_j \overset{*}{\Rightarrow} w[1, i'_{j,1} - 1] \circ B'_{j,1} \circ w[i'_{j,1}, i'_{j,2} - 1] \circ \cdots \circ B'_{j,\ell_j} \circ w[i'_{j,\ell_j}, |S_i|].$$

Hence, $w \in \mathcal{L}_j$ and completeness follows.

□

Lemma 3.14 (Soundness of $\mathcal{T}$). *For every $\varepsilon \in [0, 1]$, every $x \in \Sigma^n$ that is ε-far from $\mathcal{L}$ (parameterized by $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$) and every $(\overline{I}, \overline{B}) \in ([n']^2)^k \times V^k$, it holds that $\mathcal{T}(\overline{I}, \overline{B})$ either rejects or outputs a sequence $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$ such that:*

1. *The sets $S_1, \ldots, S_k \subseteq [n'] \backslash \{i'_1, \ldots, i'_\ell\}$ form a partition of $[n'] \backslash \{i'_1, \ldots, i'_\ell\}$.*
2. *It holds that*

$$\mathop{\mathbf{E}}_{j \sim \mathcal{D}} \left[\Delta_{\mathsf{REL}} \left(x'[S_j], \mathcal{L}_j \right) \right] \geq \varepsilon,$$

where $x' = x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$ and $\mathcal{D}$ is a distribution over $[k]$ such that $\Pr_{j \sim \mathcal{D}}[j = j'] = |S_{j'}|/n$ for every $j' \in [k]$.

Proof. Let $x \in \Sigma^n$ and let $\overline{I} = (I_1, \ldots, I_k) \in ([n']^2)^k$ be a sequence of intervals and $\overline{B} = (B_1, \ldots, B_k) \in V^k$ a sequence of variables such that the transformation $\mathcal{T}(\overline{I}, \overline{B})$ does not reject and outputs $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$. Let $I'_1, \ldots, I'_{\ell+k}$ be as defined in $\mathcal{T}$ (see Fig. 3.3).

To see that $S_1, \ldots, S_k$ form a partition of $[n'] \setminus \{i'_1, \ldots, i'_\ell\}$, observe that for each $j \in [k]$, it holds that $S_j = I_j \setminus (I'_{j,1} \cup \cdots \cup I'_{j,\ell_j})$, where $I'_{j,1}, \ldots, I'_{j,\ell_j}$ are the ordered sequence (from left to right) of all maximal sub-intervals of I'_j out of $I'_1, \ldots, I'_{\ell+k}$ (i.e., all intervals that are contained in I_j but are not contained in any other interval that is strictly contained in I_j). Thus, the S_j's are disjoint. Furthermore, since $I'_{\ell+k} = [n']$, for every index $i \in [n']$ there exists $j \in [\ell + k]$ such that $i \in I'_j$. Hence, either $i \in \{i'_1, \ldots, i'_\ell\}$ (in case $j \in [\ell]$) or $i \in S_{j'}$ for some $j' \in [k]$, and so $S_1, \ldots, S_k$ form a partition of $[n'] \setminus \{i'_1, \ldots, i'_\ell\}$.

For every $j \in [k]$, let $\varepsilon_j = \Delta_{\mathsf{REL}}(x'[S_j], \mathcal{L}_j)$. Let $\mathcal{D}$ be the distribution as in the lemma's statement (i.e., $\Pr_{j \sim \mathcal{D}}[j = j'] = |S_{j'}|/n$, for every $j' \in [k]$). Suppose that $\mathbf{E}_{j \sim \mathcal{D}}[\varepsilon_j] < \varepsilon$, for some $\varepsilon \in [0, 1]$, where $x' = x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$. We will show that x is ε-close to $\mathcal{L}$.

For every $j \in [k]$, since the transformation *explicitly* checks[31] (in Step 4d) that $\mathcal{L}_j \neq \emptyset$, there exists a string $z_j \in \Sigma^{|S_j|}$ such that $z_j \in \mathcal{L}_j$ and $\Delta_{\mathsf{REL}}(x'[S_j], z_j) = \varepsilon_j$ (i.e., $z_j \in \mathcal{L}_j$ minimizes the distance of $x'[S_j]$ to $\mathcal{L}_j$).

Using $z_1, \ldots, z_k$, we construct a string $z \in \mathcal{L}$ that is ε-close to x as follows. Let $z \in \Sigma^n$ such that the string $z' = z[1, i_1 - 1] \circ A_1 \circ z[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ z[i_\ell, n]$ satisfies $z'[S_j] = z_j$, for every $j \in [k]$. (The fact that such a string z exists follows from the fact that $S_1, \ldots, S_k$ are a partition of $n' \setminus \{i'_1, \ldots, i'_\ell\}$.)

Observe that $\Delta_{\mathsf{REL}}(x, z) = \Delta_{\mathsf{REL}}(x', z') \leq \mathbf{E}_{j \sim \mathcal{D}}\left[\Delta_{\mathsf{REL}}(x'[S_j], z'[S_j])\right] = \mathbf{E}_{j \sim \mathcal{D}}[\varepsilon_j] < \varepsilon$ and so x is ε-close to z. By applying the following claim, with respect to $j = k$, and using the fact that the transformation explicitly checks that $I_k = [n']$ and $B_k = A_0$, we obtain that $A_0 \stackrel{*}{\Rightarrow} z'$, and therefore $z \in \mathcal{L}$. Hence x is ε-close to a string $z \in \mathcal{L}$, and soundness follows.

Claim 3.14.1. *For every $j \in [k]$, it holds that $B_j \stackrel{*}{\Rightarrow} z'[I_j]$.*

Proof. We prove the claim by induction on j. Let $j \in [k]$, and suppose that the claim holds for every $j' < j$. Let $y = z'[S_j]$. Note that $y \in \mathcal{L}_j$. We show that $B_j \stackrel{*}{\Rightarrow} z'[I_j]$.

Recall that $I'_1, \ldots, I'_{\ell+k}$ were fixed above as in Fig. 3.3. That is, for $j \in [\ell]$, it holds that $I'_j = \{i_j\}$, and for $j \in [\ell + 1, \ell + k]$ it holds that $I'_j = I_{j-\ell}$.

Let $I'_{j,1}, \ldots, I'_{j,\ell_j}$ be the ordered maximal sub-intervals (in the set $\{I'_1, \ldots, I'_{\ell+k}\}$) of I_j. By the construction of $\mathcal{T}$ it holds that

$$\mathcal{L}_j = \left\{ w \in \Sigma^{|S_j|} : B_j \stackrel{*}{\Rightarrow} w[1, i'_{j,1} - 1] \circ B'_{j,1} \circ w[i'_{j,1}, i'_{j,2} - 1] \circ \cdots \circ B'_{j,\ell_j} \circ w[i'_{j,\ell_j}, |S_j|] \right\},$$

where $i_{j,s}$ is the *relative* starting position of the interval $I'_{j,1}$ inside I_j, $i'_{j,s} \stackrel{\text{def}}{=} i_{j,s} - \sum_{s' < s} |I'_{j,s'}|$ and $B'_{j,s}$ is the label of the subtree that corresponds to the interval $I'_{j,s}$, for

[31] Indeed, this was the reason that we added this additional check, and without it soundness would not hold. See further discussion in Section 3.1.2.

every $s \in [\ell_j]$. Therefore, since $y \in \mathcal{L}_j$, it holds that

$$B_j \overset{*}{\Rightarrow} y[1, i'_{j,1} - 1] \circ B'_{j,1} \circ y[i'_{j,1}, i'_{j,2} - 1] \circ \cdots \circ B'_{j,\ell_j} \circ y[i'_{j,\ell_j}, |S_i|]. \tag{3.3}$$

On the other hand, for every $i \in [\ell_j]$, it holds that

$$B'_{j,s} \overset{*}{\Rightarrow} z'[I'_{j,s}], \tag{3.4}$$

where Eq. (3.4) follows from the inductive hypothesis and from the fact that $B'_{j,s} = A_{j,s}$ and $z'[I'_{j,s}] = z'_{j,s} = A_{j,s}$. for $s \in [\ell_j]$.

By combining Eq. (3.3), Eq. (3.4), and the definition of $i'_{j,s}$ we obtain that

$$B_j \overset{*}{\Rightarrow} y[1, i'_{j,1}] \circ z'[I'_{j,1}] \circ y[i'_{j,1}, i'_{j,2} - 1] \circ \ldots z'[I'_{j,\ell_j}] \circ y[i'_{j,\ell_j}, |S_i|].$$

The claim follows by observing that

$$z'[I_j] = y[1, i'_{j,1}] \circ z'[I'_{j,1}] \circ y[i'_{j,1}, i'_{j,2} - 1] \circ \ldots z'[I'_{j,\ell_j}] \circ y[i'_{j,\ell_j}, |S_i|],$$

and therefore $B_j \overset{*}{\Rightarrow} z'[I_j]$. □

This completes the proof of Lemma 3.14 □

3.4.2 IPP for Partial Derivation Languages

Using Lemmas 3.13 and 3.14, we complete the proof of Lemma 3.12 (which is a relatively straightforward implementation of the ideas outlined in Section 3.1.2).

Proof of Lemma 3.12. Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar. We construct a proximity oblivious IPP for every partial derivation language $\mathcal{L} \subseteq \Sigma^n$ of the grammar G.

The proximity oblivious IPP has two parameters: r which is the round complexity, and k which roughly corresponds to the amount of communication in each round. The IPP runs recursively, where each round of communication proceeds as follows. The (honest) prover uses the `Generate-Intervals` procedure on its input x and parameter $t = n'/k$ (where $n' = n+\ell$), to obtain $\left(\overline{I}, \overline{B}\right)$ and sends $\left(\overline{I}, \overline{B}\right)$ to the verifier. The verifier applies the transformation $\mathcal{T}(\overline{I}, \overline{B})$ to derive the partition $S_1, \ldots, S_k$ and the corresponding partial derivation languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$. Then, the verifier selects at random $j \in [k]$ and sends j to the prover (where j is distributed according to D as above). The two parties then recurse on input $x'[S_j]$, where $x' \overset{\text{def}}{=} x[1, i_1 - 1] \circ A_1 \circ x[i_1, i_2 - 1] \circ \cdots \circ A_\ell \circ x[i_\ell, n]$, with respect to the partial derivation language $\mathcal{L}_j$. The recursion stops once either:

1. $n' \leq O(k)$ (i.e., the input is very short), in which case the prover can send $x^* = x$ to the verifier.[32] Then, the verifier checks that $x^* \in \mathcal{L}$ and that x^* is consistent with x at a randomly selected coordinate; or,

[32] This check is to ensure that the parameter $t = n'/k$ is larger than $2d$.

2. r rounds have passed, in which case the verifier reads its entire input x (which has shortened by a multiplicative factor of roughly k in each step of the recursion) and verifies that $x \in \mathcal{L}$.

The IPP for $\mathcal{L}$, denoted CFL-IPP, is detailed in Fig. 3.5.

Without loss of generality, we can measure the complexity of the protocol only when the verifier interacts with the *honest* prover (see discussion in Section 3.2.1). It can be easily verified that the round complexity is at most r rounds. By Lemma 3.13, the protocol recurses on a partial derivation language $\mathcal{L}_j$ on strings of length n_j with ℓ_j fixed variables such that $n_j + \ell_j \leq n'/k$. Hence, after at most r rounds, the current input length has length at most n'/k^r, where $n' = n + \ell$, and so the query complexity of the IPP is $O(n'/k^r)$. Since in each round the communication is at most $O(k \log n')$, the communication complexity of the IPP is $O(rk \log n')$.

Completeness. Let $\mathcal{L}$ be a partial derivation language, with $\langle \mathcal{L} \rangle \stackrel{\text{def}}{=} \left(n, \bar{i}, \overline{A}\right)$, and let $x \in \mathcal{L}$. We show that perfect completeness hold by induction on r. For $r = 0$ or $n' = O(p)$, perfect completeness follows from the fact that $\mathcal{V}$ just checks that $x \in \mathcal{L}$. For $r > 1$ (with $n'/k \geq 2d$), by Lemma 3.13, the verifier produces $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$ such that $\mathcal{L}_j$ is a partial derivation language and $x'[S_j] \in \mathcal{L}_j$, for every $j \in [k]$ (in particular, $\mathcal{L}_j \neq \emptyset$). Hence, by the inductive hypothesis, the verifier in the $r - 1$ round protocol for $\mathcal{L}_j$ will accept on input $x'[S_j]$ with probability 1.

Soundness. Soundness follows from the following lemma, which is proved by induction on the number of rounds r.

Lemma 3.15. *Let $\mathcal{L}$ be a partial derivation language, and let $k \geq 1$ and $r \geq 0$. For every $\varepsilon \in [0, 1]$ and every x that is ε-far from $\mathcal{L}$, and for every cheating prover strategy P^* it holds that:*

$$\Pr\left[\left(V, P^*\right)(x) = 0\right] \geq \varepsilon,$$

where $\mathcal{V}$ is the verifier in $\mathsf{CFL\text{-}IPP}^{\mathcal{L}}_{r,p}$ (see Fig. 3.5).

Proof. We first consider the trivial case that $n' = O(k)$. In this case, if x^* is ε-close to x, then $x^* \notin \mathcal{L}$ (since x is ε-far from $\mathcal{L}$) and the verifier rejects with probability $1 \geq \varepsilon$. Otherwise, x^* is ε-far from $\mathcal{L}$ and the verifier rejects with probability at least ε when checking the consistency of x^* and x.

We proceed to the more interesting case, in which $n'/k > 2d$, and prove by induction on r. For $r = 0$, the verifier ignores the prover and reads all of x. Hence, if $B(x) \neq 1$, then the verifier rejects with probability 1.[33]

For $r \geq 1$, let $\varepsilon \in [0, 1]$, let $x \in \Sigma^n$ be ε-far from $\mathcal{L}$, and let P^* be a *deterministic* cheating prover strategy for the protocol $\mathsf{CFL\text{-}IPP}^{\mathcal{L}}_{r,k}$ of Fig. 3.5 (with r rounds). Let $(\overline{I}, \overline{B})$ be the first message sent by P^* to $\mathcal{V}$. Assume that the invocation of the transformation

[33] In the trivial case that $\varepsilon = 0$ (i.e., $B(x) = 1$), the verifier also satisfies the requirement, since it rejects with probability at least $0 = \varepsilon$.

The Protocol $\mathsf{CFL\text{-}IPP}^{\mathcal{L}}_{k,r}$

Parameters: $\mathcal{L} \subseteq \Sigma^n$ is a partial derivation language, with $\langle \mathcal{L} \rangle = \big(n, (i_1, \ldots, i_\ell), (A_0, \ldots, A_\ell)\big)$, the parameters $k, r \in \mathbb{N}$ correspond (roughly) to the amount of communication in each round and to the number of rounds, respectively. Let $n' = n + \ell$.

Prover's Input: Direct access to $x \in \mathcal{L}$, with $n \stackrel{\text{def}}{=} |x|$.

Verifier's Input: Oracle access to x, and direct access to $\langle \mathcal{L} \rangle$.

1. If $r = 0$, then the verifier $\mathcal{V}$ checks whether $x \in \mathcal{L}$ by explicitly reading all of x. If $x \in \mathcal{L}$, then $\mathcal{V}$ accepts, otherwise it rejects, and in either case both parties terminate the protocol.
2. If $n' = O(k)$, the prover sends $x^* = x$ to $\mathcal{V}$. The verifier $\mathcal{V}$ accepts if $x^* \in \mathcal{L}$ *and* x^* agrees with x at a randomly chosen coordinate. Otherwise $\mathcal{V}$ rejects, and in either case both parties terminate the protocol.
3. The Prover $\mathcal{P}$:
   (a) Invoke $\texttt{Generate-Intervals}(x, n'/k)$ to obtain $\big(\overline{I}, \overline{B}\big)$.
   (b) Send $\big(\overline{I}, \overline{B}\big)$ to $\mathcal{V}$.
4. The Verifier $\mathcal{V}$:
   (a) Invoke $\mathcal{T}\big(\overline{I}, \overline{B}\big)$. If the transformation rejects, then immediately reject and halt. Otherwise, denote the output of the transformation by $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$.[a]
   (b) Select $j \sim \mathcal{D}$, where $\mathcal{D}$ is the distribution in the statement of Lemma 3.14 (i.e., $\Pr_{j \sim \mathcal{D}}[j = j'] = |S_{j'}|/n$, for every $j' \in [k]$).
   (c) Send j to $\mathcal{P}$.
5. Both parties (recursively) invoke $\mathsf{CFL\text{-}IPP}^{\mathcal{L}_j}_{r-1,k}$ on input $x'[S_j]$.

[a] The reader may note that, in contrast to Fig. 3.1, the verifier does not check that $\mathcal{L}_j \neq \emptyset$, for every $j \in [k]$. This check is actually performed *within* the transformation $\mathcal{T}$ (see Step 4d in Fig. 3.3).

Figure 3.5: IPP for Context-Free Languages

$\mathcal{T}\left(\overline{I}, \overline{B}\right)$ does not reject (otherwise the verifier rejects with probability 1, and we are done), and denote its output by $\left((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\right)$.

For every $j \in [k]$, let $\varepsilon_j = \Delta_{\mathsf{REL}}\left(x'[S_j], \mathcal{L}_j\right)$ denote the relative distance of $x'[S_j]$ from $\mathcal{L}_j$, and let $\mathcal{D}$ be the distribution as in $\mathsf{CFL\text{-}IPP}^{\mathcal{L}}_{r,k}$. By Lemma 3.14, it holds that

$$\mathop{\mathbf{E}}_{j \sim \mathcal{D}}[\varepsilon_j] \geq \varepsilon. \tag{3.5}$$

For every $j \in [k]$, let P^*_j be the residual $r-1$ round strategy of P^* after receiving the message j from $\mathcal{V}$ in the first round, and let $\mathcal{V}_j$ be the residual strategy of $\mathcal{V}$ after fixing its first message to j. Observe that, by construction, $\mathcal{V}_j$ is simply the strategy of the verifier in the protocol $\mathsf{CFL\text{-}IPP}^{\mathcal{L}_j}_{k,r-1}$. Hence, by the inductive hypothesis, for every $j \in [k]$ it holds that

$$\Pr\left[\left(\mathcal{V}_j, \mathcal{P}^*_j\right)(x'[S_j]) = 0\right] \geq \varepsilon_j. \tag{3.6}$$

Using Eqs. (3.5) and (3.6) we obtain that:

$$\Pr[\left(V, P^*\right)(x) = 0] = \mathop{\mathbf{E}}_{j \sim \mathcal{D}}\left[\Pr[\left(\mathcal{V}_j, \mathcal{P}^*_j\right)(x'[S_j]) = 0]\right] \geq \mathop{\mathbf{E}}_{j \sim \mathcal{D}}[\varepsilon_j] \geq \varepsilon, \tag{3.7}$$

and the lemma follows. □

This concludes the proof of Lemma 3.12 and Theorem 3.3. □

Remark 3.16 (Computational Complexity)**.** *The* IPP *prover in Fig. 3.5 can be implemented in time* $\mathsf{poly}(n,k,r)$*. As for the* IPP *verifier, Step 4d in Fig. 3.3 can be implemented in time* $\mathsf{poly}(n)$*, and so we obtain a total running-time of* $\mathsf{poly}(n,k,r)$*, which is super-linear. We remark that for context-free languages whose* partial derivation languages *are themselves context-free languages, we can actually do better and obtain running time* $\mathsf{poly}(\log n, k, r)$ *(an example for such a context-free language is the language of balanced parentheses expressions, see Section 3.4.3). See Section 3.5.4 for details.*

Alternatively, by increasing the round complexity of our IPP*, we can also obtain sublinear time verification. The technique is similar to that described in Remark 3.7. More specifically, we can implement Step 4d in Fig. 3.3 (i.e., checking that a given partial derivation language is non-empty (which is the main bottleneck in our* IPP*)) via an interactive proof-system. To do so, we first construct a (logspace) uniform low-depth circuit that, given the description of a partial derivation language, outputs* 1 *if and only if the language is non-empty. An efficient interactive proof-system follows from the efficient interactive proof-system for low-depth computation of Goldwasser et al. [GKR08, Theorem 1]. Details follow.*

Fix the grammar $G = (V, \Sigma, R, A_{\mathsf{start}})$ *and consider a description* $(m, \overline{i}, \overline{A})$ *of a partial derivation language, where* $\overline{i} = (i_1, \ldots, i_\ell)$ *and* $\overline{A} = (A_0, \ldots, A_\ell)$*. Given* $(m, \overline{i}, \overline{A})$*, the circuit first constructs a string* $z \in (V \cup \{*\})^{m+\ell}$*, where* $'*'$ *is some character that does not belong to* $V \cup \Sigma$ *and* $z \stackrel{\text{def}}{=} *^{i_1-1} \circ A_1 \circ *^{i_2-i_1} \circ \cdots \circ A_\ell \circ *^{m-i_\ell+1}$*. The circuit then checks whether* z *can be derived from* A_0*, according to an auxiliary (unary) grammar* G'*, which is identical to* G *except that all the terminals are replaced by the unique terminal* $'*'$*. By*

a result of Ruzzo [Ruz81], membership in context-free languages can be computed by a (logspace uniform) NC_2 *circuit, and so we obtain a* $(O(\log(m) + \log(|\ell|))$*-space uniform) circuit that checks if the partial derivation language is non-empty, in depth* $\mathsf{polylog}(m+\ell)$ *and size* $\mathsf{poly}(m, \ell)$*.*

Given the above circuit, we can use [GKR08, Theorem 1] to obtain an interactive proof-system in which the verifier runs in $\ell \cdot \mathsf{poly}(\log(\ell), \log(m))$ *time and the prover runs in time* $\mathsf{poly}(m, \ell)$*. We note that using this proof-system inside our* IPP *increases the round complexity of our* IPP *by a poly-logarithmic factor.*

Remark 3.17 (MAPs for Context-Free Languages)**.** *Theorem 3.2 follows directly from the proof of Lemma 3.12, while noting that the two issues the arise in the case of* MAP*s for ROBPs (see Section 3.3.2) apply also here and can be resolved similarly.*

3.4.3 Improved MAPs for Specific Context-Free Languages

In this section we show that the efficiency of the MAPs for general context-free languages (i.e., Theorem 3.1) can be improved for context-free languages whose corresponding partial derivation languages have *efficient* testers (which do not use a proof). Most notably, we obtain such an improvement for the Dyck languages (i.e., languages of balanced parentheses expressions).

Recall that in the proof of Theorem 3.1, given the MAP proof, the MAP verifier (implicitly) constructs a partition $S_1, \ldots, S_k$ of $[n]$ and partial derivation languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$. Then, the verifier chooses an index $j \in [k]$ at random and checks whether $x[S_j] \in \mathcal{L}_j$ by explicitly reading all of $x[S_j]$. However, by Lemma 3.14, the MAP verifier does not really have to check that $x[S_j] \in \mathcal{L}_j$ *exactly*, but rather it suffices to check that $x[S_j]$ is *close* to $\mathcal{L}_j$. Since no non-trivial tester is known for general context-free languages (let alone for their corresponding partial derivation languages), we could not use this fact to our advantage in the general case. However, for some *specific* languages, such as the Dyck languages, more efficient testers are known and we can utilize them to improve the efficiency of our MAPs.

A technical difficulty that we encounter when taking this approach is that when testing whether $x[S_j]$ is close to $\mathcal{L}_j$ it is not a priori clear which value of the proximity parameter the verifier should use (recall that Lemma 3.14 only guarantees that $x[S_j]$ is ε-far for an *average* $j \in [k]$ but not necessarily for every $j \in [k]$). Of course, if $\mathcal{L}_j$ has a *proximity-oblivious tester*, then the issue is mute and we can just run the tester directly. In the more general case, we can simply apply an averaging argument. The naive averaging argument shows that for an $\varepsilon/2$ fraction of $j \in [k]$, it holds that $x[S_j]$ is $\varepsilon/2$ far from $\mathcal{L}_j$. However, by applying a more refined averaging argument, due to Levin [Lev85] (see [Gol14, Appendix A.2]), we obtain an additional improvement.

Lemma 3.18. *Let* G *be a context-free grammar and* $\alpha \geq 0$ *and* $\beta \geq 1$ *be constants. Suppose that every partial derivation language of* G *(as in Definition 3.11) has a property tester with query complexity* $O\left(m^{\alpha} \cdot \delta^{-\beta}\right)$ *for inputs of length* m *and proximity parameter* $\delta > 0$*. Then, for every* $k \geq 1$ *the language* $\mathcal{L}$ *has an* MAP *with proof complexity* $O(k \log n)$

and query complexity $O\big((n/k)^{\alpha} \cdot \varepsilon^{-\beta} \cdot \log^2(1/\varepsilon)\big)$. *Furthermore, if* $\alpha = 0$, *then the query complexity is at most* $O\big((n/k)^{1-1/\beta} \cdot \varepsilon^{-1} \cdot \log^3(1/\varepsilon)\big)$.

The MAP in Lemma 3.18 has one-sided error if and only if the testers for the partial derivation languages have one-sided errors. However, even if the resulting MAP has two-sided error, a one-sided error MAP (with only a poly-logarithmic overhead) can be obtained by applying a generic transformation from two-sided error MAPs into one-sided error MAPs (see of [GR13b, Theorem 4.3]).

Note that the alternative bound for $\alpha = 0$ improves over the general case only for sub-constant values of the proximity parameter (i.e., $\varepsilon < (n/k)^{-1/\beta} \cdot \mathsf{polylog}(n)$). The bound is obtained by observing that, for very small values of the proximity parameter, it is advantageous to read the entire input rather than apply the tester. We defer the proof of Lemma 3.18, which is relatively straightforward, to Section 3.5.3.

Using Lemma 3.18 we now show how to construct an improved MAP for the Dyck languages. Loosely speaking, the κ^{th}*-order Dyck language* consists of all of strings that form a balanced parenthesis expression with κ distinct types of parentheses. The Dyck languages can be defined via a context-free grammar as follows.

Definition 3.19. *Let* $\kappa \in \mathbb{N}$ *be a constant. The* κ^{th}-order Dyck language, *denoted* Dyck_κ, *is the language generated by the context-free grammar* $G_{\mathsf{Dyck}_\kappa} = (V, \Sigma_\kappa, R, A_{\mathsf{start}})$, *where* $V = \{A\}$, $A_{\mathsf{start}} = A$, $\Sigma_\kappa = \{`[_1', `]_1', `[_2', `]_2', \ldots, `[_\kappa', `]_\kappa'\}$, *and the production rules* R *consist of: (1)* $A \Rightarrow [_i\, A\,]_i$ *for every* $i \in [\kappa]$, *(2)* $A \Rightarrow AA$, *(3)* $A \Rightarrow \lambda$, *where* λ *denotes the empty string.*

Alon *et al.* [AKNS00] showed a tester (with two-sided error) for the first order Dyck language (i.e., Dyck_1) with query complexity $\tilde{O}(1/\varepsilon^2)$. As for higher order Dyck languages, Parnas *et al.* [PRR01] showed that any Dyck language (of any fixed order) can be tested (with two-sided error) by making $O(n^{2/3} \cdot \varepsilon^{-3})$ queries.[34] Furthermore, by the following proposition, the foregoing results can be extended to the case of *partial derivation languages* of the Dyck languages (with respect to the foregoing grammars).

Proposition 3.20. *Let* $m, \kappa \in \mathbb{N}$. *If* $\mathcal{L} \subseteq (\Sigma_\kappa)^m$ *is a partial derivation language of the grammar* G_{Dyck_κ}, *then* $\mathcal{L}$ *is equal to* $\mathsf{Dyck}_\kappa \cap (\Sigma_\kappa)^m$.

Proof. Let $\mathcal{L} \subseteq (\Sigma_\kappa)^m$ be a partial derivation language of G_{Dyck_κ} such that $\langle \mathcal{L} \rangle = \big(m, (i_1, \ldots, i_\kappa), (A, \ldots, A)\big)$ (here we used the fact that the grammar G_{Dyck_κ} uses only a single variable – A).

On one hand, if $x \in \mathcal{L}$, then $A \overset{*}{\Rightarrow} x[1, i_1 - 1] \circ A \circ x[i_1, i_2 - 1] \circ \cdots \circ A \circ x[i_\ell, m]$. Using the production rule $A \Rightarrow \lambda$ we have that $A \overset{*}{\Rightarrow} x[1, i_1 - 1] \circ x[i_1, i_2 - 1] \circ \cdots \circ x[i_\ell, m] = x$ and therefore $x \in \mathsf{Dyck}_\kappa \cap (\Sigma_\kappa)^m$.

On the other hand, if $x \in \mathsf{Dyck}_\kappa \cap (\Sigma_\kappa)^m$, then $A \overset{*}{\Rightarrow} x$. The following claim shows that, for the Dyck grammars, we can generate a partial derivation in which A is inserted in any desired sequence of positions. Therefore, $A \overset{*}{\Rightarrow} x[1, i_1 - 1] \circ A \circ x[i_1, i_2 - 1] \circ \cdots \circ A \circ x[i_\ell, m]$, which implies that $x \in \mathcal{L}$.

[34] For perspective, recall that Parnas *et al.* [PRR01] also showed that, for $\kappa \geq 2$, any tester (which does not use a proof) for Dyck_κ must make at least $\tilde{\Omega}(n^{1/11})$ queries.

Claim 3.20.1. *Let $\alpha \in (\Sigma_\kappa \cup \{A\})^{m'}$, for some $m' \in \mathbb{N}$, and let $i \in [m']$. If $A \overset{*}{\Rightarrow} \alpha$, then $A \overset{*}{\Rightarrow} \alpha[1, i-1] \circ A \circ \alpha[i, m']$.*

Proof. Since $A \overset{*}{\Rightarrow} \alpha$ (according to the grammar G_{Dyck_κ}), there exists a corresponding partial derivation tree T, in which all internal vertices are labeled by the variable A and each leaf is labeled by either 'A', '$[_j$', '$]_j$', for some $j \in [\kappa]$. We prove the claim by extending T into a partial derivation tree T' that corresponds to the partial derivation $A \overset{*}{\Rightarrow} \alpha[1, i-1] \circ A \circ \alpha[i, m']$.

Denote the i^{th} leaf of T by v and denote v's parent by u. The specific way in which T' is constructed from T depends on whether the label of v is 'A', '$[_j$' or '$]_j$' (for some $j \in [\kappa]$), and is detailed in Fig. 3.6.

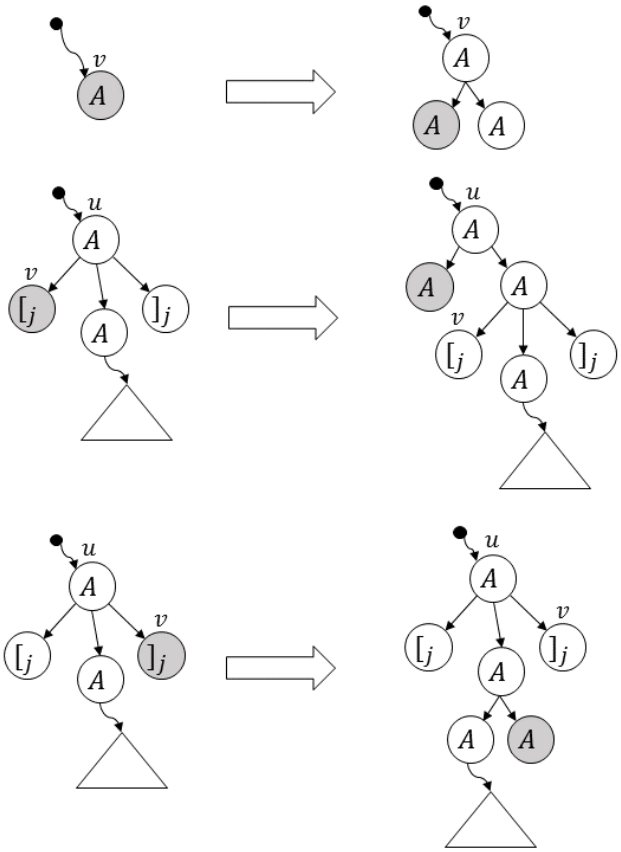

Figure 3.6: Construction of T' from T. The original tree T is on the left, and the new tree T' is on the right. In each case the i^{th} leaf of the tree has a shaded background, both in T and in T' (note that in all cases the i^{th} leaf of T is v and the i^{th} leaf of T' is labeled by A, the newly inserted symbol).

□

This concludes the proof of Proposition 3.20. □

Thus, the property testers of [PRR01] for the Dyck languages are also testers for the partial derivation languages (of the Dyck languages), and we obtain the following result.

Theorem 3.6. *Let $\kappa \geq 2$. For every p such that $2 \leq p \leq n$, there exists an* MAP *for* Dyck_κ *that uses a proof of length $O(p \log n)$ and has query complexity $O\big((n/p)^{2/3} \cdot \varepsilon^{-3} \cdot \log^2(1/\varepsilon)\big)$. Furthermore, there exists an* MAP *with* one-sided error *for* Dyck_κ *that uses a proof of length $O(p \log n + \mathsf{polylog}(n))$ and has query complexity $(n/p)^{2/3} \cdot \varepsilon^{-3} \cdot \mathsf{polylog}(n)$.*

The furthermore clause is obtained by applying the generic transformation from one-sided error MAP into two-sided error MAP (see [GR13b, Theorem 4.3]) and using the fact that without loss of generality we may assume that $\varepsilon \geq 1/n$ (and so $\log^2(1/\varepsilon) \leq \mathsf{polylog}(n)$). We conclude this section with some second order remarks.

Improvement for Dyck_1 and $\varepsilon \ll 1/\sqrt{n}$. For Dyck_1 (i.e., $\kappa = 1$), and for small values of the proximity parameter (i.e., $\varepsilon < \frac{1}{\sqrt{n} \cdot \mathsf{polylog}(n)}$) we can improve Theorem 3.6, by using the tester of Alon *et al.* [AKNS00] (which has query complexity $\tilde{O}(1/\varepsilon^2)$). Using the special case of Lemma 3.18, we obtain query complexity $O\big(\sqrt{n/p} \cdot \varepsilon^{-1} \cdot \log^3(1/\varepsilon)\big)$ with a proof of length $O(p \log n)$.

Extension to IPPs. The idea of applying non-trivial testers can also be used to obtain improved IPPs, by applying the tester after the last round of interaction (instead of running the trivial tester that reads the entire (current) input). The savings in this case are less significant since the query and communication complexities of our IPPs are already fairly small. Hence, we only elaborate briefly on these IPPs below.

If the partial derivation languages of the grammar have *proximity-oblivious* testers, then the latter can simply be employed in the last step of the recursion in Fig. 3.5. However, if only standard testers (which are not proximity oblivious) are available, then we can generalize the strategy in the proof of Lemma 3.18 by applying an averaging argument in each step of the recursion, while incurring an $\tilde{O}(1/\varepsilon)$ multiplicative overhead in each round. Unfortunately, the latter strategy results in an exponential dependence on the round complexity of the protocol.

Computational Complexity for Dyck Languages. In general, as noted in Remark 3.16, the running time of the verifier in Fig. 3.5 is $\mathsf{poly}(n)$ (because it verifies that each of the languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ is non-empty). However, as shown in Proposition 3.20, for the Dyck languages, the partial derivation languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are themselves Dyck languages. Since the Dyck language on m-bit strings is non-empty if and only if m is even, the running time of the verifier can be reduced to $\mathsf{poly}(\log n, k, r)$ (see also Section 3.5.4).

The MAP proof in Theorem 3.6 is generated efficiently (i.e., in time $\mathsf{poly}(n)$) for every context-free language, and in particular for the Dyck language. However, for the furthermore clause of Theorem 3.6, we apply the transformation of [GR13b, Theorem 4.3], which in general does not preserve *computational* efficiency of the proof generating procedure. Hence, we do not obtain an MAP for the Dyck languages that simultaneously has both one-sided error and an efficient procedure of generating the MAP proof.

3.5 Appendices for Chapter 3

3.5.1 Parallel Repetition of IPPs

The k-fold parallel repetition of an IPP $(\mathcal{V}_1, \mathcal{P}_1)$ is an IPP $(\mathcal{V}_k, \mathcal{P}_k)$ in which the two parties perform k parallel repetitions of $(\mathcal{V}_1, \mathcal{P}_1)$, using independent random coins for each invocation. Note that the query and communication complexities of $(\mathcal{V}_k, \mathcal{P}_k)$ are k times the query and communication complexities of $(\mathcal{V}_1, \mathcal{P}_1)$, respectively. The verifier $\mathcal{V}_k$ accepts if $\mathcal{V}_1$ accepts in a majority of the k invocations. For our applications it suffices to focus on the case that $(\mathcal{V}_1, \mathcal{P}_1)$ has a one-sided error, in which case $\mathcal{V}_k$ can just check that $\mathcal{V}_1$ accepts in *all* the k invocations.

It is clear that if $(\mathcal{V}_1, \mathcal{P}_1)$ has perfect completeness, then so does $(\mathcal{V}_k, \mathcal{P}_k)$. The main challenge is in proving that the soundness error decreases exponentially with k since if P^* is the optimal cheating strategy against $\mathcal{V}$, it is not a priori clear that the optimal cheating strategy against $\mathcal{V}_k$ is k independent copies of $\mathcal{P}^*$.

Nevertheless, the following lemma, taken verbatim from [Gol99, Lemma C.1] shows that the soundness error for any interactive machine $\mathcal{V}_k$ does decrease exponentially.

Lemma 3.21 ([Gol99, Lemma C.1])**.** *Let $\mathcal{V}_1$ be an interactive machine, and $\mathcal{V}_k$ be an interactive machine obtained from $\mathcal{V}_1$ by playing k versions of $\mathcal{V}_1$ in parallel. Let*

$$p_1(x) \stackrel{\text{def}}{=} \max_{\mathcal{P}^*} \{\Pr[(\mathcal{P}^*, \mathcal{V}_1)(x) = 1]\}, \text{ and}$$
$$p_k(x) \stackrel{\text{def}}{=} \max_{\mathcal{P}^*} \{\Pr[(\mathcal{P}^*, \mathcal{V}_k)(x) = 1]\}.$$

Then,

$$p_k(x) = (p_1(x))^k.$$

We stress that Lemma 3.21 holds for any x and is independent of the operation of $\mathcal{V}_1$. It holds as long as $\mathcal{V}_k$ executes k independent copies of $\mathcal{V}_1$ and accepts if all copies accept. Hence, it holds also when $\mathcal{V}_1$ is an IPP verifier; in that case $\mathcal{V}_k$ has query complexity that is k times that of $\mathcal{V}_1$.

3.5.2 Computing ROBPs in Low-Depth

For any branching program B (including branching programs that are not *read-once*), we show that the language $\mathcal{L}_B = \{x \in \{0,1\}^* : B(x) = 1\}$ can be recognized by a $\mathsf{poly}(|B|, n)$-size circuit of depth $O((\log(|B|))^2)$ (with fan-in 2). We stress that the branching program B is fixed and the circuit only gets x as input. For simplicity, we assume without loss of generality that B has a *unique* accepting sink (otherwise we can add a new unique accepting sink t and have all former accepting sinks direct to t).

The idea (which is in essence the folklore proof that (non-deterministic) log-space is contained in NC_2) proceeds as follows. First, based on the input x (and the fixed branching program B), compute a $|B| \times |B|$ matrix M_x whose $(u,v)^{\text{th}}$ entry is 1 if the branching program traverses from the vertex $u \in B$ to $v \in B$ on input x *in a single step*.

In addition, for every sink $t \in B$ we set the $(t,t)^{\text{th}}$-entry of M_x to 1 (these correspond to self loops). All other entries of M_x are set to 0. Given input x, the matrix M_x (which is a permutation matrix) can be computed by a constant-depth circuit of size $\mathsf{poly}(|B|)$ (in fact, every entry in M_x is either a fixed constant, or equal to some variable or its negation).

Observe that for every $k \geq 1$, the $(u,v)^{\text{th}}$-th entry of $(M_x)^k$ is equal to 1 if and only if the branching program traverses from u to v, on input x, in k steps (or at most k steps if v is a sink). Hence, to check whether the source s leads to the (unique) *accepting* sink t on input x, it suffices to check whether the $(s,t)^{\text{th}}$-th entry of $(M_x)^{|B|}$ is equal to 1. Using repeated squaring we can compute $(M_x)^{|B|}$ in $O(\log^2(|B|))$ depth and we obtain a circuit as required.

3.5.3 Proof of Lemma 3.18

We proceed to describe the MAP, which is similar to the MAP of Theorem 3.1 except that we use the guaranteed property testers for the partial derivation languages. Given $x \in \mathcal{L}$, the MAP proof is the output $(\overline{I}, \overline{B})$ of `Generate-Intervals`(x,t) (see Fig. 3.2), where $t = n/k$ and as in the proof of Theorem 3.1 we assume that $t \geq 2d$. The MAP verifier, given direct access to $(\overline{I}, \overline{B})$ and oracle access to $x \in \Sigma^n$, first runs $\mathcal{T}(\overline{I}, \overline{B})$ to obtain $(S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_\ell, \langle \mathcal{L}_\ell \rangle)$ and rejects if $\mathcal{T}$ rejects. Otherwise, the verifier runs the following procedure for every $j \in \left[\lceil \log_2(2/\varepsilon) \rceil \right]$:

1. Select uniformly at random $O\left(\frac{\log(1/\varepsilon)}{2^j \varepsilon}\right)$ indices in $[\ell]$. Denote the chosen indices by $\mathcal{I}$.

2. For every index $i \in \mathcal{I}$, run the property tester for $\mathcal{L}_i$ on input $x[S_i]$ (while simulating its oracle queries with queries to x), with respect to proximity parameter 2^{-j} and with completeness and soundness errors $\mathsf{poly}(\varepsilon)$ (as usual, the latter can be obtained by taking the majority of $O(\log(1/\varepsilon))$ independent tests). If the tester rejects then reject and halt.

If none of the above test fails then the verifier accepts.

We first show that completeness and soundness hold and later show that the query complexity is as stated.

Completeness. If $x \in \mathcal{L}$, by Lemma 3.13, the transformation $\mathcal{T}$ produces as output $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_k \rangle)\big)$ such that $\mathcal{L}_j$ is a partial derivation language and $x[S_j] \in \mathcal{L}_j$, for every $j \in [k]$. Since the tester for each partial derivation language $\mathcal{L}_j$ has completeness error $\mathsf{poly}(\varepsilon)$ and we perform Step 2 $O(\varepsilon^{-1} \cdot \log^2(1/\varepsilon))$ times in total, the verifier accepts in all tests with probability at least $2/3$. Furthermore, if the testers for the partial derivation languages have a one-sided error, then the MAP verifier accepts with probability 1 and otherwise we can apply a generic transformation (as discussed in the beginning of the proof) to obtain a one-sided error.

Soundness. Let $x \in \Sigma^n$ that is ε-far from $\mathcal{L}$, and let $(\overline{I}, \overline{B})$ be an alleged proof. By Lemma 3.14, the transformation $\mathcal{T}$ either rejects (in which case the verifier rejects and we are done), or produces $\big((S_1, \langle \mathcal{L}_1 \rangle), \ldots, (S_k, \langle \mathcal{L}_\ell \rangle)\big)$, where $S_1, \ldots, S_\ell$ form a partition of $[n]$ and $\mathcal{L}_j$ is a partial derivation language, such that x is ε-far from $\left\{z \in \Sigma^n \,:\, \forall j \in [k], z[S_j] \in \mathcal{L}_j\right\}$. The following claim, which is a refined averaging argument, shows that either there are many indexes $i \in [k]$ such $x[S_i]$ is mildly far from $\mathcal{L}_i$ or there are few indexes $i \in [\ell]$ such that $x[S_i]$ is extremely far from $\mathcal{L}_i$ (or anything in between).

Lemma 3.22 (Precision Sampling)**.** *There exists $j^* \in [\lceil \log_2 2/\varepsilon \rceil]$ such that for a $\frac{2^{j^*}\varepsilon}{4 \cdot \lceil \log_2(2/\varepsilon) \rceil}$ fraction of the indexes $i \in [\ell]$ it holds that $x[S_i]$ is 2^{-j^*}-far from $\mathcal{L}_i$.*

For completeness, we provide the proof of Lemma 3.22, which is standard.

Proof. Let $d \stackrel{\text{def}}{=} \lceil \log_2(2/\varepsilon) \rceil$. Recall that $\Delta_{\mathsf{REL}}(z, W)$ is the minimal *relative* Hamming distance of z from the set W. For every $k \in [d]$, let

$$B_k \stackrel{\text{def}}{=} \left\{ i \in [\ell] \,:\, \Delta_{\mathsf{REL}}(x[S_i], \mathcal{L}_i) \in \left(2^{-k}, 2^{-(k-1)}\right] \right\},$$

and let $B_{d+1} = [\ell] \backslash (\cup_{i \in [d]} B_k)$. Note that the sets $B_0, \ldots, B_d, B_{d+1}$ form a partition $[\ell]$. Also note that by our setting of d, for every $i \in B_{d+1}$ it holds that $x[S_i]$ is $\varepsilon/2$-close to $\mathcal{L}_i$.

Suppose towards a contradiction that for every $k \in [d]$ it holds that $|B_k| < \frac{2^k \varepsilon}{4d} \cdot \ell$. Using the fact that for every $i \in B_k$ it holds that $x[S_i]$ is $2^{-(k-1)}$-close to $\mathcal{L}_i$, we obtain that

$$\begin{aligned}
\Delta_{\mathsf{REL}}(x, \mathcal{L}) &\leq \frac{1}{\ell} \sum_{i=1}^{\ell} \Delta_{\mathsf{REL}}(x[S_i], \mathcal{L}_i) \\
&= \frac{1}{\ell} \sum_{i \in B_{d+1}} \Delta_{\mathsf{REL}}(x[S_i], \mathcal{L}_i) + \frac{1}{\ell} \sum_{k \in [d]} \sum_{i \in B_k} \Delta_{\mathsf{REL}}(x[S_i], \mathcal{L}_i) \\
&\leq \frac{|B_{d+1}|}{\ell} \cdot \frac{\varepsilon}{2} + \frac{1}{\ell} \sum_{k \in [d]} 2^{-(k-1)} \cdot |B_k| \\
&< \frac{\varepsilon}{2} + \sum_{k \in [d]} \frac{\varepsilon}{2d} \\
&= \varepsilon,
\end{aligned}$$

in contradiction to our assumption that x is ε-far from $\mathcal{L}$. □

Next, consider the execution of iteration j^* of the verifier, where j^* is as guaranteed by Lemma 3.22. Since the verifier selects uniformly at random $O\left(\frac{\log(1/\varepsilon)}{2^{j^*}\varepsilon}\right)$ indices in $[k]$, with probability at least $9/10$ it selects at least one index $i \in [k]$ such that $x[S_i]$ is 2^{-j^*}-far from $\mathcal{L}_i$. In this case, the tester for $\mathcal{L}_i$, with respect to proximity parameter 2^{-j^*} will reject $x[S_i]$ with probability $1 - \mathsf{poly}(\varepsilon)$. Thus, the verifier rejects x with probability at least $(1 - \mathsf{poly}(\varepsilon)) \cdot 9/10 \geq 2/3$.

Query Complexity. Recall that we assumed that every partial derivation language has a tester with query complexity $Q(m,\delta) = O(m^{\alpha} \cdot \delta^{-\beta})$, for inputs of length m with respect to proximity parameter $\delta > 0$. By definition, it holds that $|S_i| \leq t = n/p$, for every $i \in [\ell]$. Thus, the query complexity is at most

$$\begin{aligned}\sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \sum_{i \in \mathcal{I}} \left(\log(1/\varepsilon) \cdot Q\left(n/k, 2^{-j}\right)\right) &= O\left(\log(1/\varepsilon) \cdot \sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} 2^{j\beta} \cdot \frac{\log(1/\varepsilon)}{2^j \cdot \varepsilon} \cdot (n/k)^{\alpha}\right) \\ &= O\left((n/k)^{\alpha} \cdot \frac{(\log(1/\varepsilon))^2}{\varepsilon} \cdot \sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} 2^{(\beta-1)j}\right) \\ &= O\left((n/k)^{\alpha} \cdot \varepsilon^{-\beta} \cdot \log^2(1/\varepsilon)\right).\end{aligned}$$

For the particular case in which $\alpha = 0$, we tighten the analysis for small values of ε by noting that the query complexity for any language is upper bounded by the size of the object:

$$\begin{aligned}\sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \sum_{i \in \mathcal{I}} \left(\log(1/\varepsilon) \cdot Q\left(n/k, 2^{-j}\right)\right) &= O\left(\log(1/\varepsilon) \cdot \sum_{j \in [\lceil \log_2 2/\varepsilon \rceil]} \frac{\log(1/\varepsilon)}{2^j \cdot \varepsilon} \cdot \min\left(n/k, 2^{j\beta}\right)\right) \\ &= O\left((n/k)^{1-1/\beta} \cdot \varepsilon^{-1} \cdot \log^3(1/\varepsilon)\right),\end{aligned}$$

where the last equality follows since $\min\left(n/k, 2^{j\beta}\right) \leq (n/k)^{1-1/\beta} \cdot \left(2^{j\beta}\right)^{1/\beta}$, for every $j \geq 1$ (while using the fact that $\beta \geq 1$). Note that $\log^3(1/\varepsilon) \leq \mathsf{polylog}(n)$ since without loss of generality we may assume that $\varepsilon \geq 1/n$.

3.5.4 Efficient Verification for Special Context-Free Languages

As stated in Remark 3.16, in this section we show that for special context-free languages we can improve the running time of the verifier in Fig. 3.5 from $\mathsf{poly}(n,k,r)$ to $\mathsf{poly}(\log n,k,r)$. Specifically, we refer to context-free languages whose *partial derivation languages* are themselves context-free languages (e.g., the Dyck language, see Proposition 3.20).

The crucial step in improving the verifier's running-time is an efficient implementation of Item 4d in Fig. 3.3. In the general case, this step can be implemented in time $\mathsf{poly}(n)$, but we show that if the partial derivation languages are context-free languages, then we obtain running time $\mathsf{polylog}(n)$.

Lemma 3.23. *For every context-free language $\mathcal{L}$ over an alphabet Σ, there exist an algorithm that given an integer $n \in \mathbb{N}$, runs in time $\mathsf{polylog}(n)$ and accepts if and only if $\mathcal{L} \cap \Sigma^n \neq \emptyset$.*

Proof. Let G be a context-free grammar that accepts $\mathcal{L}$, and let G' be the context-free grammar that is obtained from G by replacing all the terminal symbols in G by a single terminal symbol, denoted 0. Note that $\mathcal{L} \cap \Sigma^n \neq \emptyset$ if and only if G' accepts 0^n.

Observe that the language $\mathcal{L}'$ accepted by G' is a *unary* context-free language. Ginsburg and Rice [GR62] showed that such a language must be *regular*.

Proposition 3.24 ([GR62])**.** *Every* unary *context-free language is* regular.

Hence, there exists a finite-state automaton over the unary alphabet that accepts $\mathcal{L}'$. Such an automaton can be viewed as a directed graph with a single outgoing edge from each node. Hence, the graph is a directed path (from the start node) of length a feeding into a directed cycle of length b, and some of the nodes are accepting. Hence, the accepted lengths have the form $j + i \cdot b$, where $j \in [a + b - 1]$ and $i \geq 0$.

The lemma follows by observing that an algorithm can easily check in $\mathsf{polylog}(n)$ time if the given input n has the desired form, by checking if $n - j$ is divisible by b, for the specific set of $j \in [a + b - 1]$ that correspond to accepting nodes of the automaton. □

Chapter 4

A Hierarchy Theorem for Interactive Proofs of Proximity

4.1 Introduction

Interactive Proofs, introduced by Goldwasser at el. [GMR89] (and in their public-coin form, by Babai and Moran [BM88]), are protocols in which a computationally unbounded prover tries to convince a verifier that an input x belongs to a language $\mathcal{L}$. A recent line of work, initiated by Ergün, Kumar and Rubinfeld [EKR04] and more recently by Rothblum, Vadhan and Wigderson [RVW13], considers a variant of interactive proofs in which the verifier is required to run in *sublinear* time. Since the verifier does not have enough time to even read its entire input, we cannot expect it to reject every false statement. Rather, following the property testing literature [RS96, GGR98] (see also [Gol16]), we relax the soundness condition and only require that the verifier reject inputs that are *far* from the language (no matter what cheating strategy the prover uses). Since the verifier is only assured that the input is close to the language, such interactive proofs are called *interactive proofs of proximity* (IPPs). Indeed, IPPs may be thought of as the property testing analogue of interactive proofs.

From an information theoretic perspective, the key parameters of an IPP are its *query complexity*, *communication complexity* and *round complexity*. The *query complexity* is the number of bits of the input string that the verifier reads. The *communication complexity* is the number of bits exchanged between the prover and the verifier, and the *round complexity* is the number of rounds of interaction. We think of all of these parameters as being *sublinear* in the input length. Additional computational parameters that we aim to minimize are the verifier's running time (which should also be sublinear) and the prover's running time (which, ideally, should be proportional to the complexity of deciding the language).

In this chapter we focus on the round complexity of IPPs, and on the relation between the number of rounds and the other parameters. Specifically, we ask the following question:

Does the power of Interactive Proofs of Proximity grow with the number of rounds?

Understanding the round complexity of protocols is a central problem in the theory of computation (most notably in complexity theory and cryptography). Some of the main motivations for reducing round complexity are considerations such as network latency, the need to stay online or to synchronize messages between the parties, and the overhead involved in sending and receiving messages.

4.1.1 Our Results

Our main result answers the foregoing question by showing a hierarchy of IPPs: we show that for a gap function $g(r) = \Theta(r^2)$, and for every constant $r \geq 1$, it holds that r-round IPPs can be outperformed by $g(r)$-round IPPs, in the sense that the verifier in the latter system is significantly more efficient. We prove our hierarchy theorem by constructing a *single* explicit language for which the power of IPPs grows with the number of rounds.

Theorem 1 (Hierarchy theorem, informally stated (see Theorem 4.1))**.** *There exists an explicit language $\mathcal{L}$ such that for every constant $r \geq 1$ and for inputs of length n:*

1. *There is an $O(r^2)$-round* IPP *for $\mathcal{L}$ in which the verifier runs in time $t = n^{O(1/r)}$; and*
2. *The verifier in* any *r-round* IPP *for $\mathcal{L}$* must *run in time at least $t' = t^{100}$ (where the constant* 100 *is arbitrary). Furthermore, either the communication complexity or query complexity of the verifier must be at least t'.*

Thus, we obtain a characterization (which is exact, up to the specific polynomial of the gap function g) of the complexity of constant-round IPPs for the language $\mathcal{L}$.

For simplicity, the statement in Theorem 1 is restricted to constant-round protocols. However, the complexity of the IPP protocol in Theorem 1 actually reduces further as the round complexity grows to be super-constant. In particular, we obtain a poly-logarithmic round IPP for $\mathcal{L}$ with poly-logarithmic communication and query complexities, and an $\omega(1)$-round IPP with $n^{o(1)}$ communication and query complexities. Together with the lower bound in Theorem 1, these yield a separation between the power of constant-round IPPs and super-constant round IPPs, and a *sub-exponential* separation with respect to poly-logarithmic round IPPs.

Theorem 2 (Constant Round versus General IPPs)**.** *There exists a language $\mathcal{L}$ that has a* polylog(n)*-round* IPP *with a* polylog(n) *time verifier and an $\omega(1)$-round* IPP *with $n^{o(1)}$ time verifier, but for every constant $r \geq 1$, the verifier in any r-round* IPP *for $\mathcal{L}$ must run in time at least $n^{\Omega(1/r)}$.*

Prior to this work, only a separation between the power of MAPs (which are *non-interactive* IPPs, i.e., the entire "interaction" consists of a *single* message) and IPPs was known [GR13b].

We remark that Theorems 1 and 2, and their proofs, shed new light also on standard interactive proofs (in which the verifier is given direct access to the input and can run in polynomial time). We proceed to discuss such implications.

Optimality of the Babai-Moran Round Reduction. Following Vadhan [Vad00], we consider *black-box transformations on interactive proofs*, which are transformations that take prover and verifier strategies $(\mathcal{P}, \mathcal{V})$, for an interactive-proof for some language $\mathcal{L}$, and output new strategies $(\mathcal{P}', \mathcal{V}')$, for the same language $\mathcal{L}$, such that new prover and verifier strategies can only make oracle calls to the original strategies $(\mathcal{P}, \mathcal{V})$. More specifically, the new verifier $\mathcal{V}'$ is only allowed to make oracle calls to $\mathcal{V}$ (and in particular does not have direct access to the input) and $\mathcal{P}'$ may make oracle calls to both $\mathcal{V}$ and $\mathcal{P}$.[1]

As pointed out by Vadhan, many (but not all) of the known transformations on interactive proofs from the literature are in fact black-box. We focus on such a transformation, due to Babai and Moran [BM88], for reducing the number of rounds of interaction in public-coin interactive proofs. Using our hierarchy theorem, we show that the overhead incurred by the round reduction transformation of [BM88] is close to optimal among all black-box transformations.

Algebrization of Interactive Proofs. As our second application, we show a connection between our hierarchy theorem and the *algebrization* framework of Aaronson and Wigderson [AW09]. This framework, which is an extension of the *relatization* framework of Baker, Gill, and Solovay [BGS75], is viewed as a barrier to proving complexity-theoretic lower bounds using currently known proof techniques. Loosely speaking, [AW09] show that almost all known complexity theoretic results "algebrize" (i.e., fall within their framework), whereas making progress on some of our most fundamental questions (such as $\mathcal{P} \neq \mathsf{NP}$) requires non-algebrizing techniques.

Using our hierarchy theorem for IPPs, we show that any proof of the complexity class inclusion $\#\mathcal{P} \subseteq \mathsf{AM}$ (which is widely disbelieved, and in particular implies the collapse of the polynomial hierarchy) must make use of non-algebrizing techniques, and therefore *must* introduce a fundamentally different proof technique. A conceptual connection between our results and interactive proofs in the algebrization framework is further discussed in Section 4.1.3 and elaborated on in Section 4.5.

4.1.2 Technical Overview

Loosely speaking, the language for which we prove the round hierarchy theorem consists of error-correcting encodings of strings $x \in \{0,1\}^k$ whose Hamming weight $\mathsf{wt}(x) \stackrel{\text{def}}{=} \sum_{i \in [k]} x_i$, is divisible by 3 (i.e., $\mathsf{wt}(x) = 0 \pmod 3$).

The specific encoding that we use is the low degree extension code $\mathsf{LDE} : \mathbb{F}^k \to \mathbb{F}^n$, over

[1] One could also restrict $\mathcal{P}'$ to make only oracle calls to $\mathcal{P}$ (and not to $\mathcal{V}$ as we do). However, giving $\mathcal{P}'$ more freedom only makes our results stronger (since we rule out the broader class of transformations).

a field $\mathbb{F}$ that is an extension field of $\mathsf{GF}(2)$.[2] Indeed, it is crucial that the characteristic of $\mathbb{F}$ is different than the modulus 3. The parameter k (which specifies the message length) is the same as in the preceding paragraph, where we view $\{0,1\}^k$ as a subset of the message space $\mathbb{F}^k$.

Before proceeding, we note that throughout this chapter we use the standard convention that codes map messages of length k to codewords of length $n = \mathsf{poly}(k)$. In particular, this will mean that inputs to IPPs, which will typically refer to (possibly corrupt) codewords, have length n, whereas inputs to other types of protocols and sub-routines, may refer to the underlying messages, which have length k.

Recall that the LDE code is parameterized by a finite field $\mathbb{F}$, a subset of the field $H \subseteq \mathbb{F}$ and a dimension m. To encode a message $x \in \{0,1\}^k$, where $k = |H|^m$, we view the message as a function $x : H^m \to \{0,1\}$ (by associating $[k]$ with H^m) and consider the unique individual degree $|H|-1$ polynomial $P : \mathbb{F}^m \to \mathbb{F}$ that agrees with x on H^m. We denote this polynomial by $P = \mathsf{LDE}_{\mathbb{F},H,m}(x)$. For the time being, the sizes of $|\mathbb{F}|$, $|H|$ and m should all be thought of as at most poly-logarithmic in n. (See Section 4.2.3 for additional details about the LDE encoding.)

Thus, the language for which we prove our round hierarchy, which we denote by Enc-MOD3, consists of all polynomials $P : \mathbb{F}^m \to \mathbb{F}$ of individual degree $|H|-1$ that obtain Boolean values in the subcube H^m, such that these Boolean values sum up to 0 (mod 3). That is, all polynomials P such that $P|_{H^m} : H^m \to \{0,1\}$ and $\sum_{z \in H^m} P(z) \equiv 0 \pmod 3$.

We prove our hierarchy theorem by showing that for every constant $r \geq 1$, the language Enc-MOD3 has an $O(r^2)$-round IPP in which the verifier runs in time roughly $n^{O(1/r)}$, and that the verifier in *any* r-round IPP for Enc-MOD3 must run in time at least $n^{\Omega(1/r)}$, where the constant in the Ω-notation can be made arbitrarily larger than the constant in the O-notation. In Section 4.1.2.1 we give an overview of the upper bound, which is technically more involved, and then, in Section 4.1.2.2 we give an overview of the lower bound.

4.1.2.1 Upper Bound

Our goal is to construct an IPP in which the verifier is given oracle access to a function $f : \mathbb{F}^m \to \mathbb{F}$ and needs to verify that f is close to a polynomial of individual degree $|H|-1$ that obtains only Boolean values in H^m such that their sum modulo 3, over the subcube H^m, is 0. The verifier may interact with the prover for $O(r^2)$ rounds.

As its initial step, our verifier checks that the given input f is close to some low degree polynomial by invoking the low degree test. This test, introduced by Rubinfeld and Sudan [RS96], ensures that if f is far from every low degree polynomial, then the verifier will reject with high probability. Thus, we can assume that f is close to some low degree polynomial. Moreover, using the self-correction property of polynomials, this means that with a small overhead, we can treat f as though it were itself a low degree

[2] We remark that a similar result could be obtained if we replaced the modulus 3 and the field's characteristic by any two *distinct* and *constant-sized* primes.

polynomial (rather than just being close).[3]

Given this initial step, we can now assume without loss of generality that the function $f : \mathbb{F}^m \to \mathbb{F}$ is in fact a low degree polynomial. However, the verifier still needs to check that $\sum_{z \in H^m} f(z) = 0 \pmod 3$ and that $f|_{H^m} : H^m \to \{0,1\}$. For now though, let us focus on the former task, which is the main step in our proof: checking that $\sum_{z \in H^m} f(z) = 0 \pmod 3$ (and we just assume that $f|_{H^m} : H^m \to \{0,1\}$).

Viewing $f|_{H^m}$ as a string $x \in \{0,1\}^k$, we need to construct an interactive proof in which the verifier uses oracle access to $\mathsf{LDE}(x)$ to verify that $\mathsf{wt}(x) = 0 \pmod 3$ in sublinear time. We refer to this type of proof-system, in which the verifier is given oracle access to an *encoding* of the input and runs in sublinear time, as a *holographic*[4] *interactive proof* (HIP).

More precisely, we say that a language has an HIP, with respect to some error-correcting code C, if it has an interactive proof in which the verifier has oracle access to an encoding under C of the input and verifies membership in the language using few queries to this encoding. The redundant representation of the input often allows the verifier to run in *sub-linear time*. We remark that HIPs play a central role in this chapter and we discuss them more in Section 4.1.3.

Thus, our task is now to construct an HIP (with respect to the LDE code) for the language

$$\mathcal{L}_{\mathsf{MOD3}} \stackrel{\text{def}}{=} \left\{ x \in \{0,1\}^k \ : \ \mathsf{wt}(x) = 0 \pmod 3 \right\}.$$

Before describing the construction of an HIP for $\mathcal{L}_{\mathsf{MOD3}}$, it will be instructive to consider as a warm-up, the construction of an HIP for the related language $\mathcal{L}_{\mathsf{MOD2}} = \{x \in \{0,1\}^k : \mathsf{wt}(x) = 0 \pmod 2\}$, where the important distinction is that the modulus 2 is also the characteristic of the field $\mathbb{F}$ under which x is encoded.

In this warmup case, we assume that the verifier is given oracle access to a polynomial $f : \mathbb{F}^m \to \mathbb{F}$ that obtains Boolean values in H^m (i.e. $f|_{H^m} : H^m \to \{0,1\}$), and needs to check that $\sum_{z \in H^m} f(z) = 0$, where the sum is over $\mathsf{GF}(2)$. Importantly, since we assumed that $f|_{H^m}$ is Boolean valued, and that the field $\mathbb{F}$ has characteristic 2, we can instead take the sum over the field $\mathbb{F}$ (rather than taking the integer sum mod 2).

The latter problem, of checking whether the sum of a given input polynomial is 0 over a subcube of its domain (i.e., over H^m), has a well-known interactive proof due to Lund *et al.* [LFKN92], which is often referred to as the *sumcheck protocol*. In this protocol the verifier only needs to query the polynomial f at a single point and so it can be viewed as an HIP. Furthermore, there are known variants of the sumcheck protocol that offer a suitable tradeoff between the number of rounds and verifier's complexity, which suffice for our purposes (i.e., an r-round IPP with verification time roughly $n^{1/r}$).

[3] Loosely speaking, the self-correction property of polynomials says that if f is guaranteed to be close to a low degree polynomial P, then one can read values from P by only making few queries to f. See Lemma 4.23 for the precise statement.

[4] The terminology of "holographic" interactive proofs originates from the "holographic proofs" of Babai *et al.* [BFLS91], which refers to probabilistic proof systems for *encoded* inputs. The notion of HIP, and its relation to other notions, is further discussed in Section 4.1.3.

The aforementioned variants of the sumcheck protocol suffice for an upper bound for the warmup case. However, we do not know how to prove a corresponding lower bound, which is the reason that we set the modulus in our construction to be different from the field's characteristic.[5] While the original language $\mathcal{L}_{\mathsf{MOD3}}$ allows us to prove the desired lower bound, unfortunately it makes obtaining an upper bound more challenging. We proceed to the actual problem at hand: constructing an HIP (with respect to the LDE code over a field of characteristic 2) for checking $\mathcal{L}_{\mathsf{MOD3}}$.

Since the modulus and characteristic are different, our task can no longer be expressed as a linear constraint (over $\mathbb{F}$) on the bits of x. Since we do not know how to solve this problem directly using the sumcheck protocol, we turn to more complex interactive proofs from the literature. Specifically, our starting point will be the interactive proof-system of Goldwasser, Kalai and Rothblum [GKR08].[6]

The [GKR08] Protocol. Goldwasser *et al.* give an interactive proof for any language computable by a logspace-uniform circuit of size S and depth D such that the number of rounds in their protocol is $D \cdot \mathsf{polylog}(S)$, the communication is also $D \cdot \mathsf{polylog}(S)$, and the verifier runs in time $(n + D) \cdot \mathsf{polylog}(S)$. Their protocol is based on algebraic techniques and, in particular, uses ideas originating from the interactive proof and PCP literature (cf., [Sud92]). Our HIP for MOD3 will be based on a variant of their proof system.

Observe that one can check whether a given string x's Hamming weight is divisible by 3 using a highly uniform logarithmic-depth formula.[7] Thus, applying the [GKR08] result gives us an interactive proof for $\mathcal{L}_{\mathsf{MOD3}}$. Most importantly for our purposes, if the [GKR08] verifier is given oracle access to the LDE encoding of the input, then it only needs to check a single (random) element from the encoding (and in particular runs in sublinear time). In other words, the [GKR08] protocol can be thought of as an HIP with *sublinear* time verification.[8] While the [GKR08] protocol does yield an HIP for $\mathcal{L}_{\mathsf{MOD3}}$, its round complexity is poly-logarithmic and therefore too large for our purposes (recall that we are aiming for constant round protocols). The large round complexity is due to the fact that the high-level strategy in the [GKR08] protocol is to process the circuit layer by layer, where the transition between each two consecutive layers uses an interactive protocol, which itself is based on the sumcheck protocol.

Even if we were to use a constant-round variant of the sumcheck protocol for each transition, the [GKR08] protocol still uses $\Omega(D)$ rounds, where D is the depth of the

[5] We conjecture that for the warmup case (i.e., when the modulus is 2) a lower bound that (roughly) corresponds to the upper bound given by the sumcheck protocol does hold.

[6] In Section 4.6.1.1 we discuss our reason for basing our protocol on the [GKR08] proof-system, rather than other general purpose interactive proof-systems from the literature.

[7] E.g., consider the $\log(k)$-depth full binary tree with the input bits at its leaves, in which each internal vertex computes the sum modulo 3 of its two children, where each such modulo 3 sum can be computed by a simple constant size gadget composed of AND, OR and NOT gates.

[8] Note that obtaining an interactive-proof for $\mathcal{L}_{\mathsf{MOD3}}$ with a *linear-time* verifier is trivial, since the verifier can decide membership by itself in linear-time. The key benefit that we get from using the [GKR08] protocol is that it allows for *sublinear* time verification given access to an encoded input.

circuit, which in our case is logarithmic and therefore too large. To get around this, we rely on an unpublished observation, due to Kalai and Rothblum [KR09], which shows that for every constant $r \geq 1$, if the circuit satisfies an extreme (and somewhat unnatural) uniformity condition[9], then $\log(n)/r$ layers can be processed at once, using r rounds of interaction and roughly $n^{1/r}$ communication. Thus, overall, a logarithmic depth circuit can processed in $O(r^2)$ rounds. Using this observation, [KR09] obtain *constant-round* interactive-proofs for all languages in NC^1 that satisfy the aforementioned uniformity condition.[10]

In our actual construction we do not use the [KR09] protocol directly (even though the language $\mathcal{L}_{\mathsf{MOD3}}$ satisfies the desired uniformity), but rather give a special purpose protocol tailored for $\mathcal{L}_{\mathsf{MOD3}}$ (which is inspired by their techniques). Doing so allows us to avoid stating their somewhat cumbersome uniformity condition and to introduce other simplifications (due to the simple and regular structure of the formula for $\mathcal{L}_{\mathsf{MOD3}}$). We proceed to describe this HIP.

A Holographic Interactive Proof for $\mathcal{L}_{\mathsf{MOD3}}$. Recall that we are given oracle access to a polynomial $X : \mathbb{F}^m \to \mathbb{F}$ promised to be the low-degree extension of a *Boolean* assignment $x \in \{0,1\}^k$, and our goal is to construct an $O(r^2)$-round HIP for verifying whether $x \in \mathcal{L}_{\mathsf{MOD3}}$. Also recall that we have fixed the parameters of the LDE code, including a field $\mathbb{F}$, a subset $H \subseteq \mathbb{F}$, and a dimension m such that $|H^m| = k$. However, for now we think of the sizes of these parameters as being $|H| = k^{1/r}$, $m = r$, and $|\mathbb{F}| = \mathsf{poly}(|H|, m)$, rather than $|H|$ being poly-logarithmic in k.[11]

For a given input polynomial $X : \mathbb{F}^m \to \mathbb{F}$ (of individual degree $|H| - 1$), we define a sequence of polynomials $V_0, \ldots, V_r$, where each $V_i : \mathbb{F}^i \to \mathbb{F}$ has individual degree $|H|-1$ (note that these polynomials have gradually increasing domains). The polynomial $V_r : \mathbb{F}^r \to \mathbb{F}$ is defined as $V_r \equiv X$. The polynomials $V_1, \ldots, V_{r-1}$ are each defined to be the (unique) individual degree $|H| - 1$ polynomial that satisfies the following recursive

[9] Loosely speaking, the uniformity condition requires that it be possible to compute *low degree extensions* of gate indicator functions that refer to gates of fan-in $t = n^{O(1/r)}$. That is, we view the formula as a depth r circuit consisting of gates of fan-in $t = n^{O(1/r)}$ (by grouping together every $\log(n)/r$ consecutive layers). For each of these r layers, and every type of fan-in t gate $g : \{0,1\}^t \to \{0,1\}$ that appears in that layer, we consider a gate indicator function I_g that given as input indices of $t+1$ wires, outputs 1 if the first wire is the result of an application of g to the other t wires. The [KR09] uniformity requirement is that it be possible to efficiently compute the *low degree extension* of I_g.

[10] Recall that the class NC^i consists of languages computable by polynomial-size $O\left((\log n)^i\right)$-depth circuits with fan-in 2. We emphasize that the [KR09] result gives *constant-round* protocols only for NC^1 circuits (that are sufficiently uniform), whereas the [GKR08] result gives protocol with a *poly-logarithmic* round complexity for all (logspace uniform) languages in $\mathsf{NC} = \cup_{k \in \mathbb{N}} \mathsf{NC}^k$. (Furthermore, the [GKR08] protocol for NC has poly-logarithmic communication complexity whereas the [KR09] protocol has $n^{1/O(1)}$ communication.)

[11] We remark that setting $|H| = k^{1/r}$ is actually problematic for us since it induces a dependence between the language $\mathsf{Enc\text{-}MOD3}$ and the desired round complexity r. Nevertheless, it does yield a weaker hierarchy theorem in which we use a different language for each value of r. At the end of Section 4.1.2.1 we discuss how we overcome this difficulty.

relation:

$$\forall i \in [r],\ \forall h \in H^{i-1}, \quad V_{i-1}(h) = \sum_{\alpha \in H} V_i(h, \alpha) \pmod 3, \tag{4.1}$$

where the arithmetic is over the integers (modulo 3). Indeed, $V_0 \in \mathbb{F}$ is defined as a single field element $V_0 = \sum_{\alpha \in H} V_1(\alpha) \pmod 3$. Note that we identify the integers $\{0, 1, 2\}$ with three distinct elements in $\mathbb{F}$. Indeed, each of the V_i polynomials takes values in the set $\{0, 1, 2\} \subseteq \mathbb{F}$ over the subcube H^i.

Taking the [GKR08, KR09] view, each polynomial $V_i : \mathbb{F}^i \to \mathbb{F}$ can be thought of as the low degree extension of the i^{th}-layer (counting from the output layer) in a depth r formula of fan-in $k^{1/r}$ for $\mathcal{L}_{\mathsf{MOD3}}$ such that each gate computes the sum modulo 3 of its $k^{1/r}$ children. In particular,

$$V_0 = \sum_{\alpha \in H} V_1(\alpha) = \cdots = \sum_{h \in H^i} V_i(h) = \cdots = \sum_{h \in H^r} V_r(h) = \mathsf{wt}(x) \pmod 3.$$

Our main step is an interactive protocol that reduces a claim about an (arbitrary) single point in the polynomial V_{i-1} to a claim about a single (random) point in V_i. By applying this interactive reduction r times, we can reduce the initial claim $V_0 = 0$ to a claim about a single point in V_r, which we can explicitly check (since we have oracle access to $V_r \equiv X$). Each interactive reduction will take $O(r)$ rounds so overall we get an HIP for $\mathcal{L}_{\mathsf{MOD3}}$ with $O(r^2)$ rounds.

Towards showing such an interactive reduction protocol, we would like to express Eq. (4.1), which is a modular equation over the integers, as a low degree relation over the field $\mathbb{F}$. Let $t \stackrel{\text{def}}{=} |H| = k^{1/r}$, and let $\xi_1, \ldots, \xi_t$ be the enumeration of all elements in H. Define the polynomial $\widetilde{\mathsf{MOD3}} : \mathbb{F}^t \to \mathbb{F}$ as the (unique) individual degree two polynomial such that for every $z \in \{0, 1, 2\}^t$, it holds that $\widetilde{\mathsf{MOD3}}(z) = \sum_{j \in [t]} z_i \pmod 3$, where the tilde in the notation is meant to remind us that $\widetilde{\mathsf{MOD3}}$ is not the modulo 3 summation function but rather its low degree extension over $\mathbb{F}$. Eq. (4.1) can now be re-stated as:

$$\forall i \in [r],\ \forall h \in H^{i-1}, \quad V_{i-1}(h) = \widetilde{\mathsf{MOD3}}\Big(V_i(h, \xi_1), \ldots, V_i(h, \xi_t)\Big) \tag{4.2}$$

(where we use the fact that the V_i polynomials take values in $\{0, 1, 2\}$ over H^i.)

Observe that Eq. (4.2) is a polynomial relation between V_{i-1} and V_i that holds for inputs in H^{i-1}. We would like to obtain a similar relation for general inputs (i.e., in $\mathbb{F}^{i-1}$). To do so, we observe that, for every $z \in \mathbb{F}^{i-1}$, we can express $V_{i-1}(z)$ as an $\mathbb{F}$-linear combination of the values $\{V_{i-1}(h)\}_{h \in H^{i-1}}$ (this follows directly from the fact that the low degree extension is a *linear* code). We denote the coefficients in this linear combination by $\{\beta_z(h)\}_{h \in H^{i-1}}$ (these coefficients arise from Lagrange interpolation, but we ignore the specifics for this overview). Combining this observation together with

Eq. (4.2) we obtain:

$$\forall i \in [r],\ \forall z \in \mathbb{F}^{i-1}, \quad V_{i-1}(z) = \sum_{h \in H^{i-1}} \beta_z(h) \cdot V_{i-1}(h)$$
$$= \sum_{h \in H^{i-1}} \beta_z(h) \cdot \widetilde{\mathsf{MOD3}}\Big(V_i(h, \xi_1), \ldots, V_i(h, \xi_t)\Big). \tag{4.3}$$

Using Eq. (4.3) we will describe an interactive reduction from a claim about V_{i-1} to a claim about V_i. Suppose that our interactive reduction starts with a claim that $V_{i-1}(z_{i-1}) = \nu_{i-1}$ for some $z_{i-1} \in \mathbb{F}^{i-1}$ and $\nu_{i-1} \in \mathbb{F}$. By Eq. (4.3) this translates into the claim:

$$\nu_{i-1} = \sum_{h \in H^{i-1}} \beta_{z_{i-1}}(h) \cdot \widetilde{\mathsf{MOD3}}\Big(V_i(h, \xi_1), \ldots, V_i(h, \xi_t)\Big). \tag{4.4}$$

We now observe that $Q_i(w) \stackrel{\text{def}}{=} \beta_{z_{i-1}}(w) \cdot \widetilde{\mathsf{MOD3}}\Big(V_i(w, \xi_1), \ldots, V_i(w, \xi_t)\Big)$ is a low degree polynomial over $\mathbb{F}$ (since $\beta_{z_{i-1}}$, $\widetilde{\mathsf{MOD3}}$, and V_i have low degree). Thus, the claim in Eq. (4.4) refers to the sum of a low degree polynomial over a subcube, which is precisely the problem that the sumcheck protocol solves.

It seems that we are done, except that a problem arises. In the sumcheck protocol the verifier is given oracle access to the polynomial whose sum over a subcube we wish to check. Although the polynomial Q_i on which we wish to run the sumcheck protocol is well-defined, our verifier does not have oracle access to it. Therefore it is not immediately clear how we can hope to run the sumcheck protocol with respect to Q_i.

We resolve this problem by noting that the sumcheck protocol can be used in an *input-oblivious* manner. In this variant, the verifier does not need to have oracle access to Q_i, but rather than accepting or rejecting, the verifier outputs a claim of the form $Q_i(w_{i-1}) = \gamma_{i-1}$, for some point $w_{i-1} \in \mathbb{F}^{i-1}$ and value $\gamma_{i-1} \in \mathbb{F}$. Completeness means that if the original claim is true (i.e., $\sum_{h \in H^{i-1}} Q_i(h) = \nu_{i-1}$), then the verifier always outputs (w_{i-1}, γ_{i-1}) such that $Q_i(w_{i-1}) = \gamma_{i-1}$, and soundness means that if the original claim is false (i.e., $\sum_{h \in H^{i-1}} Q_i(h) \neq \nu_{i-1}$), then for any cheating prover strategy, with high probability $Q_i(w_{i-1}) \neq \gamma_{i-1}$ (or the verifier rejects during the interaction). We stress that in this variant the verifier makes no queries to Q_i.[12] As for the number of rounds, recall that in the sumcheck protocol in each iteration one of the variables is "stripped" from the summation, which leads to a total of $i - 1 \leq r$ rounds.

Having run the input-oblivious variant of the sumcheck protocool, our verifier is now left with the claim $Q_i(w_{i-1}) = \gamma_{i-1}$. However, to obtain our interactive reduction, we still need to reduce the foregoing claim to a claim about a (single) point in the polynomial V_i. To do so, the first idea that comes to mind is to have the prover provide the values $\mu_j = V_i(w_{i-1}, \xi_j)$, for every $j \in [t]$. Given these values, the verifier can explicitly check

[12] To see that this variant is possible, observe that in the classical sumcheck protocol [LFKN92], the verifier only queries the polynomial at a single point and (at the end of the interaction) checks that it is equal to a particular value.

that indeed $\gamma_{i-1} = \beta_{z_{i-1}}(w_{i-1}) \cdot \widetilde{\mathsf{MOD3}}(\mu_1, \ldots, \mu_t)$.[13] If the prover indeed sent the correct values, then this last check assures us that indeed $Q_i(w_{i-1}) = \gamma_{i-1}$. However, since we cannot assume that the prover sent the correct values, we are left with t claim of the form $V_i(w_{i-1}, \xi_j) = \mu_j$, which the verifier needs to check.

Notice that we have actually reduced a single claim about V_{i-1} to t claims about V_i. This still falls short of our goal which was to reduce to only a *single* claim about V_i. (Indeed, we cannot afford to increase the number of claims by a t factor in each iteration, since this would yield a protocol with complexity $t^r = k$, which is trivial).

The final observation is that the points $\{(w_{i-1}, \alpha)\}_{\alpha \in H}$ lie on the (axis parallel) line $(w_{i-1}, *)$. Note that the restriction of a low degree polynomial to an axis parallel line is a low degree (univariate) polynomial. Thus, we will have the prover specify the entire polynomial $P_i : \mathbb{F} \to \mathbb{F}$ defined as $P_i(\alpha) = V_i(w_{i-1}, \alpha)$, for every $\alpha \in \mathbb{F}$. The verifier checks that $\gamma_{i-1} = \beta_{z_{i-1}}(w_{i-1}) \cdot \widetilde{\mathsf{MOD3}}\,(P_i(\xi_1), \ldots, P_i(\xi_t))$. The point is that now if the prover supplies an incorrect values for some $P_i(\alpha)$ (i.e., $P_i(\alpha) \neq V_i(w_{i-1}, \alpha)$), since both P_i and $V_i(w_{i-1}, *)$ are low degree polynomials, for most $\rho \in \mathbb{F}$ it holds that $P_i(\rho) \neq V_i(w, \rho)$. Thus, the verifier chooses at random $\rho_i \in \mathbb{F}$ and sets the claim for the next iteration to be $V_i(z_i) = \nu_i$, where $z_i = (w_{i-1}, \rho_i)$ and $\nu_i = P_i(\rho_i)$.[14]

To summarize, our HIP for $\mathcal{L}_{\mathsf{MOD3}}$ works in r phases. In the i^{th} phase we reduce a claim of the form $V_{i-1}(z_{i-1}) = \nu_{i-1}$, for some point $z_{i-1} \in \mathbb{F}^{i-1}$ and value $\nu_{i-1} \in \mathbb{F}$, into a claim $V_i(z_i) = \nu_i$, for $z_i \in \mathbb{F}^i$ and $\nu_i \in \mathbb{F}$ (which are generated during the interactive reduction). In particular, the first iteration begins with the claim $V_0 = 0$ (i.e., z_0 is the empty string and $\nu_0 = 0$), which corresponds to the claim that $x \in \mathcal{L}_{\mathsf{MOD3}}$ (i.e., $\mathsf{wt}(x) = 0 \pmod 3$). Thus, the i^{th} phase in our HIP begins with the claim $V_{i-1}(z_{i-1}) = \nu_{i-1}$. In the i^{th} phase, first the prover and verifier engage in the sumcheck protocol that arises from Eq. (4.4). This yields the claim $Q_i(w_{i-1}) = \gamma_{i-1}$, for a point $w_{i-1} \in \mathbb{F}^{i-1}$ and value $\gamma_{i-1} \in \mathbb{F}$ (generated by the sumcheck protocol). Since the verifier has no access to Q_i, it asks the prover to send the polynomial $P_i : \mathbb{F} \to \mathbb{F}$ defined as $P_i(\alpha) = V_i(w_{i-1}, \alpha)$. The verifier checks that the values of this polynomial are consistent with the claim $Q_i(w_{i-1}) = \gamma_{i-1}$, and then selects a random point $\rho_i \in \mathbb{F}$. The claim for the following phase is that $V_i(z_i) = \nu_i$, where $z_i = (w_{i-1}, \rho_i)$ and $\nu_i = P_i(\rho_i)$. After r such phases we are left with the claim $V_r(z_r) = \nu_r$, for $z_r \in \mathbb{F}^r$ and $\nu_r \in \mathbb{F}$, which the verifier can explicitly check (since it has oracle access to $V_r \equiv X$).

The total number of rounds per interactive reduction is $O(r)$, and the communication complexity is roughly $\mathsf{poly}(t, r) = \mathsf{poly}(r, k^{1/r})$. Since we invoke r such reductions, overall we obtain an HIP for $\mathcal{L}_{\mathsf{MOD3}}$ with round complexity $O(r^2)$ and communication complexity $\mathsf{poly}(r, k^{1/r})$.

[13] Note that both $\beta_{z_{i-1}}$ and $\widetilde{\mathsf{MOD3}}$ are *explicit* functions that the verifier can compute. Moreover they can even be computed *efficiently* using standard techniques, see the technical sections for details.

[14] We remark that this final step is actually very reminiscent of an individual round of the sumcheck protocol.

Obtaining an HIP over a Small Field. The approach outlined above yields an r^2-round HIP for $\mathcal{L}_{\mathsf{MOD3}}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$, in which the field size $|\mathbb{F}|$ is quite large (i.e., $|\mathbb{F}| \geq k^{1/r}$) and in particular depends on the value of r. Unfortunately, when we transform this HIP into an IPP for the language Enc-MOD3, the dependence of the field size on r in the HIP introduces a dependence of the language $\text{Enc-MOD3} \stackrel{\text{def}}{=} \{C(x) \ : \ x \in \{0,1\}^k \text{ with } \mathsf{wt}(x) = 0 \pmod 3\}$ on r. This dependence results in a weaker hierarchy theorem, in which we use a different language for each value of r . Our goal however is to obtain a *single* language, for which we can show an r-round IPP for every value of r (with a corresponding lower bound, which will be discussed in Section 4.1.2.2).

To this end we show a general reduction that transforms any HIP over a large field $\mathbb{F}$ into an HIP over a much smaller field $\mathbb{F}'$, as long as $\mathbb{F}$ is an extension field of $\mathbb{F}'$. We do so by showing that any $\mathbb{F}$-linear claim regarding the input (e.g., a claim about a single point in the $\mathsf{LDE}_{\mathbb{F},H,m}$ encoding) can be broken down (coordinate-wise) into d claims that are $\mathbb{F}'$-linear, where $d = \log(|\mathbb{F}|/|\mathbb{F}'|)$ is the degree of the field extension (i.e., $(\mathbb{F}')^d$ is isomorphic to $\mathbb{F}$). We can then easily verify each one of these $\mathbb{F}'$-linear claims using the sumcheck protocol over the smaller field $\mathbb{F}'$. We remark that the ability to switch fields when using (holographic) interactive proofs seems like a useful tool, and we believe that it will be useful in other contexts as well.

Checking Booleanity. In the above analysis we assumed for simplicity that the input $x = f|_{H^m}$ is Boolean valued. In order to actually check this, we follow an idea of Kalai and Raz [KR08] (which was used in the context of constructing interactive PCPs). We observe that the polynomial $f : \mathbb{F}^m \to \mathbb{F}$ is Boolean valued in a subcube H^m if and only if the (slightly higher degree) polynomial $g : \mathbb{F}^m \to \mathbb{F}$, defined as $g(z) = f(z) \cdot (1 - f(z))$ is identically 0 in H^m. The latter problem (of checking whether a polynomial vanishes on a particular subcube) can be solved via a relatively simple reduction to the sumcheck protocol, that has been used in the construction of PCPs.[15] We note that we crucially use fact that the reduction from f to g is local (i.e., the value of g at a point depends on the value of f at $O(1)$ points), and therefore can be used in our setting.

4.1.2.2 Lower Bound

We need to show a lower bound on the complexity of r-round IPPs for our language $\text{Enc-MOD3} = \{C(x) \ : \ x \in \{0,1\}^k \text{ with } \mathsf{wt}(x) = 0 \pmod 3\}$, where $C : \mathbb{F}^k \to \mathbb{F}^n$ is the low degree extension code. Our lower bound will strongly use the fact that any $\mathbb{F}$-linear code (and in particular the low degree extension code that we use), for a field $\mathbb{F}$ of characteristic 2, is also a $\mathsf{GF}(2)$-linear code.

Our lower bound relies on a connection between IPPs and low-depth circuits, which

[15] In a nutshell, to check whether $g|_{H^m} \equiv 0$ we consider the restriction of g to the domain H^m and take the low degree extension $\hat{g}$ of that partial function. We observe that g is identically 0 in H^m if and only if $\hat{g}$ is identically 0 in $\mathbb{F}^m$. Thus, it suffices to check whether for a random point $z \in \mathbb{F}^m$, which the verifier chooses, it holds that $\hat{g}(z) = 0$. The linearity of the LDE code now means that this check can be solved by invoking the sumcheck protocol. See Section 4.3.4 for details.

was discovered by Rothblum, Vadhan and Wigderson [RVW13]. Following their approach, in Section 4.4.3 we show that to prove an IPP lower bound for Enc-MOD3, it suffices to construct two distributions D_0 and D_1 over n-bit strings such that:

1. D_0 is distributed over the support of Enc-MOD3 (with high probability);
2. D_1 is far from Enc-MOD3 (with high probability); and
3. Every sufficiently small DNF formula cannot distinguish between inputs from D_0 and D_1 (with more than, say, 0.1 advantage).

The two distributions that we consider are D_0 and D_1 such that D_b is uniform over the set $\{C(x) \ : \ x \in \{0,1\}^k \text{ and } \mathsf{wt}(x) = b \pmod 3\}$. Note that D_0 is the uniform distribution over Enc-MOD3, and so satisfies requirement (1), whereas the fact that D_1 satisfies requirement (2) follows from the distance of the code C. To show that the third requirement holds, consider a DNF ϕ that distinguishes between D_0 and D_1. We show that the size of ϕ must be large. Consider the distributions D'_0 and D'_1 over k-bit strings defined as

$$D'_b = \{x \in \{0,1\}^k \ : \ \mathsf{wt}(x) = b \pmod 3\}.$$

We can easily construct from ϕ a circuit ϕ' that distinguishes between D'_0 and D'_1: the circuit ϕ' first computes the encoding $C(x)$ of its input $x \in \{0,1\}^k$, and then applies the DNF ϕ to the result. Using the fact that C is linear over GF(2), it follows that ϕ' is a DNF of parities (i.e., a depth-3 formula with an OR gate at the top layer, AND gates at the middle layer, and XOR gates at the bottom layer). Now, we can apply the Razborov-Smolensky [RS96] lower bound, which shows that any small $\mathsf{AC}_0[2]$ circuit (i.e., circuits of constant-depth circuits with AND, OR, and PARITY gates of unbounded fan-in), and in particular a DNF of parities, cannot even approximate the summation modulo 3 function (i.e., distinguish between D'_0 and D'_1).

4.1.3 Holographic Interactive Proofs

The proof of our hierarchy theorem utilizes a special type of interactive proofs, which we call *holographic interactive proofs*. A holographic interactive proof (HIP) is an interactive proof in which, instead of getting its input x explicitly, the verifier is given *oracle* access to $C(x)$, an error-corrected encoding of the input x, for a bounded number of queries. Hence, HIPs may be thought of as interactive proofs for promise problems of the form $(\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ with $\Pi_{\mathsf{YES}} = \{C(x) : x \in \mathcal{L}\}$ and $\Pi_{\mathsf{NO}} = \{C(x) : x \notin \mathcal{L}\}$.

The notion of HIP was used, either implicitly or explicitly as a technical tool that underlies many probabilistic proof systems (e.g., [LFKN92, BFL91, BFLS91, KR08, GKR08, KRR13, RVW13, GR13b, KRR14, RRR16, GG16a]).[16] These works demonstrate that, by using the redundant encoding of the input, we can often achieve sublinear verification time. (As a matter of fact, in most of these works, it suffices for the verifier to read just a *single* point in the encoding.) We remark that throughout this chapter (as well as in

[16] The first explicit use is in [BFLS91].

most previous works[17]), the specific code that is used is the *low-degree extension code* (LDE).

Some of the techniques that were outlined in Section 4.1.2.1, can be viewed as generic transformations on HIPs (with respect to the LDE code), and we present them as such in the technical parts of this chapter. These techniques include the ability to switch fields, or check Booleanity, and the connection to IPPs. We wish to highlight the conceptual importance of HIPs, and advocate a continued systematic study of these proof systems.

We also remark that HIPs with respect to the LDE code are closely related to interactive proofs in the algebrization framework [AW09]. In both models the verifier is given oracle access to a low degree polynomial and may interact with the prover to decide on some property of the "message" or "oracle" encoded within the polynomial. See Section 4.5 for further discussion of this connection.

4.1.4 Related Works

In this section, we discuss several lines of works that are related to our work.

Interactive Proofs of Proximity. The notion of interactive proofs of proximity (IPP) was first considered by Ergün, Kumar and Rubinfeld [EKR04]. Its study was re-initiated by Rothblum, Vadhan and Wigderson [RVW13], who showed that every language computable by a low-depth circuit has an IPP with a sublinear time verifier. IPPs were further studied by [GGR15, GG16a] who showed more efficient IPPs for certain restricted complexity classes. Other works have focusing on variants such as non-interactive (MA) proofs of proximity [GR13b, FGL14, GGK15] and interactive *arguments* of proximity [KR15]. Proofs of proximity have also found applications to property testing and related models [GR13a, GR14, FLV15].

Hierarchy Theorems for Standard Interactive Proofs. Aiello, Goldwasser and Håstad [AGH90] showed a round hierarchy theorem in a relativized world (i.e., with respect to an oracle). However, the later results of [LFKN92, Sha92], which are based on non-relativizing techniques, demonstrate that relativization is not an actual barrier, especially in the context of interactive proofs.[18] We note that although they are technically quite different, both our lower bound and the lower bound of [AGH90] are based on circuit lower bounds for low depth circuits.

Goldreich, Vadhan and Wigderson [GVW02] showed a *conditional* round hierarchy result for standard interactive proofs, based on the assumption that co-SAT does not

[17] A notable exception is the work of Meir [Mei13], which is based on general tensor codes. We remark that using Meir's techniques it may be possible to extend our results to other tensor codes. We leave exploring this possibility to future work.

[18] Indeed, Fortnow and Sipser [FS88] show that the proof of IP = PSPACE cannot be relativized (in fact, IP does not even contain coNP relative to a random oracle [CCG+94]). In fact, the algebrization framework of Aaronson and Wigderson [AW09] was proposed precisely to address this issue. Connections between our results and algebrization are further discussed in Section 4.5.

have a 1-round AM proof-system with complexity $2^{o(n)}$.[19] We emphasize that the result of [GVW02] is based on an unproven and arguably strong (yet believable) assumption, whereas our result is unconditional.

We also note that for *computationally sound* proofs, also known as arguments, under reasonable cryptographic assumptions there are extremely efficient 2-round protocols [Kil92] and even 1-round protocols [KRR14]. In particular, these results show that the power of arguments does not scale with additional rounds (since a fixed constant number of rounds suffice). A similar statement holds for arguments of proximity that are the computationally sound variant of IPPs (see [RVW13, KR15]).

Interactive PCPs. Holographic interactive proofs (HIPs) are closely related to the notion of *interactive* PCPs, introduced by Kalai and Raz [KR08]. Roughly speaking, interactive-PCPs are encodings of NP-witnesses that, like PCPs can be verified using few queries, but here the verification procedure may use interaction with an unbounded (and untrusted) prover. Thus, using our terminology, an interactive PCP can be thought of as an HIP for checking the NP witness relation.

Arthur-Merlin Query Complexity. Every IPP for a language $\mathcal{L}$ can be viewed as a protocol, for a promise problem related to $\mathcal{L}$, in the Arthur Merlin query complexity model, previously studied by Raz *et al.* [RTVV98]. This model, similarly to IPPs, considers a sub-linear time verifier, that is given oracle access to an input and may interact with an (untrusted) prover. Indeed, one may view IPPs as Arthur Merlin query complexity protocols which focus on promise problems in which the goal of the verifier is to distinguish between inputs having a certain property from those that are *far* from having the property.

Thus, our main result directly yields a round hierarchy theorem (for a promise problem) in the *Arthur-Merlin Query Complexity* model and a sub-exponential separation between the complexity of constant-round vs. general (i.e., unbounded round) Arthur-Merlin Query Complexity protocols.

Interactive Proofs in Other Models. Interactive proof systems were studied also in the communication complexity setting (e.g., [BFS86, Kla11, She12, GPW15b, GPW15a]). Here Alice and Bob may interact with an untrusted Merlin, who sees both of their inputs. We remark that showing any non-trivial explicit lower bound in the AM variant of this model, much less a hierarchy of separations, is a notorious open problem.

A recent line of works has studied interactive proofs in the data streaming model (e.g., [CCM09, CMT12, CMT13, GR15b, CCGT14, Tha16, DTV15]). Most relevant is a result of Chakrabarti *et al.* [CCM+15], who show a hierarchy theorem for the first four levels in the model of *online interactive proofs* (with exponential separations between these four levels).

[19] Related assumptions have recently been studied also by Carmosino *et al.* [CGI+16] and Williams [Wil16].

Universal Locally Verifiable Codes. In a recent work, Goldreich and Gur [GG16b] introduced the notion of *universal locally verifiable codes* (universal-LVC), which is closely related to holographic interactive proofs. A universal-LVC $C : \{0,1\}^k \to \{0,1\}^n$ for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ is a code such that for every $i \in [M]$, membership in the subcode $\{C(x) : f_i(x) = 1\}$ can be verified locally given an explicit access to a short (sublinear length) proof; put differently, for every $i \in [M]$ there exists a 1-message IPP for the property $\{C(x) : f_i(x) = 1\}$, with sublinear communication and query complexity.

Organization

In Section 7.2 we define IPPs and introduce some notations and definitions that we use throughout this chapter. In Section 4.3 we define holographic interactive proofs (HIPs) and prove some general results on them. In Section 4.4, using some of the results of Section 4.3, we prove the hierarchy theorem. Lastly, in Section 4.5 we discuss the implications to classical complexity theory.

Some of the discussion and proofs are deferred to the appendix. In Section 4.6.1.2 we discuss an alternative language for the round hierarchy theorem and our choice of basing our protocol on [GKR08] rather than, say a recent protocol of Reingold *et al.* [RRR16]. Sections 4.6.2 to 4.6.4 contain some standard proofs that are included for completeness.

4.2 Preliminaries

We begin with some standard notations:

- We denote the relative distance, over alphabet Σ, between two strings $x \in \Sigma^n$ and $y \in \Sigma^n$ by $\Delta_{\mathsf{REL}}(()\, x, y) \stackrel{\text{def}}{=} \frac{|\{x_i \neq y_i \,:\, i\in[n]\}|}{n}$. If $\Delta_{\mathsf{REL}}(()\, x, y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we denote the relative distance of x from a non-empty set $S \subseteq \Sigma^n$ by $\Delta_{\mathsf{REL}}(()\, x, S) \stackrel{\text{def}}{=} \min_{y\in S} \Delta_{\mathsf{REL}}(()\, x, y)$. If $\Delta_{\mathsf{REL}}(()\, x, S) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S.

- We denote the projection of $x \in \Sigma^n$ to a subset of coordinates $I \subseteq [n]$ by $x|_I$ and, for $i \in [n]$, write $x_i = x|_{\{I\}}$ to denote the projection to a singleton.

An additional notation that we will use is that if $S = (S_k)_{k\in\mathbb{N}}$ and $T = (T_k)_{k\in\mathbb{N}}$ are ensembles of sets, we denote by $S \subseteq T$ the fact that $S_k \subseteq T_k$ for every $k \in \mathbb{N}$.

Integrality. Throughout this chapter, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the closest integer.

4.2.1 Interactive Proofs of Proximity

A language is an ensemble $\mathcal{L} = (\mathcal{L}_n)_{n\in\mathbb{N}}$, where $\mathcal{L}_n \subseteq (\Sigma_n)^n$ for every $n \in \mathbb{N}$ and where $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$ is the alphabet.

Definition 4.1 (Interactive Proofs of Proximity (IPP))**.** *Let $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$ be an alphabet ensemble. An r-round* interactive proof of proximity*, with respect to proximity parameter $\varepsilon > 0$, (in short, ε-IPP) for the language $\mathcal{L}$ is an interactive protocol between a prover $\mathcal{P}$, which gets* free *access to ε and to an input $x \in \Sigma^n$, and a verifier $\mathcal{V}$, which gets free access only to ε and n, as well as* oracle *access to x. At the end of the protocol, the following conditions are satisfied:*

- **Completeness***: If $x \in \mathcal{L}$, then, when $\mathcal{V}$ interacts with $\mathcal{P}$, with probability $2/3$ it accepts.*
- **Soundness***: If x is ε-far from $\mathcal{L}$, then for every prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, with probability $2/3$ it rejects.*

If the completeness condition in Definition 7.1 holds with probability 1, then we say that the IPP has perfect completeness. A public-coin IPP is an IPP in which every message from the verifier to the prover consists only of fresh random coin tosses.

An IPP is said to have query complexity $q : \mathbb{N} \times [0,1] \to \mathbb{N}$ if for every $n \in \mathbb{N}$, $\varepsilon > 0$, $x \in \{0,1\}^n$, and any prover strategy $\mathcal{P}^*$, the verifier makes at most $q(n,\varepsilon)$ queries to x when interacting with $\mathcal{P}^*$. The IPP is said to have communication complexity $c : \mathbb{N} \times [0,1] \to \mathbb{N}$ if for every $n \in \mathbb{N}$, $\varepsilon > 0$, and $x \in \mathcal{L}_n$ the communication between $\mathcal{V}$ and $\mathcal{P}$ consists of at most $c(n,\varepsilon)$ bits.

4.2.2 Constructible Error Correcting Codes and Finite Fields

An error correcting code over an alphabet Σ is an injective function $C : \Sigma^k \to \Sigma^n$. The code C is said to have relative distance δ if for any $x \neq x' \in \Sigma^k$ it holds that $\Delta_{\mathsf{REL}}(() x, x') \geq \delta$.

Throughout this chapter we deal with (uniform) polynomial-time algorithms, and so we will need (families of) codes that are efficiently computable. Formally, for a parameter $n = n(k) \geq 1$ that is called the blocklength, and ensemble of alphabets $\Sigma = (\Sigma_k)_{k\in\mathbb{N}}$, we define a constructible error correcting code over Σ as an ensemble $C = \left(C_k : \Sigma_k^k \to \Sigma_k^n\right)_{k\in\mathbb{N}}$ of error correcting codes, such that the function $f(x) = C_{|x|}(x)$ is computable by a polynomial-time Turing machine (in particular this implies that $n = \mathsf{poly}(k, \log(\Sigma))$). An ensemble of error correcting codes $C = (C_k)_{k\in\mathbb{N}}$ is said to have relative distance δ if for all sufficiently large k, each code C_k in the ensemble has relative distance δ.

Throughout this chapter, we mostly consider codes defined over finite fields (i.e., the alphabets Σ_k are all finite fields). Such codes are called linear if they are linear functions over the field.

4.2.2.1 Finite Fields and Polynomials

Many of our algorithms and interactive proofs deal with finite fields. We consider ensembles of finite fields $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$, where $|\mathbb{F}_k|$ and say that such ensembles are $\mathsf{constructible}$ if the field operations can be done in $\mathsf{poly}\log(|\mathbb{F}_k|)$ time. Namely, there exist a Turing machine that given as input k and an appropriate number of elements in $\mathbb{F}_k$ (represented as strings of length $O(\log(|\mathbb{F}_k|))$ bits) can compute the field operations (i.e., addition, subtraction, multiplication, inversion, and sampling random elements) in $\mathsf{polylog}(|\mathbb{F}_k|)$ time.

The following fact shows that there exist constructible finite fields of characteristic 2.

Fact 4.2. *For every time-constructible function $f = f(k) \geq 1$, there exists a* constructible *field ensemble $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ such that $|\mathbb{F}| = O(f)$ and $\mathbb{F}_k$ has characteristic 2 (i.e., is an extension field of $\mathsf{GF}(2)$) for every $k \in \mathbb{N}$.*

For details see [Gol08, Appendix G.3] and references therein. We will also use the well-known Schwartz-Zippel Lemma.

Lemma 4.3 (Schwartz-Zippel Lemma)**.** *Let $P : \mathbb{F}^m \to \mathbb{F}$ be a non-zero polynomial of total degree d over the field $\mathbb{F}$. Then,*

$$\Pr_{x\in_R\mathbb{F}^m}[P(x) = 0] \leq \frac{d}{|\mathbb{F}|}.$$

4.2.3 Low-Degree Extension

Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ be an ensemble of fields, and let $H = (H_k)_{k\in\mathbb{N}} \subseteq \mathbb{F}$ (the notation $H \subseteq \mathbb{F}$ means that $H_k \subseteq \mathbb{F}_k$, for every $k \in \mathbb{N}$). Let $m = m(k) \geq 1$ be a parameter, which we often call the dimension.

A basic fact is that for every function $f : H^m \to \mathbb{F}$ there exists a *unique* function $\tilde{f} : \mathbb{F}^m \to \mathbb{F}$ such that $\tilde{f}$ is a polynomial with individual degree $|H| - 1$ that agrees with f on H^m. Moreover, there exists an individual degree $|H| - 1$ polynomial $\beta : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ such that for every function $f : H^m \to \mathbb{F}$ it holds that

$$\tilde{f}(z) = \sum_{x\in H^m} \beta(x, z) \cdot f(x).$$

The function $\tilde{f}$ is called the low degree extension of $\mathbb{F}$ (with respect to the field $\mathbb{F}$, subset H and dimension m).

The following two propositions show that the low degree extension encoding can be computed efficiently.

Proposition 4.4. *Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ be a constructible field ensemble, let $H = (H_k)_{k\in\mathbb{N}} \subseteq \mathbb{F}$ be an ensemble of subsets and let $m = m(k)$ be the dimension.*

There exists a Turing machine that on input k runs in time $\mathsf{poly}(|H|, m, \log|\mathbb{F}|)$ and space $O(\log(|\mathbb{F}|) + \log(m))$, and outputs the polynomial $\beta : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ defined above, represented as an arithmetic circuit over $\mathbb{F}$.

Moreover, the arithmetic circuit β can be evaluated in time $\mathsf{poly}(|H|, m, \log(|\mathbb{F}|))$ *and space* $O(\log(|\mathbb{F}|) + \log(m))$*. Namely, there exists a Turing machine with the above time and space bounds that given an input pair* $(x, z) \in \mathbb{F}^m \times \mathbb{F}^m$ *outputs* $\beta(x, z)$*.*

See, e.g., [Rot09, Proposition 3.2.1] for a proof of Proposition 4.4.

Proposition 4.5. *Let* $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ *be a constructible field ensemble, let* $H = (H_k)_{k\in\mathbb{N}} \subseteq \mathbb{F}$ *be an ensemble of subsets and let* $m = m(k)$ *be the dimension.*

Let $\phi : H^m \to \mathbb{F}$ *and suppose that* ϕ *can be evaluated by a Turing Machine in time* t *and space* s*. Then, there exists a Turing machine that, given as an input a point* $z \in \mathbb{F}^m$*, runs in time* $|H|^m \cdot (\mathsf{poly}(|H|, m, \log(|\mathbb{F}|)) + O(t))$ *and space* $O(m \cdot \log(|H|) + s + \log(|\mathbb{F}|))$ *and outputs the value* $\hat{\phi}(z)$ *where* $\hat{\phi}$ *is the unique low degree extension of* ϕ *(with respect to* $H, \mathbb{F}, m$*).*

Proof. The Turing machine computes

$$\hat{\phi}(z) = \sum_{x\in H^m} \beta(x, z) \cdot \phi(x)$$

by generating and evaluating β as in Proposition 4.4. □

4.2.3.1 Low Degree Extension as an Error-Correcting Code

The low degree extension can also be viewed as an error-correcting code in the following way. Suppose that H and m are such that $|H|^m = k$. Then, we can associate a string $x \in \mathbb{F}^k$ with a function $x : H^m \to \mathbb{F}$ by identifying H^m with $[k]$ in some canonical way.

We define the low degree extension of a string x as $\mathsf{LDE}_{\mathbb{F},H,m}(x) = \tilde{x}$. That is, the function $\mathsf{LDE}_{\mathbb{F},H,m}$ is given as input the string $x \in \mathbb{F}^k$, views it as a function $x : H^m \to \mathbb{F}$ and outputs its low degree extension $\tilde{x}$. By Proposition 4.5 the code $\mathsf{LDE}_{\mathbb{F},H,m}$ is constructible, and by the Schwartz-Zippel Lemma (Lemma 4.3), the code $\mathsf{LDE}_{\mathbb{F},H,m}$ has relative distance $1 - \frac{m\cdot|H|}{|\mathbb{F}|}$.

4.3 Holographic Interactive Proofs

In this section we define *holographic interactive proofs* and show several transformations and generic results (which will be used in Section 4.4 for the proof of the hierarchy theorem). In Section 4.3.1 we give a formal definition and some basic facts. Having read Section 4.3.1, the reader may freely skip the rest of Section 4.3 and proceed directly to Section 4.4, which is the main technical section, and return to read the results of Sections 4.3.2 to 4.3.4 when they are used in Section 4.4.

Sections 4.3.2 to 4.3.4 focus on HIPs with respect to the low degree extension encoding. In Section 4.3.2 we show that such HIPs imply interactive proofs of *proximity* (for a related language). In Section 4.3.3 we show that one can switch the field under which the HIPs input is encoded (at a moderate cost) to any other field *that shares the same characteristic*. Finally, in Section 4.3.4 we show that HIPs can efficiently verify that the

input (which can presumably be an arbitrary vector over the field) is actually Boolean valued (i.e., in $\{0,1\}^k$).

4.3.1 Definition and Basic Facts

A *holographic interactive proof* is similar to a standard interactive proof, except that rather than getting the input explicitly, the verifier gets oracle access to an encoding of the input (via an error correcting code). Using this redundant representation, we could potentially hope to have protocols in which the verifier runs in *sublinear* time and, in particular, does not even read its entire input. This hope is indeed materialized in several protocols from the literature (e.g., [LFKN92, GKR08, RRR16]).

As a matter of fact, it turns out that for some codes (specifically the low degree extension), reading just a single point p from the encoded input suffices for the verifier.[20] Thus, we restrict our attention to such protocols. Furthermore, in order to facilitate composition, rather than having the verifier actually read the (encoded) input at the point p, the verifier outputs a claim about the point (i.e., it outputs p together with a symbol that it would have expected to see, had it actually queried the (encoded) input at p).

Formally, holographic interactive proofs are parametrized by a (constructible) error correcting code C, under which the input is encoded, and are defined as follows.

Definition 4.6 (Holographic Interactive Proofs (HIP))**.** *Let $\Sigma = (\Sigma_k)_{k\in\mathbb{N}}$ and $\Lambda = (\Lambda_k)_{k\in\mathbb{N}}$ be alphabet ensembles such that $\Lambda \subseteq \Sigma$. Let $\mathcal{L} \subseteq \Lambda$, and let $C : \Sigma^k \to \Sigma^n$ be a constructible error correcting code.*

An r-round public-coin holographic interactive proof (HIP) *for the language $\mathcal{L}$, with respect to the code C, is an interactive protocol between a prover $\mathcal{P}$, which gets as input $x \in \Sigma^k$, and a verifier $\mathcal{V}$, which gets as input only k. At the end of the protocol either the verifier rejects or it outputs a coordinate $i \in [n]$ and a symbol $\sigma \in \Sigma$ such that:*

- **Completeness***: If $x \in \mathcal{L}$, then, when $\mathcal{V}$ interacts with $\mathcal{P}$, with probability 1 it outputs (i,σ) such that $C(x)|_i = \sigma$.*
- **Soundness***: If $x \notin \mathcal{L}$, then for every prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, with probability $1-\varepsilon$ either $\mathcal{V}$ rejects or it outputs (i,σ) such that $C(x)|_i \neq \sigma$, where $\varepsilon = \varepsilon(k) \in [0,1]$ is called the* soundness error*.*

In this chapter, all the holographic proofs that we consider are with respect to the low degree extension code (using a variety of different parameters), which was defined in Section 4.2.3 above.

[20] For the low degree extension this can be shown to hold generically. The high level idea is to consider a low degree curve passing through all the points that the verifier wishes to read. The prover specifies the values for all the points on the curve and the verifier checks the provided answer on a random point on the curve. Soundness follows from the fact that composing a low-degree curve with a low-degree polynomial results in a low degree univariate polynomial. See, e.g., [KR08, Section 6] for details.

Remark 4.7 (Different Alphabets for the Language and the Code)**.** *Typically, when using* HIP*s the alphabet Λ over which the language is defined will be the same as the alphabet Σ over which the code is defined. Still, in some cases it will be convenient for us to present* HIP*s that only work for particular sub-alphabets of the code (e.g., when the input is binary but the code is more naturally defined over some large alphabet) and so we give this more flexible definition.*

Our definition of HIPs tries to capture many of the known interactive proof-systems in the literature, while being flexible and easy to compose. Indeed, the fact that HIPs can be transformed into standard interactive proofs which is immediate, is captured by the following proposition.

Proposition 4.8. *Let $\Sigma = (\Sigma_k)_{k\in\mathbb{N}}$ and $\Lambda = (\Lambda_k)_{k\in\mathbb{N}}$ be alphabets such that $\Lambda \subseteq \Sigma$. Let $\mathcal{L}$ be a language over the alphabet Λ and let $C : \Sigma^k \to \Sigma^n$ be a constructible error correcting code.*

Any HIP *for $\mathcal{L}$ can be converted into a standard interactive proof with only a $\mathsf{poly}(n)$ additive overhead to the verifier's running time (and all other parameters remain unchanged). Moreover, the precise overhead is equal to the time that it takes to compute the i^{th} character of $C(x)$, given $x \in \Lambda^k$ and the index $i \in [n]$.*

Proof. The prover and verifier run the HIP. If the HIP verifier rejects, then we immediately reject. Otherwise, the HIP verifier outputs a pair $(i, \sigma) \in [n] \times \Sigma$ with the associated claim $C(x)|_i = \sigma$. We can now check this claim directly by computing $C(x)|_i$ and comparing with σ. □

4.3.1.1 The Sumcheck Protocol (as an HIP)

We will make extensive use of the classical sumcheck protocol of Lund *et al.* [LFKN92]. Recall that the sumcheck protocol is an interactive proof for verifying that the sum, over a subcube, of a low degree polynomial is zero. Our protocol differs slightly from the "textbook" sumcheck protocol in two ways:

1. The verifier does not actually read any points from the input polynomial. Rather, at the end of the protocol it outputs a claim about a single point of the polynomial (i.e., the protocol is an HIP).

2. Following other works in the literature, our protocol allows a trade-off between the number of rounds and the communication complexity (rather than having the number of rounds correspond exactly to the dimension of the polynomial).

Lemma 4.9 (Sumcheck as an HIP)**.** *Let $\mathbb{F}$ be a constructible field ensemble and let $H \subseteq \mathbb{F}$ be an ensemble of subsets of $\mathbb{F}$. Let $m = m(k)$ be an ensemble of integers such that $m = \log_{|H|}(k)$.*

Let $\mathcal{L} = \cup_{k\in\mathbb{N}}\mathcal{L}_k$, where $\mathcal{L}_k = \{x \in \mathbb{F}^k : \sum_{i\in[k]} x_i = 0\}$ and where the summation is over the field $\mathbb{F}$. Then, for every $r \in [m]$, there exists an r-round (public-coin) HIP

for $\mathcal{L}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$, with soundness error $\frac{m\cdot|H|}{|\mathbb{F}|}$ and communication complexity $|H|^{\lceil m/r\rceil}\cdot r\cdot\log|\mathbb{F}|$. The verifier runs in time $|H|^{\lceil m/r\rceil}\cdot r\cdot\mathsf{polylog}(|\mathbb{F}|)$ and the prover runs in time $\mathsf{poly}(|\mathbb{F}|^m, r)$.

The proof of Lemma 4.9, which is standard, is included for completeness in Section 4.6.3.

4.3.2 From HIP to IPPs

Proposition 4.8 above, shows that an HIP can be easily transformed into a standard interactive proof. We now show that HIPs, with respect to the low degree extension encoding, can be easily transformed into highly efficient (and in particular sublinear) *interactive proof of proximity* (IPP) for a related language. More specifically, we transform an HIP for the language $\mathcal{L}$ with respect to the $\mathsf{LDE}_{\mathbb{F},H,m}$ code, into an IPP for the language $\mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L}) \stackrel{\text{def}}{=} \{\mathsf{LDE}_{\mathbb{F},H,m}(x) : x \in \mathcal{L}\}$.[21]

Lemma 4.10. *Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ be an ensemble of finite fields, let $H = (H_k)_{k\in\mathbb{N}}$ be an ensemble of subsets (i.e. $H \subseteq \mathbb{F}$) and let $m = m(k)$ be such that $|H|^m = k$.*

Suppose that the language $\mathcal{L}$ has an r-round HIP, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$, with communication complexity c. Then, the language $\mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L})$ has an r-round ε-IPP with query complexity $O(|H|\cdot m\cdot 1/\varepsilon)$ and communication complexity c.

The key observations that we use to prove Lemma 4.10 are that (1) the IPP verifier can first check that its input is close to a low degree polynomial using low degree test. If the test passes, then, using the self-correctability of polynomials, the IPP verifier can emulate access to the encoded input of the HIP. Given these two observations the proof of Lemma 4.10 is standard and so we defer it to Section 4.6.2.

4.3.3 Field Switching

In this subsection we show that HIPs can evaluate points in a LDE over an *extension* field of the base field under which the input is actually encoded. This fact is used in the proof Lemma 4.15 and allows us to first construct an HIP over a large field, and later convert it into an HIP over the smaller field.

The key observation for our field switching, is that verifying a linear claim involving the LDE over an extension field $\mathbb{K}/\mathbb{F}$ can be reduced to verifying several linear claims over the base field $\mathbb{F}$. Each of these linear claims can be verified via a sumcheck protocol (in fact, it suffices to verify a random linear combination of these claims), and so an HIP can emulate access to the LDE over the extension field $\mathbb{K}$ by making queries to the LDE over field $\mathbb{F}$. We proceed to the formal statement and proof.

[21] More generally, for any code C that is locally testable and decodable (such as the LDE code), one can transform an HIP for the language $\mathcal{L}$ into an IPP for the language $C(\mathcal{L}) = \{C(x) : x \in \mathcal{L}\}$. Moreover, if the query location produced by the HIP verifier is uniformly distributed (which is typically the case), then local testability by itself suffices.

Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ and $\mathbb{K} = (\mathbb{K}_k)_{k\in\mathbb{N}}$ be constructible field ensembles such that $\mathbb{K}$ is a degree $s = s(k) \leq \log(k)$ field extension of $\mathbb{F}$ (i.e., $\mathbb{K}_k \cong \mathbb{F}_k^{s(k)}$, for every $k \in \mathbb{N}$). Let $H = (H_k)_{k\in\mathbb{N}} \subseteq \mathbb{F}$ and $\mathbb{G} = (\mathbb{G}_k)_{k\in\mathbb{N}} \subseteq \mathbb{K}$ be ensembles of subsets of $\mathbb{F}$ and $\mathbb{K}$, respectively. Let $m = m(k)$ and $\ell = \ell(k)$ be ensembles of integers such that $|H|^m = |\mathbb{G}|^\ell = k$.

Recall that for a given string $x \in \{0,1\}^k$, we define $\mathsf{LDE}_{\mathbb{F},H,m}$ as the unique individual degree $|H|-1$ polynomial $P : \mathbb{F}^m \to \mathbb{F}$ such that $P(z) = x_z$, for every $z \in H^m$ (where we identify the sets H^m and $[k]$ in some, computationally efficient, canonical way). Similarly, we define $\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}$ as the unique individual degree $|\mathbb{G}|-1$ polynomial $P : \mathbb{K}^\ell \to \mathbb{K}$ such that $P(z) = x_z$, for every $z \in \mathbb{G}^\ell$ (where now we identify $\mathbb{G}^\ell$ and $[k]$).

Lemma 4.11. *Let Let Let $\mathbb{F}$ and $\mathbb{K}$ be finite field ensembles as defined above. Let $\mathcal{L} = \cup_{k\in\mathbb{N}}\mathcal{L}_k$ be a language such that $\mathcal{L}_k \subseteq \{0,1\}^k$ for every $k \in \mathbb{N}$. Suppose that $\mathcal{L}$ has a ρ-round HIP, with respect to the code $\mathsf{LDE}_{\mathbb{K},\mathbb{G},r}$, with soundness error $\delta = \delta(k) \in [0,1]$ and communication complexity $c = c(k)$. Then, for every parameter $r = r(k) \geq 1$, the language $\mathcal{L}$ also has a $(\rho + r + 1)$-round HIP, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$, with soundness error $\left(\delta + O\left(\frac{|H|\cdot m}{|\mathbb{F}|}\right)\right)$ and communication $\left(c + \mathsf{poly}(k^{1/r}, |H|, r, \log|\mathbb{F}|)\right)$.*

Furthermore, the computational overhead for the verifier is $\mathsf{poly}(k^{1/r}, |H|, r, \log|\mathbb{F}|)$ and the computational overhead for the prover is $\mathsf{poly}(k)$.

We remark that for the furthermore part, we make use of the [KR09] constant-round variant of the [GKR08] protocol.

Proof of Lemma 4.11. Before presenting the desired HIP, we start with some algebraic notation and basic facts. Throughout this proof we use $\langle\cdot,\cdot\rangle_{\mathbb{K}}$ and $\langle\cdot,\cdot\rangle_{\mathbb{F}}$ to denote inner products over the fields $\mathbb{K}$ and $\mathbb{F}$, respectively.

Recall that elements in $\mathbb{K}$ are represented as vectors in $\mathbb{F}^s$. Let $b_1,\ldots,b_s : \mathbb{K}^* \to \mathbb{F}^*$ be functions defined as follows. For every $\alpha \in \mathbb{K}^*$ it holds that $\alpha = (b_1(\alpha),\ldots,b_s(\alpha))$. That is, the functions $b_1,\ldots,b_s$ decompose a vector $w \in \mathbb{K}^t$ into its s components over $\mathbb{F}^t$.

Proposition 4.12. *For every $w \in \mathbb{K}^k$ and $x \in \{0,1\}^k$ it holds that*

$$\langle w, x\rangle_{\mathbb{K}} = (\langle b_1(w), x\rangle_{\mathbb{F}}, \ldots, \langle b_s(w), x\rangle_{\mathbb{F}}).$$

Proof. We denote by $*$ multiplication in $\mathbb{K}$ and by $\cdot$ multiplication in $\mathbb{F}$. For $k = 1$ the proposition simply states that, for $w \in \mathbb{K}$ and $x \in \{0,1\}$ it holds that $w * x = (b_1(w)\cdot x,\ldots,b_s(w)\cdot x)$. The latter can be easily verified to hold for $x \in \{0,1\}$ by observing that, in both $\mathbb{K}$ and $\mathbb{F}$, multiplication by $x = 0$ always returns 0 and multiplication by $x = 1$ is identity. The proposition follows by induction on k. □

We proceed to describe the HIP $(\mathcal{P}',\mathcal{V}')$. Let $(\mathcal{P},\mathcal{V})$ be an HIP for $\mathcal{L}$, with respect to the code $\mathsf{LDE}_{\mathbb{K},\mathbb{G},r}$, with soundness error δ. To prove the lemma, we need to construct an HIP $(\mathcal{P}',\mathcal{V}')$ for $\mathcal{L}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$.

First, $\mathcal{P}'$ and $\mathcal{V}'$ emulate the HIP $(\mathcal{P},V)$. If $\mathcal{V}$ rejects, then $\mathcal{V}'$ immediately rejects. Otherwise, $\mathcal{V}$ outputs a pair $(z,\nu) \in \mathbb{K}^\ell \times \mathbb{K}$ with the associated claim that

$\left(\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}(x)\right)|_z = \nu$. Since $\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}$ is a $\mathbb{K}$-linear code, there exists a vector $w \in \mathbb{K}^k$ (that depends only on the code $\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}$ and the point z) such that $\left(\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}(x)\right)|_z = \langle w, x\rangle$, for every $x \in \{0,1\}^k$. Thus, $\mathcal{V}'$ only needs to verify that $\langle w, x\rangle_{\mathbb{K}} = \nu$.

For every $i \in [s]$, let $w_i \stackrel{\text{def}}{=} b_i(w) \in \mathbb{F}^k$ and let $\nu_i \stackrel{\text{def}}{=} b_i(\nu) \in \mathbb{F}$. By Proposition 4.12 the $\mathbb{K}$-linear equation $\langle w, x\rangle = \nu$ is equivalent to the following s $\mathbb{F}$-linear equations:

$$\forall i \in [s], \quad \langle w_i, x\rangle_{\mathbb{F}} = \nu_i. \tag{4.5}$$

The verifier $\mathcal{V}'$ chooses at random an $\mathbb{F}$-linear combination of these s linear equations. Namely, it selects at random $\gamma_1, \ldots, \gamma_s \in \mathbb{F}$ and sends these coefficients to the prover. Let $w' \stackrel{\text{def}}{=} \sum_{i\in[s]} \gamma_i \cdot w_i$ and $\nu' \stackrel{\text{def}}{=} \sum_{i\in[s]} \gamma_i \cdot \nu_i$ (where the summations are over $\mathbb{F}$). Note that if Eq. (4.5) holds then (with probability 1) $\langle w', x\rangle_{\mathbb{F}} = \nu'$, whereas if Eq. (4.5) does not hold then $\langle w', x\rangle_{\mathbb{F}} \neq \nu'$ with probability $1 - \frac{1}{|\mathbb{F}|}$ over the choice of $\gamma_1, \ldots, \gamma_s \in \mathbb{F}$. We next observe that the latter is an $\mathbb{F}$-linear claim about the input x and such claims can be directly solved using the sumcheck protocol.

Let $\tilde{x} : \mathbb{F}^m \to \mathbb{F}$ (resp., $\tilde{w'}$) be the low degree extension of the input x (resp., the vector $w' \in \mathbb{F}^k$) with respect to the field $\mathbb{F}$, set H and dimension m. That is, $\tilde{x}$ and $\tilde{w'}$ are the unique individual degree $|H| - 1$ polynomial that agree with x and w', respectively, on H^m. Let $P : \mathbb{F}^m \to \mathbb{F}$ be defined as the individual degree $2(|H| - 1)$ polynomial $P(z) = \tilde{w'}(z) \cdot \tilde{x}(z)$. Note that $\sum_{z\in H^m} P(z) = \langle w', x\rangle_{\mathbb{F}}$. Thus, checking that $\langle w', x\rangle_{\mathbb{F}} = \nu'$ is equivalent to $\sum_{z\in H^m} P(z) = \nu'$ which we can solve by having the prover and verifier run the sumcheck protocol with respect to the polynomial P.[22]

In case the sumcheck verifier rejects then $\mathcal{V}'$ immediately rejects. Otherwise, the result is a pair $(z'', \nu'') \in \mathbb{F}^m \times \mathbb{F}$. The prover sends to the verifier the value $\mu = \tilde{x}(z'')$. The verifier $\mathcal{V}'$ checks that $\mu \cdot \tilde{w'}(z'') = \nu''$ and if so it outputs (z'', μ), otherwise it rejects. This completes the description of the protocol.

Actually, one point about this protocol remains unclear - how can the verifier efficiently compute $\tilde{w'}(z'')$. If we were to ignore the *computational* resources of the verifier, then we could do this by brute force (e.g., in time roughly $|H|^m$), since $\tilde{w'}$ is independent of the input x. Nevertheless, we do aim for efficient verification and so we need to be able to compute $\tilde{w'}(z'')$ efficiently. We will do so by using additional interaction with the prover, based on the [KR09] variant of the [GKR08] protocol. We give a sketch in the following paragraph.

Computing $\tilde{w'}(z'')$. We start by taking a closer look at the vector w defined above. By the definition of the low degree extension (see Section 4.2.3), the vector $w \in \mathbb{K}^{G^\ell}$ is defined as $w_h = \beta(h, z)$, for every $h \in \mathbb{G}^\ell$, where β is as defined in Section 4.2.3. Thus,

[22] We remark that while we defined sumcheck as a protocol for the language $\mathcal{L} = \{x \in \mathbb{F}^k : \sum_{i\in[k]} x_i = 0\}$, a trivial, standard modification of the sumcheck protocol yields a protocol for $\mathcal{L}_\nu = \{x \in \mathbb{F}^k : \sum_{i\in[k]} x_i = \nu\}$, for every $\nu \in \mathbb{F}$.

we have that:

$$\tilde{w'}(z'') = \sum_{i\in[s]} \gamma_i \cdot b_i \left(\sum_{h\in H^m} \beta(h,z'') \cdot \beta(h,z) \right). \tag{4.6}$$

We observe that Eq. (4.6) can be represented as a (highly uniform) depth $O(\log(s) + m \cdot \log(H) + \log(|\mathbb{G}|) + \log(\ell)) = O(\log(k))$ Boolean circuit (on input z, z'') of size $s \cdot H^m \cdot \mathsf{poly}(|\mathbb{G}|, \ell, \log(|\mathbb{K}|)) = \mathsf{poly}(k)$. Applying the [KR09] variant of the [GKR08] protocol, we obtain an r-round interactive proof for verifying Eq. (4.6) in which the verifier runs in time $k^{O(1/r)} \cdot \mathsf{polylog}(|\mathbb{F}|)$ and with similar communication complexity.

Completeness. Fix $x \in \mathcal{L}$. By the completeness of $(\mathcal{P}, \mathcal{V})$, the verifier outputs $(z, \nu) \in \mathbb{K}^\ell \times \mathbb{K}$ such that $\left(\mathsf{LDE}^{\mathbb{K}}_{\mathbb{G},\ell}(x)\right)\big|_z = \nu$, or equivalently, $\langle w, x\rangle_{\mathbb{K}} = \nu$. By Proposition 4.12 this implies that $\langle w_i, x\rangle_{\mathbb{F}} = \nu_i$, for every $i \in [s]$. Therefore, for every $\gamma_1, \ldots, \gamma_s \in \mathbb{F}$ it holds that:

$$\langle w', x\rangle_{\mathbb{F}} = \sum_{i\in[s]} \gamma_i \cdot \langle w_i, x\rangle_{\mathbb{F}} = \sum_{i\in[s]} \gamma_i \cdot \nu_i = \nu'.$$

By definition of P, this means that $\sum_{z\in H^m} P(z) = \langle w', x\rangle_{\mathbb{F}} - \nu' = 0$ and the completeness of the sumcheck protocol implies that $\nu'' = P(z'') = \tilde{x}(z'') \cdot \tilde{w}(z'')$. Thus the verifier accepts when checking that $\mu \cdot \tilde{w}(z'') = \nu''$.

Soundness. Fix $x \notin \mathcal{L}$ and a cheating prover strategy $\mathcal{P}^*$. By the soundness of (P, V), with probability $1-\varepsilon$, the verifier either rejects (in which case $\mathcal{V}'$ also rejects) or outputs $(z, \nu) \in \mathbb{K}^\ell \times \mathbb{K}$ such that $\langle w, x\rangle_{\mathbb{K}} \neq \nu$. Assuming that the latter holds, by Proposition 4.12 there exists some $i^* \in [s]$ such that $\langle w_{i^*}, x\rangle_{\mathbb{F}} \neq \nu_{i^*}$. Therefore,

$$\begin{aligned}
\Pr\left[\langle w', x\rangle_{\mathbb{F}} = \nu'\right] &= \Pr\left[\sum_{i\in[s]} \gamma_i \cdot \langle w_i, x\rangle_{\mathbb{F}} = \sum_{i\in[s]} \gamma_i \cdot \nu_i\right] \\
&= \Pr\left[\gamma_{i^*} \cdot (\langle w_{i^*}, x\rangle_{\mathbb{F}} - \nu_{i^*}) = \sum_{i\neq i^*} \gamma_i \cdot (\nu_i - \langle w_i, x\rangle_{\mathbb{F}})\right] \\
&= 1/|\mathbb{F}|.
\end{aligned}$$

Thus, with probability $1 - \frac{1}{|\mathbb{F}|}$ it holds that $\langle w', x\rangle_{\mathbb{F}} \neq \nu'$, and in particular $\sum_{z\in H^m} P(z) \neq 0$ (where P is the polynomial as defined above).

Hence, by the soundness of the sumcheck protocol, with probability $\frac{|H|\cdot m}{|\mathbb{F}|}$ either the sumcheck verifier rejects (in which case we also reject) or it outputs a pair $(z'', \nu'') \in \mathbb{F}^m \times \mathbb{F}$ such that $P(z'') \neq \nu''$, or in other words $\tilde{x}(z'') \cdot \tilde{w}(z'') \neq \nu''$. Now, the prover sends over a value μ. If $\tilde{x}(z'') = \mu$ then, conditioned on the above event, the verifier rejects when checking that $\mu \cdot \tilde{w}(z'') = \nu'$. If $\tilde{x}(z'') \neq \mu$, then the verifier outputs a pair (z'', μ) such that $\left(\mathsf{LDE}_{\mathbb{F},H,m}(x)\right)_{z''} \neq \mu$ as desired. By a union bound, the overall soundness error is $\varepsilon + \frac{1}{|\mathbb{F}|} + \frac{|H|\cdot m}{|\mathbb{F}|}$.

Complexity. On top of the ρ rounds that $(\mathcal{P}, \mathcal{V})$ takes, the verifier also sends the message $(\gamma_1, \ldots, \gamma_s)$, but this message can be appended to the last message from $\mathcal{V}$ to $\mathcal{P}$. In addition, the two parties run an r-round sumcheck protocol and an r round variant of the [GKR08] protocol. There is one additional message from the prover with the value μ, so the overall number of rounds is $\rho + O(r)$.

The communication in the first part of the protocol (i.e., the emulation of (P, V)) is c. In addition, the verifier sends the linear combination $(\gamma_1, \ldots, \gamma_s)$ which takes $s \cdot \log|\mathbb{F}|$ bits. Lastly, both the sumcheck and the [GKR08] protocol add communication $\mathsf{poly}(k^{1/r}, |H|, r, \log|\mathbb{F}|)$ and the additional prover message is just $\log_2|\mathbb{F}|$ bits.

As for the verifier's complexity, beyond running the original $(\mathcal{P}, \mathcal{V})$ protocol, it runs the sumcheck and [GKR08] protocols which takes time $\mathsf{poly}(k^{1/r}, |H|, r, \log|\mathbb{F}|)$. The prover's additional time in running these two protocols is $\mathsf{poly}(|H|^m) = \mathsf{poly}(k)$. □

4.3.4 Booleanity Testing

In this subsection we show that HIPs can efficiently check that their input is the low-degree extension of a *Boolean* assignment. To do so, we follow an idea of Kalai and Raz [KR08], which was introduced in the context of constructing interactive PCPs.

We show a simple reduction from checking whether a polynomial $P : \mathbb{F}^m \to \mathbb{F}$ is Boolean valued in a subcube H^m (i.e., $P|_{H^m} H^m \to \{0, 1\}$) to checking whether a related (slightly higher degree) polynomial Q vanishes on H^m. Specifically, consider the polynomial $Q(x) = P(x) \cdot (1 - P(x))$, and observe that P is Boolean-valued in H^m if only if Q is identically zero in H^m. Checking whether a given polynomial is identically 0 (i.e., vanishes) on a subcube of its domain can be solved via a fairly well-known reduction to the sumcheck protocol. We also note that the reduction from P to Q is local (i.e., each query to Q can be computed by a single query to P) and therefore can be used in our setting.

We start by showing an HIP for inputs that vanish on a subcube. We first note that checking whether an individual degree $|H| - 1$ polynomial vanishes on the subcube H^m is trivial, since such a polynomial vanishes on H^m if and only if it vanishes on $\mathbb{F}^m$. The actual challenge is checking whether a higher degree polynomial (e.g., with individual degree $|\mathbb{G}| - 1$ for some $\mathbb{G}$ such that $|\mathbb{G}| > |H|$) vanishes on H^m.

Formally, for a given field ensemble $\mathbb{F}$, ensembles of subsets $H, \mathbb{G} \subseteq \mathbb{F}$ and dimension m, let $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$ be the set of all functions $f : G^m \to \mathbb{F}$ that vanish on H^m (i.e., $f|_{H^m} \equiv 0$).

The following proposition, which gives an HIP for $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$, is implicit in many classical constructions of PCPs (e.g., [BFLS91]). We include a proof in Section 4.6.4 for completeness.

Proposition 4.13. *Let $\mathbb{F}$ be a constructible field ensemble, let $H \subseteq \mathbb{G} \subseteq \mathbb{F}$ be ensembles of subsets, and let $m = m(k)$. For every $r = r(k) \leq \frac{\log(k)}{\log\log(k)}$, there exists an $(r+2)$-round (public-coin) HIP for $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},\mathbb{G},m}$, with soundness error $O\left(\frac{m\cdot|\mathbb{G}|}{|\mathbb{F}|}\right)$ and communication complexity $m \cdot \log(|\mathbb{F}|) + |\mathbb{G}|^{\lceil m/r \rceil} \cdot r \cdot \log|\mathbb{F}|$.*

The verifier runs in time $|\mathbb{G}|^{\lceil m/r \rceil} \cdot r \cdot \mathsf{polylog}(|\mathbb{F}|)$ *and the prover runs in time* $\mathsf{poly}(|\mathbb{F}|^m)$.

Denote by $\mathsf{Bool}_{\mathbb{F}}$ the set of all Boolean strings, viewed as a subset of $\mathbb{F}^*$. We show an HIP for Bool, which given access to a polynomial $P = \mathsf{LDE}_{\mathbb{F},H,m}(x)$ for some $x \in \mathbb{F}^k$, checks that $x \in \{0,1\}^k$.

Proposition 4.14. *Let* $\mathbb{F}$ *be a constructible field ensemble, let* $H \subseteq \mathbb{F}$*, and let* $m \in \mathbb{N}$*. For every* $r \in [m]$*, there exists an* $(r+2)$*-round (public-coin)* HIP *for* $\mathsf{Bool}_{\mathbb{F},H,m}$*, with respect to the code* $\mathsf{LDE}_{\mathbb{F},H,m}$*, with communication complexity* $O(r \cdot (2d + |H| - 1)^{m/r} \cdot \log|\mathbb{F}| + m \cdot \log(|\mathbb{F}|))$ *and soundness error* $O\left(\frac{m \cdot |H|}{|\mathbb{F}|}\right)$.

Proof. Given a degree d polynomial $P : \mathbb{F}^m \to \mathbb{F}$ such that $P = \mathsf{LDE}_{\mathbb{F},H,m}(x)$ for some $x \in \mathbb{F}^{|H|^m}$, define the degree $2d$ polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ as $Q(x) \stackrel{\text{def}}{=} P(x) \cdot (1 - P(x))$. Note that we can write $Q = \mathsf{LDE}_{\mathbb{F},\mathbb{G},m}(y)$ for $H \subseteq \mathbb{G} \subseteq \mathbb{F}$ and $y \in \mathbb{F}^{|G|^m}$, where $|G| = O(|H|)$.

Observe that P is Boolean-valued in H^m if and only if Q is identically 0 in H^m (this follows from the fact that the univariate polynomial $z \cdot (1 - z)$ has exactly two roots: 0 and 1). Thus, to verify that P is Boolean-valued in H^m, we run the HIP for $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$ in Proposition 4.13, with respect to the polynomial Q. Note that each query $Q(x)$ can be answered by a single query to P (specifically, by returning $P(x) \cdot (1 - P(x))$). Correctness follows from the correctness of the HIP for $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$. Communication complexity and soundness error follow from Proposition 4.13. □

4.4 The Hierarchy Theorem

In this section we prove our main theorem: a round hierarchy for IPPs.

Theorem 4.1 (IPP Hierarchy Theorem)**.** *There exists a language* $\mathcal{L}$ *and a gap function* $g(r) = \Theta(r^2)$ *such that for every constant* $r \geq 1$ *it holds that:*

1. **Upper Bound:** *There exists a* $g(r)$*-round (public-coin)* ε-IPP*, for* $\mathcal{L}$ *with communication complexity* $n^{O(1/r)}$ *and query complexity* $\mathsf{poly}(\log n, \varepsilon)$*. The verifier runs in time* $n^{O(1/r)} + \mathsf{poly}(\log(n), \varepsilon)$ *and the prover runs in time* $\mathsf{poly}(n)$*.*

2. **Lower Bound:** *For every* r*-round* IPP *for* $\mathcal{L}$*, with respect to proximity parameter* $\varepsilon = 1/10$*, that has query complexity* q *and communication complexity* c*, it holds that* $\max(c, q) = n^{\Omega(1/r)}$*.*

Furthermore, $\mathcal{L}$ *also has a* $\mathsf{polylog}(n)$*-round (public-coin)* ε-IPP *with communication* $\mathsf{polylog}(n)$ *and query complexity* $\mathsf{poly}(\log n, 1/\varepsilon)$*, and with a* $\mathsf{poly}(\log n, \varepsilon)$*-verifier and* $\mathsf{poly}(n)$*-time prover.*

The O and Ω notation in the theorem statement hide universal constants that do not depend on r. Note that any constant gap between the exponents in the upper and lower bounds can be obtained by increasing g by a suitable constant factor.

The rest of this section is devoted to the proof of Theorem 4.1. In Section 4.4.1 we present the language for which we show the IPP round hierarchy, in Section 4.4.3 we prove the lower bound (see Lemma 4.18), and in Section 4.4.2 we prove the upper bound (see Lemma 4.15). Combining Lemma 4.18 and Lemma 4.15 yields Theorem 4.1.

4.4.1 The Language: Encoded MOD3

Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ be a (constructible) field ensemble of characteristic 2 (i.e., each $\mathbb{F}_k$ is an extension field of $\mathsf{GF}(2)$). Let $H = (H_k)_{k\in\mathbb{N}}$ be an ensemble of subsets $H \subseteq \mathbb{F}$ and let $m = m(k)$ be the dimension such that $|H| = \log(k)$, $m = \frac{\log(k)}{\log\log(k)}$ and $|\mathbb{F}| = \Theta(|H|^2 m)$. Denote $n \stackrel{\text{def}}{=} |\mathbb{F}^m|$, and note that $|H|^m = k$ and that $k^2 \leq n \leq k^3$.

We first define an (auxiliary) language $\mathcal{L}_{\mathsf{MOD3}}$, where:

$$\mathcal{L}_{\mathsf{MOD3}} \stackrel{\text{def}}{=} \{x \in \{0,1\}^* \ : \ \mathsf{wt}(x) = 0 \pmod 3\}\,.$$

That is, $\mathcal{L}_{\mathsf{MOD3}}$ simply consists of strings whose Hamming weight is divisible by 3. The actual language for which we prove the IPP lower bound is $\mathsf{Enc\text{-}MOD3} = \mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L}_{\mathsf{MOD3}})$. That is,

$$\mathsf{Enc\text{-}MOD3} = \left\{\mathsf{LDE}_{\mathbb{F},H,m}(x) \ : \ x \in \mathcal{L}_{\mathsf{MOD3}}\right\}.$$

Or in words, Enc-MOD3 consists of all m-variate polynomials over $\mathbb{F}$, of individual degree $|H| - 1$, that take Boolean values in H^m such that the integer sum over all elements in H^m is divisible by 3.

4.4.2 The Upper Bound

In this section, we construct an IPP for Enc-MOD3. This IPP suffices both for the results in the constant-round regime and poly-logarithmic round regime of Theorem 4.1.

Lemma 4.15. *For every $r = r(n) \leq \frac{\log(n)}{\log\log(n)}$, there exists an $O(r^2)$-round public-coin ε-IPP for Enc-MOD3 with perfect completeness and soundness error $1/2$. The communication complexity is $n^{O(1/r)}$ and the query complexity is $\mathsf{poly}(\log(n), 1/\varepsilon)$. Furthermore, the verifier runs in time $\left(n^{O(1/r)} + \mathsf{poly}(\log(n), 1/\varepsilon)\right)$ and the prover runs in time $\mathsf{poly}(n)$.*

The main step in the proof of Lemma 4.15 is the construction of an HIP for the related language $\mathcal{L}_{\mathsf{MOD3}}$ (defined above), with respect to the LDE code (with the parameters that were specified in Section 4.4.1). Given this HIP, Lemma 4.15 follows by using a generic transformation from HIPs (with respect to the LDE encoding) into IPPs, which we establish in Lemma 4.10.

Before constructing this HIP, as an intermediate goal, we first construct an HIP for $\mathcal{L}_{\mathsf{MOD3}}$, with respect to the low-degree extension with different parameters than those that were set in Section 4.4.1. Specifically, we shall use a larger field $\mathbb{K}$, whose size is polynomially related to k (rather than poly-logarithmic). In particular, there will be a dependence between the size of $\mathbb{K}$ and the number of rounds in the HIP. Later we will use

a generic transformation to convert this HIP into one in which the low degree extension can be over a much smaller field (e.g., of poly-logarithmic size), which in particular does not depend on the number of rounds.

The following lemma, which is the main lemma proved in this section, gives an HIP for $\mathcal{L}_{\mathsf{MOD3}}$ over the relatively large field $\mathbb{K}$.

Lemma 4.16. *Let $r = r(k) \geq 1$, let $\mathbb{K} = (\mathbb{K}_k)_{k\in\mathbb{N}}$ be a constructible field ensemble of size $|\mathbb{K}| = \Omega(r^2 \cdot k^{2/r})$, let $\mathbb{G} = (\mathbb{G}_k)_{k\in\mathbb{N}} \subseteq \mathbb{K}$ be an ensemble of subsets of $\mathbb{K}$ of size $|G| = k^{1/r}$.*

Then, there exists an r^2-round public-coin HIP for $\mathcal{L}_{\mathsf{MOD3}}$, with perfect completeness and soundness error $O\left(\frac{r^2 \cdot k^{2/r}}{|\mathbb{K}|}\right)$. The communication complexity is $O(r^2 \cdot k^{2/r} \cdot \log|\mathbb{K}|)$. The verifier runs in time $k^{O(1/r)} \cdot \mathsf{poly}(r, \log(k))$ and the prover runs in time $\mathsf{poly}(|\mathbb{K}|^r)$.

(See Section 4.1.2 for a high-level overview of the proof.)

Proof. Let $r = r(k) \geq 1$. Recall that $\mathbb{K} = (\mathbb{K}_k)_{k\in\mathbb{N}}$ is a constructible field ensemble field of size $|\mathbb{K}| = \Omega(r^2 \cdot k^{1/r})$ and that $\mathbb{G} = (\mathbb{G}_k)_{k\in\mathbb{N}}$ is an ensemble of subsets of size $|\mathbb{G}| = k^{1/r}$. Since we only deal with a single input length k (which we think of as varying), in the following we omit the subscripts and use $\mathbb{K}$ (resp., $\mathbb{G}$) when we actually mean $\mathbb{K}_k$ (resp., $\mathbb{G}_k$).

Denote by $t \stackrel{\text{def}}{=} |\mathbb{G}| = k^{1/r}$ and fix a canonical ordering $\alpha_1, \ldots, \alpha_t$ of the set of elements in $\mathbb{G}$ (i.e., $\mathbb{G} = \{\alpha_1, \ldots, \alpha_t\}$). Let $\mathsf{MOD3}_t : \{0,1,2\}^t \to \{0,1,2\}$ be defined as $\mathsf{MOD3}_t(\sigma_1, \ldots, \sigma_t) \stackrel{\text{def}}{=} \sum_{j\in[t]} \sigma_j \pmod 3$.

Fix an input $x \in \{0,1\}^k$. As described in Section 4.1.2, we define polynomials $V_0, \ldots, V_r$ that contain sums, modulo 3, of certain intervals in x. Taking the [GKR08] view, one can consider a depth r formula, with fan-in $t = k^{1/r}$, composed of $\mathsf{MOD3}_t$ gates, that computes the sum mod 3 of its input (see Fig. 4.1). Viewed this way, each polynomial V_i corresponds to the low degree extension of the i^{th} layer of this formula (counting from output to input).

Since $|\mathbb{G}^r| = k$, we can associate elements in $\mathbb{G}^r$ with the integers in the set $\{1, \ldots, k\}$ in the natural way. Thus, we can view the input $x \in \{0,1\}^k$ as a function, which we denote by $V_r : \mathbb{G}^r \to \{0,1\}$, that is defined as $V_r(p) = x_p$, for every $p \in \mathbb{G}^r$. We define functions $V_0, \ldots, V_{r-1}$ via backward recursion as follows. For every $i \in [r]$, let $V_{i-1} : \mathbb{G}^{i-1} \to \{0,1,2\}$ be defined as:

$$\forall p \in \mathbb{G}^{i-1}, \quad V_{i-1}(p) = \mathsf{MOD3}_t\big(V_i((p,\alpha_1)), \ldots, V_i((p,\alpha_t))\big), \tag{4.7}$$

where (p,α) denotes the element in $\mathbb{G}^i$ which is obtained by concatenating $p \in \mathbb{G}^{i-1}$ with $\alpha \in \mathbb{G}$. For the case $i = 0$, we define $\mathbb{G}^0 = \{\perp\}$, where $\perp$ is defined as the empty string (in particular $(\perp, p) = (p, \perp) = p$), and note that, for $i = 1$, Eq. (4.7) reduces to $V_0(\perp) = \mathsf{MOD3}_t\left(V_1(\alpha_1)\right), \ldots, V_1(\alpha_t)\big)$.

As noted above, intuitively, each V_i should be thought of as specifying a sum of certain intervals in the input, according to a partition (which depends on i). For example, V_r contains the value of each of the individual coordinate of x (i.e., the most fine grained partition) whereas V_0 contains the overall sum (i.e., the coarsest partition). More generally, we have the following immediate fact:

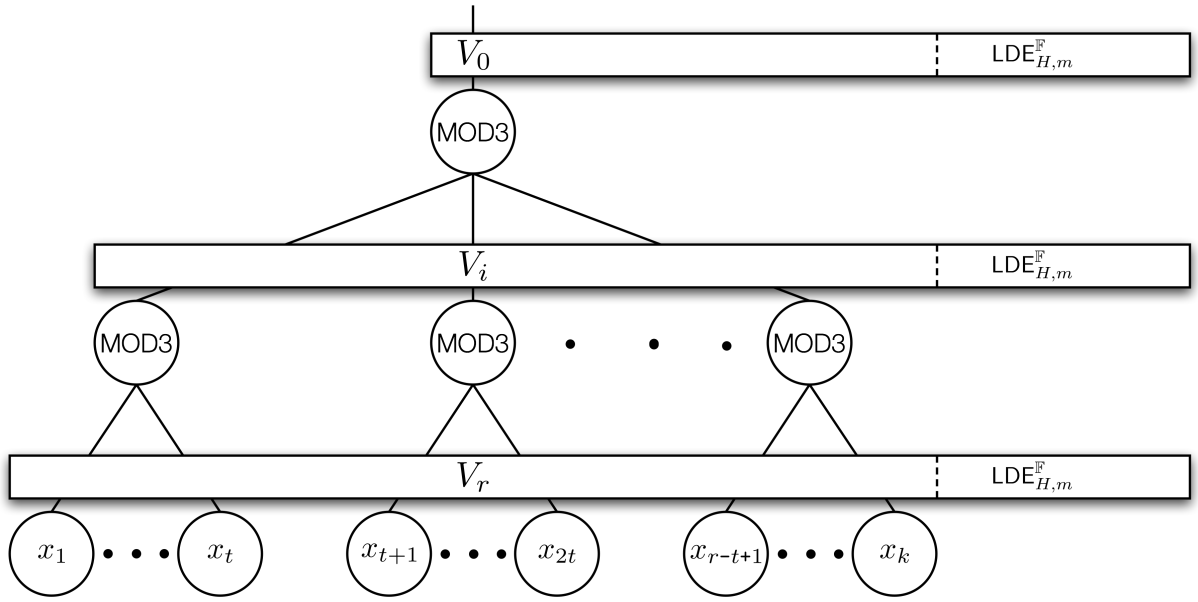

Figure 4.1: The recursive depth r formula of fan-in $k^{1/r}$ that computes the sum mod 3 of its input $x \in \{0,1\}^k$, and the low-degree extension of each one of the formula's layers when evaluated on x.

Fact 4.17. *For every $i \in \{0,\ldots,r\}$ and $p \in \mathbb{G}^i$ it holds that $V_i(p) = \sum_{q\in\mathbb{G}^{r-i}} x_{(p,q)}$ (mod 3) (where (p,q) denotes the concatenation of the two vectors p and q).*

In particular, by setting $i = 0$, we have that $V_0(\perp) = \sum_{q\in\mathbb{G}^r} x_q \pmod 3 = \sum_{i\in[k]} x_i \pmod 3$.

For the rest of the proof we use $\tilde{f}$ to denote the low degree extension of a function f (see Section 4.2.3 for details on the low degree extension encoding) and associate the integers 0, 1 and 2 with three distinct elements in $\mathbb{K}$ in some canonical way (so that we can view $\{0,1,2\} \subseteq \mathbb{K}$). Let $\widetilde{\mathsf{MOD3}}_t : \mathbb{K}^t \to \mathbb{K}$ be the unique individual degree 2 extension of the function $\mathsf{MOD3} : \{0,1,2\}^t \to \{0,1,2\}$ with respect to the field $\mathbb{K}$, the subset $\{0,1,2\} \subseteq \mathbb{K}$, and dimension t. For every $i \in [r]$, let $\tilde{V}_i : \mathbb{K}^i \to \mathbb{K}$ be the unique individual degree $|\mathbb{G}| - 1$ extension of V_i with respect to the field $\mathbb{K}$, the set $\mathbb{G}$ and dimension i. Let $\tilde{V}_0 \equiv V_0$ (recall that $V_0 : \{\perp\} \to \{0,1,2\}$ is just a singleton value $V_0(\perp) \in \mathbb{K}$). Observe that the polynomial $\tilde{V}_r$ is the low degree extension of the input x with respect to the field $\mathbb{K}$, the set $\mathbb{G}$ and dimension r.

A crucial fact that we will use is that, for every $i \in [r]$, each point in $\tilde{V}_{i-1}$ can be expressed as a certain type of composition of the low degree polynomial $\tilde{V}_i$ with the low degree polynomial $\widetilde{\mathsf{MOD3}}$. More specifically, using the properties of the low degree

extension (see Section 4.2.3), it holds that for every $i \in [r]$ and $z \in \mathbb{K}^{i-1}$:

$$\begin{aligned}\tilde{V}_{i-1}(z) &= \sum_{p \in \mathbb{G}^{i-1}} \beta(z,p) \cdot V_{i-1}(p) \\ &= \sum_{p \in \mathbb{G}^{i-1}} \beta(z,p) \cdot \mathsf{MOD3}_t\big(V_i((p,\alpha_1)),\ldots,V_i((p,\alpha_t))\big) \\ &= \sum_{p \in \mathbb{G}^{i-1}} \beta(z,p) \cdot \widetilde{\mathsf{MOD3}}_t\big(\tilde{V}_i((p,\alpha_1)),\ldots,\tilde{V}_i((p,\alpha_t))\big). \end{aligned} \tag{4.8}$$

where the polynomial β is as defined in Section 4.2.3, and the last equality uses the fact that $\tilde{V}_i|_{\mathbb{G}^i} \equiv V_i|_{\mathbb{G}^i}$ and $\widetilde{\mathsf{MOD3}}_t|_{\{0,1,2\}^t} \equiv \mathsf{MOD3}_t|_{\{0,1,2\}^t}$.

Using the above definition, we proceed to describe our HIP for $\mathcal{L}_{\mathsf{MOD3}}$. The protocol is performed in r phases, each of which takes at most r rounds of interaction (for a total of at most r^2 rounds). We begin the protocol with a claim about the value of a single point (as a matter of fact, the only point) in $\tilde{V}_0$ (recall that, by Fact 4.17, the value of $\tilde{V}_0(\perp)$ corresponds to the desired output - the sum modulo 3 of the input bits). In the i^{th} phase, we reduce the task of verifying the value of a single (arbitrary) point in $\tilde{V}_{i-1}$ to verifying the value of a single point in $\tilde{V}_i$. Thus, after r phases, we have reduced the problem of verifying $\tilde{V}_0(\perp) = \sum_{j \in [k]} x_j \pmod 3$ to verifying a single point in $\tilde{V}_r$, which is the low degree extension of the input x.

Define $z_0 = \perp$ and $\nu_0 = 0$. The original claim is that $\tilde{V}_r(z_0) = \nu_0$. We shall maintain the invariant that for every phase $i \in \{0,\ldots,r\}$, at the end of the i^{th} phase, the prover and verifier both know a vector $z_i \in \mathbb{K}^i$ and a scalar $\nu_i \in \mathbb{K}$ such that the current claim is that $\tilde{V}_i(z_i) = \nu_i$. Thus, the goal of the i^{th} phase is to (interactively) reduce the claim $\tilde{V}_{i-1}(z_{i-1}) = \nu_{i-1}$ to a claim of the form $\tilde{V}_i(z_i) = \nu_i$ (for some z_i and ν_i that are generated during the i^{th} phase):

Phase i:

1. **Reduce to Claim about t Points in $\tilde{V}_i$:** The phase begins with a claim that $\nu_{i-1} = \tilde{V}_{i-1}(z_{i-1})$. By Eq. (4.8) this is equivalent to:

$$\nu_{i-1} = \sum_{p \in \mathbb{G}^{i-1}} \beta(z_{i-1},p) \cdot \widetilde{\mathsf{MOD3}}_t\big(\tilde{V}_i((p,\alpha_1)),\ldots,\tilde{V}_i((p,\alpha_t))\big) \tag{4.9}$$

   We now observe that the right-hand side of Eq. (4.9) corresponds to a sum, over an $(i-1)$-dimensional subcube, of the values of a low degree polynomial. Specifically, denote

$$f_{i-1}(w) = \beta(z_{i-1},w) \cdot \widetilde{\mathsf{MOD3}}_t\big(\tilde{V}_i((w,\alpha_1)),\ldots,\tilde{V}_i((w,\alpha_t))\big),$$

   and observe that f_{i-1} has *total* degree $(t-1)\cdot(i-1)+2t\cdot(t-1)\cdot i \leq 3t^2 r$ polynomial. Eq. (4.9) can be rewritten as $\nu_{i-1} = \sum_{p \in \mathbb{G}^{i-1}} f_{i-1}(p)$. The prover and verifier run an i-round sumcheck protocol with respect to this equation.

In case the sumcheck verifier rejects, our verifier immediately rejects. Otherwise, the output of the sumcheck protocol is a (random) point $w_{i-1} \in \mathbb{K}^{i-1}$ and value $\gamma_{i-1} \in \mathbb{K}$ with an associated alleged claim that $\gamma_{i-1} = f_{i-1}(w_{i-1})$.

2. **Query Reduction:** At this point the verifier has a claim regarding the values of t points of $\tilde{V}_i$ (specifically, the claim $\gamma_{i-1} = f_{i-1}(w_{i-1})$ refers to the points $(w_i, \alpha_1), \ldots, (w_i, \alpha_t)$). The goal of this step is to reduce this more elaborate claim to a claim about a *single* point in $\tilde{V}_i$:

   (a) The prover sends to the verifier the univariate degree $t-1$ polynomial $P_i : \mathbb{K} \to \mathbb{K}$ defined as $P_i(\eta) = \tilde{V}_i(w_i, \eta)$ (given by its t coefficients).

   (b) The verifier receives a degree $t-1$ polynomial Q_i (which is allegedly equal to P_i). The verifier checks that $\gamma_i = \beta(z_{i-1}, w_{i-1}) \cdot \widetilde{\mathsf{MOD3}}_t\big(Q_i(\alpha_1), \ldots, Q_i(\alpha_t)\big)$. If the check fails then the verifier immediately rejects and halts. Otherwise, the verifier chooses a random field element $\eta_i \in \mathbb{K}$ and sends η_i to the prover.

   (c) The claim for the next round is that $\tilde{V}_i(z_i) = \nu_i$, where $\nu_i = P_i(\eta_i)$ and $z_i = (w_i, \eta_i)$.

After all of the r phases are complete, the verifier outputs (z_r, ν_r) and the associated claim is that $\tilde{V}_r(z_r) = \nu_r$. Since $\tilde{V}_r$ is simply the low degree extension of the input x, the latter is a claim about a single point in the low degree extension of the input as required by the definition of an HIP verifier.

Complexity. Since the communication complexity of each sumcheck is $O(r \cdot k^{1/r} \cdot \log|\mathbb{K}|)$, the total communication complexity is $O(r^2 \cdot k^{1/r} \cdot \log|\mathbb{K}|)$. As for the round complexity, the i^{th} phase uses a sumcheck of $i \leq r$ rounds of interaction. Moreover, each sumcheck concludes with a message from the prover to the verifier so we can "piggy-back"and attach the polynomial Q_i to that last message from the prover and send back the value η_i as our response (which is still part of the last round of the sumcheck protocol) so each phase just takes $\leq r$ rounds and overall we have $\leq r^2$ rounds.

As for computational complexity, in the first step of each phase, the parties invoke a sumcheck protocol in which, by Lemma 4.9, the verifier runs in time $k^{O(1/r)} \cdot r \cdot \mathsf{polylog}|\mathbb{K}|$, and the prover runs in time $\mathsf{poly}(|\mathbb{K}|^r)$. In the second step of each phase, the prover computes and sends P_i, which clearly can be done in time $\mathsf{poly}(|\mathbb{K}|^r)$, and the verifier computes γ_i, which boils down to evaluating the functions β and $\widetilde{\mathsf{MOD3}}_t$ at a single point, which can be done in time $\mathsf{poly}(t, \log k) = k^{O(1/r)} \cdot \mathsf{poly}(\log|\mathbb{K}|)$ (see Proposition 4.4 and Section 4.6.5 for the time complexity of computing β and $\widetilde{\mathsf{MOD3}}_t$, respectively). The obtain the total running times (for the entire r phases), we multiply the time per phase by r.

Completeness. Perfect completeness follows readily from the construction (and the prefect completeness of the sumcheck protocol).

Soundness. To conclude the proof of Lemma 4.16 we only need to show that soundness holds. Our analysis follows the soundness analysis in [GKR08, Theorem 3.1].

Fix an input $x \in \{0,1\}^k$ such that $\sum_{i\in[k]} x_i \not\equiv 0 \pmod 3$ (i.e. $x \notin \mathcal{L}_{\mathsf{MOD3}}$) and a cheating strategy $\mathcal{P}^*$. Denote by A the event that the verifier does not reject in the interaction with the prover $\mathcal{P}^*$. For every $i \in \{0,1,\ldots,r\}$, denote by T_i the event that $\tilde{V}_i(z_i) = \nu_i$. Note that since $\sum_{i\in[k]} x_i \neq 0 \pmod 3$ it holds that the event $\neg T_0$ occurs with probability 1. For every $i \in [r]$, let E_i denote the event that the polynomial Q_i that the prover sent is indeed identical to $P_i(\eta) = \tilde{V}_i(w_i, \eta)$.

Our analysis will be based on the following two claims.

Claim 4.17.1.

$$\Pr\left[A \wedge E_i \mid \neg T_{i-1}\right] \leq \frac{3t^2 r}{|\mathbb{K}|}.$$

Proof. Assume that the event T_{i-1} occurs. Then, by the soundness of the sumcheck protocol, with probability $\frac{3t^2 r}{|\mathbb{K}|}$ (over the verifier's coins in the sumcheck protocol) it holds that $f_{i-1}(w_{i-1}) \neq \gamma_{i-1}$, or in other words $\beta(z_{i-1}, w_{i-1}) \cdot \widetilde{\mathsf{MOD3}}_t\big(\tilde{V}_i((w_{i-1},\alpha_1)),\ldots,\tilde{V}_i((w_{i-1},\alpha_t))\big) \neq \gamma_{i-1}$. If the latter happens and then the prover sends the correct polynomial P_i (i.e., the event E_i occurs) then the verifier immediately rejects in Item 2b. Thus, with probability $1 - \frac{3t^2 r}{|\mathbb{K}}$, either the event $\neg A$ or $\neg E_i$ must occur. □

On the other hand:

Claim 4.17.2.

$$\Pr\left[T_i \mid \neg E_i\right] \leq \frac{t}{|\mathbb{K}|}.$$

Proof. The event $\neg E_i$ implies that the polynomial Q_i sent by the prover differs from the correct polynomial P_i. Since both Q_i and P_i are degree $t-1$ polynomials, they can agree on at most $t-1$ points, and so, with probability $1 - \frac{t-1}{|\mathbb{K}|}$ over the choice of η_i it holds that $\nu_i = Q(\eta_i) \neq P(\eta_i) = \tilde{V}_i((w_i,\eta_i)) = \tilde{V}_i(z_i)$. □

Finally, observe that the probability that the verifier errs is simply $\Pr[A \wedge \neg T_r]$, which we can bound (using Claim 4.17.1, Claim 4.17.2 and elementary probability theory) as

follows:

$$\begin{aligned}
\Pr[A \wedge T_r] &= \Pr[A \wedge \neg T_0 \wedge T_r] \\
&\leq \Pr\left[\exists i \in [r] \text{ such that } A \wedge \neg T_{i-1} \wedge T_i\right] \\
&\leq \sum_{i=1}^{r} \Pr\left[A \wedge \neg T_{i-1} \wedge T_i\right] \\
&= \sum_{i=1}^{r} \left(\Pr[A \wedge \neg T_{i-1} \wedge T_i \wedge E_i] + \Pr[A \wedge \neg T_{i-1} \wedge T_i \wedge \neg E_i]\right) \\
&\leq \sum_{i=1}^{r} \left(\Pr\left[A \wedge E_i \mid \neg T_{i-1}\right] + \Pr\left[T_i \mid \neg E_i\right]\right) \\
&\leq \sum_{i=1}^{r} \left(\frac{3t^2 r}{|\mathbb{K}|} + \frac{t}{|\mathbb{K}|}\right) \\
&\leq \frac{4t^2 r^2}{|\mathbb{K}|}.
\end{aligned}$$

This concludes the proof of Lemma 4.16. □

Lemma 4.16 provides an r^2-round HIP for $\mathcal{L}_{\mathsf{MOD3}}$, with respect to the code $\mathsf{LDE}_{\mathbb{K},\mathbb{G},r}$, where $\mathbb{K}$ is a field ensemble of size $\Theta\left(r^2 \cdot k^{2/r}\right)$. We now use a general result, which is stated and proved in Section 4.3, which transforms any such HIP, in which the field $\mathbb{K}$ has small characteristic, into an HIP over the code $\mathsf{LDE}_{\mathbb{F},H,m}$ where the size of the field $\mathbb{F}$ is now only *poly-logarithmic* in k. Specifically, by applying Lemma 4.11 to the protocol of Lemma 4.16, and using a field $\mathbb{K}$ which is an extension field of some field $\mathbb{F}$ of size $\mathsf{polylog}(k)$, we obtain the following corollary:

Corollary 4.2. *Let $\mathbb{F} = (\mathbb{F}_k)_{k\in\mathbb{N}}$ be a constructible field ensemble, $H = (H_k)_{k\in\mathbb{N}} \subseteq \mathbb{F}$ be an ensembles of subsets of $\mathbb{F}$ and let $m = m(k)$ be a dimension such that $|H| = \log(k)$, $m = \frac{\log(k)}{\log\log(k)}$ and $|\mathbb{F}| = \Theta(|H| \cdot m)$.*

Then, for every parameter $r = r(k) \leq \frac{\log(k)}{\log\log(k)}$, the language $\mathcal{L}_{\mathsf{MOD3}}$ has an has an $O(r^2)$-round (public-coin) HIP with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$ with soundness error $1/2$ and communication complexity $k^{O(1/r)}$. The verifier runs in time $k^{O(1/r)}$ and the prover runs in time $\mathsf{poly}(k)$.

Lemma 4.15 follows from Corollary 4.2 by applying Lemma 4.10, which is a generic transformation from any HIP, over the low degree extension encoding, into an IPP.

4.4.3 The Lower Bound

Lemma 4.18. *Let $r = r(k) \geq 1$ be a constant. For every r-round IPP for Enc-MOD3, with respect to proximity parameter $\varepsilon = 1/10$, with query complexity q and communication complexity $c \geq \Omega(\log n)$, it holds that $\max(c, q) = n^{\Omega(1/r)}$.*

We remark that our proof of Lemma 4.18 gives a similar result even for super constant values of r (as long as $r = O\left(\sqrt{\frac{\log(n)}{\log\log(n)}}\right)$) but for simplicity we restrict ourselves to constant r. We also remark that the constants in the lemma's statement can be improved but we avoid optimizing them for sake of readability.

Proof. Throughout the proof of Lemma 4.18 all proofs of proximity refer to proximity parameter $\varepsilon = 1/10$.

The following proposition, due to [RVW13] (building on [BM88, GS86, GVW02]), shows that to prove Lemma 4.18, it suffices to prove a lower bound for AMPs, which are *public-coin* IPPs with only a *single* round of interaction between the verifier and prover. More precisely, in an AMP for a language $\mathcal{L}$, the verifier first sends a random string r to the prover, who responds with a proof π, which can depend on both the input x and the verifier's message r. Then, given π (and based on its original random coins r), the verifier needs to decide whether to accept or reject. (Note that the verifier is not allowed to toss additional coins after receiving the message from the prover.)

Proposition 4.19 (IPP to AMP)**.** *If there exists an r-round (public or private coin)* IPP *for a language $\mathcal{L}$, with communication complexity $c \geq \log(n)$ and query complexity q, then there exists an* AMP *for $\mathcal{L}$ with communication complexity $c^{r+2} \cdot (\log(c) \cdot r)^{O(r)}$ and query complexity $c^{r+1} \cdot q \cdot (\log(c) \cdot r)^{O(r)}$.*

The proof of Proposition 4.19, which appears in [RVW13, Section 4], proceeds by observing that the private-coin to public-coin transformation of [GS86] as well as the round reduction transformation of [BM88, GVW02], which are transformations on standard interactive proofs, can be applied to IPPs as well.

Thus, given Proposition 4.19, and using the fact that r is constant, to prove Lemma 4.18 it suffices to show that every AMP for Enc-MOD3 with query complexity q and communication complexity c satisfies $\max(c, q) = n^{\Omega(1)}$, or equivalently, since $n = O(k^3)$, that $\max(c, q) = k^{\Omega(1)}$. The following proposition, which is inspired by the [RVW13] lower bound, shows that AMPs for properties of *linear codes* can be viewed as distributions over (relatively) small DNF*s of parities*. By DNF of parities, we refer to depth 3 circuits whose bottom layer consists of parity gates, middle layer consists of AND gates and top layer is a single OR gate. In the following we denote such circuits by $\mathsf{DNF}_{\oplus}$.

Proposition 4.20. *Let $\mathbb{F}$ be an extension field of* GF(2)*, let $C : \mathbb{F}^k \to \mathbb{F}^n$ be an $\mathbb{F}$-linear code, and let $f : \{0,1\}^k \to \{0,1\}$. If there exists an* AMP *for $\Pi_f \stackrel{\text{def}}{=} \{C(x) : x \in \{0,1\}^k \wedge f(x) = 1\}$ with communication complexity $c \geq \log(n)$ and query complexity q, then there exists a distribution $\mathcal{D}$ over $\mathsf{DNF}_{\oplus}$ circuits of size $2^{O(c+q\cdot\log_2(|\mathbb{F}|))}$ such that $\Pr_{\varphi\sim\mathcal{D}}[\varphi(x) = f(x)] \geq 0.9$, for all $x \in \{0,1\}^k$.*

Proof. Let $\mathcal{V}$ be an AMP verifier for Π. We assume without loss of generality that $\mathcal{V}$ has soundness error at most 0.1 (e.g., by repeating the protocol in parallel $O(1)$ times). Recall that in an AMP protocol, for a given input $y \in \mathbb{F}^n$, the verifier first sends a random string r, then the prover replies with an alleged proof $\pi = \pi(r, y)$, and finally the verifier

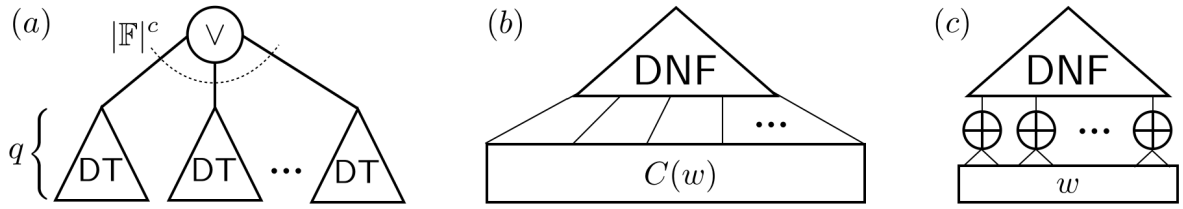

Figure 4.2: After fixing the randomness, an AMP for Π can be expressed as follows: (a) a disjunction over $O(2^c)$ decision trees of depth $q \cdot \log(|\mathbb{F}|)$, (b) a DNF formula with $O(2^{c+q\cdot\log(|\mathbb{F}|)})$ clauses of width $q \cdot \log|\mathbb{F}|$ over the linear code $C(w)$, and (c) a $\mathsf{DNF}_\oplus$ circuit of size $\tilde{O}(2^{c+q\cdot\log(|\mathbb{F}|)})$ over x.

makes queries to y and decides whether to accept or reject. Denote by $\mathcal{V}^y_{r,\pi}$ the output of the verifier for a fixed string r, given oracle access to y and direct access to π.

For a fixed string r and alleged proof π, the verifier $\mathcal{V}^y_{r,\pi}$ can be represented as an $|\mathbb{F}|$-ary decision tree of depth q (on input y), which we denote by $D_{r,\pi} : \{0,1\}^k \to \{0,1\}$. The completeness and soundness requirements of an AMP guarantee that for a fixed string r, the verifier accepts an input $C(x)$ if and only if there exists a string π such that $\mathcal{V}^{C(x)}_{r,\pi} = 1$. Thus, $\mathcal{V}^{C(x)}_{r,\pi} = \bigvee_{\pi\in\{0,1\}^{O(c)}} D_{r,\pi}\big(C(x)\big)$ (see Fig. 4.2(*a*)). Observe that by viewing elements of $\mathbb{F}$ in their bit-representation and assigning a clause for each accepting leaf in the decision tree, each $D_{r,\pi}$ can be represented as a *binary* DNF formula with $|\mathbb{F}|^{O(q)}$ clauses of width $O(q \cdot \log|\mathbb{F}|)$. Merging the two consecutive layers of disjunctions, we obtain a binary DNF formula that on input $y \in \mathbb{F}^n$ computes $\mathcal{V}^y_{r,\pi}$ with $2^{O(c+q\cdot\log_2(|\mathbb{F}|))}$ clauses of width $O(q \cdot \log_2(|\mathbb{F}|))$ each (see Fig. 4.2(*b*) for an illustration).

We next observe that every linear combination over the field $\mathbb{F}$, which is an extension field of $\mathsf{GF}(2)$, can be represented by $\log_2(|\mathbb{F}|)$ linear combinations over $\mathsf{GF}(2)$.[23] Thus, we can view the function $C : \mathbb{F}^k \to \mathbb{F}^n$, which is an $\mathbb{F}$-linear function, as a $\mathsf{GF}(2)$-linear function $C : \mathsf{GF}(2)^{k\cdot\log_2(|\mathbb{F}|)} \to \mathsf{GF}(2)^{n\cdot\log_2(|\mathbb{F}|)}$. Hence, for every random string r, there exists a $\mathsf{DNF}_\oplus$ circuit of size:

$$2^{O(c+q\cdot\log_2(|\mathbb{F}|))} \cdot q \cdot \log_2(|\mathbb{F}|) + n \cdot \log_2(|\mathbb{F}|) = 2^{O(c+q\cdot\log(|\mathbb{F}|))}$$

(which is constructed by composing the code C with the DNF $\bigvee_{\pi\in\{0,1\}^{O(c)}} D_{r,\pi}$) that on input $x \in \{0,1\}^k$ outputs 1 if and only if there exists a proof π that $\mathcal{V}$ would accept, given input $C(x)$.

Therefore, there exists a distribution $\mathcal{D}$ over $\mathsf{DNF}_\oplus$s of size $2^{O(c+q\cdot\log(|\mathbb{F}|))}$ such that for every $x \in \{0,1\}^k$, it holds that $\Pr_{\varphi\in\mathcal{D}}[\varphi(x) = f(x)] \geq 0.9$. This concludes the proof of Proposition 4.20. □

Let $f_{\mathsf{MOD3}} : \{0,1\}^k \to \{0,1\}$ such that $f_{\mathsf{MOD3}} = 1$ if and only if $\sum_{i\in[k]} x_i \equiv 0 \pmod 3$. By Proposition 4.20, choosing $\Pi = \mathsf{Enc\text{-}MOD3}$, $C = \mathsf{LDE}^{\mathbb{F}}_{H,m}$, $f = f_{\mathsf{MOD3}}$, and using

[23] Fix a linear combination $\alpha \in \mathbb{F}^t$ over $\mathbb{F}$ (the extension field). For every $i \in [\log_2(|\mathbb{F}|)]$, the function $\ell_{\alpha,i}(x) = \mathsf{bit}_i(\langle\alpha, x\rangle)$ that outputs the t^{th} bit of $\langle\alpha, x\rangle$ is a linear function over $\mathsf{GF}(2)$.

the (easy direction of) Yao's minimax principle, it suffices to show that there exists a distribution $\mathcal{X}$ over inputs in $\{0,1\}^k$ such that for every $\mathsf{DNF}_\oplus$ φ of size $\left(2^{O(c+q\cdot\log_2(|\mathbb{F}|))}\right)$ it holds that $\Pr_{x\in\mathcal{X}}\left[\varphi(x) = f_{\mathsf{MOD3}}(x)\right] < 0.9$ (where recall that $|\mathbb{F}| = \mathsf{polylog}(k)$). To that end, we shall use the celebrated result of Razborov [Raz87] and Smolensky [Smo87].

Theorem 4.3 (Razborov-Smolensky (see also [Vio09, Theorem 2]))**.** *Every* $\mathsf{AC}^0(\oplus)$ *circuit* φ *of size* s *and depth* d *satisfies*

$$\Pr_{x\in\{0,1\}^k}\left[\varphi(x) = f_{\mathsf{MOD3}}(x)\right] < \frac{2}{3} + O\left(\frac{\log(s)^d}{\sqrt{k}}\right).$$

This concludes the proof of Lemma 4.18. □

4.5 Implications for Classical Interactive Proofs

In this section, we derive from our hierarchy theorem implications to standard interactive proofs (in which the verifier can run in polynomial time). Loosely speaking, in Section 4.5.1 we show that the celebrated round reduction of public-coin interactive proofs, due to Babai and Moran [BM88], is (almost) optimal among all blackbox transformations, and in Section 4.5.2 we show that any proof that $\#\mathcal{P} \subseteq \mathsf{AM}$ will require using non-algebrizing techniques.

4.5.1 Blackbox Round Reduction Transformations

Babai and Moran [BM88] proved a "speedup" theorem, which loosely speaking, shows that very r-round public-coin interactive proof protocol can be transformed into an $(r-1)$-round protocol at the cost of increasing the communication complexity quadratically (some quantitative improvements were later obtained by Goldreich, Vadhan and Wigderson [GVW02]). Combined with the private-coin to public-coin transformation of Goldwasser and Sipser [GS86], one can obtain a similar "speedup" theorem for private-coin interactive proofs.

Vadhan [Vad00] considered the affect of certain transformations on interactive proofs. He introduced the notion of a "blackbox transformation" (defined below) and showed that the aforementioned private-coin to public-coin transformation, and a transformation from 2-sided error to 1-sided error of Goldreich, Mansour and Sipser [GMS87], are (in a certain sense) optimal amongst all *black-box* transformation.

In this section, we use our hierarchy theorem to derive a similar result for the round reduction theorem of Babai and Moran. Following [Vad00], we define a black-box transformation on interactive proofs as a procedure that takes as input an interactive proof $(\mathcal{P},\mathcal{V})$ for some language $\mathcal{L}$ and outputs a new interactive proof $(\mathcal{P}',\mathcal{V}')$, for the same language $\mathcal{L}$, such that:

- The strategy of the verifier $\mathcal{V}'$ can be implemented by an algorithm given oracle access to the strategy of $\mathcal{V}$.

- The strategy of the prover $\mathcal{P}'$ can be implemented by a algorithm given oracle access to the strategy of both $\mathcal{P}$ and $\mathcal{V}$.

Here, the strategy of a party (i.e., prover or verifier) is the function that takes the party's random coins and the history of messages exchanged and outputs its next message. We stress that the new strategies $(\mathcal{P}', \mathcal{V}')$ cannot even explicitly look at the input x; their only access to the input x is given by queries to the strategies $(\mathcal{P}, \mathcal{V})$.

An r-to-r' blackbox round reduction transformation, for $r' < r$, is a black-box transformation that, given as input an r-round interactive proof, produces an r'-interactive proof (for the same language). We remark that the [BM88] round-reduction is a blackbox round reduction transformation, and we show that it is nearly optimal, out of all blackbox reductions.

Theorem 4.4. *There exists a language $\mathcal{L}$ such that for every constant $r \geq 1$, there exists an r-round (public-coin) interactive proof $(\mathcal{P}, V)$ for $\mathcal{L}$, with communication complexity $c = c(n)$, such that for every r-to-r' blackbox round reduction transformation T, in the resulting interactive proof $(\mathcal{P}', \mathcal{V}') = T(\mathcal{P}, \mathcal{V})$ it holds that either the communication is at least $c^{\Omega(\sqrt{r}/r')}$ or $\mathcal{V}'$ invokes V at least $c^{\Omega(\sqrt{r}/r')}$ times.*

Proof. Let $r \in \mathbb{N}$ be a constant, and consider the language

$$\mathcal{L}_{\mathsf{MOD3}} = \left\{x \in \{0,1\}^k \ : \ \mathsf{wt}(x) = 0 \pmod 3\right\}_{k \in \mathbb{N}}.$$

Fix input length $k \in \mathbb{N}$, field $\mathbb{F}$, subset $H \subset \mathbb{F}$, and dimension $m = \frac{\log(k)}{\log\log(k)}$ such that $|H| = \log(k)$ and $|\mathbb{F}| = \Theta(|H|^2 m)$.

By Corollary 4.2, there exists an r-round HIP for $\mathcal{L}_{\mathsf{MOD3}}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},H,m}$, with communication complexity $c \stackrel{\text{def}}{=} k^{O(1/\sqrt{r})}$. As noted in Proposition 4.8, this HIP implies an interactive proof $(\mathcal{P}, \mathcal{V})$ for $\mathcal{L}_{\mathsf{MOD3}}$, with communication complexity c. Recall that on input $x \in \{0,1\}^k$, the parties $(\mathcal{P}, \mathcal{V})$ invoke the HIP for $\mathcal{L}_{\mathsf{MOD3}}$, and the verifier checks the HIP's output claim by computing a single point of $\mathsf{LDE}_{\mathbb{F},H,m}(x)$.

Let T be an r-to-r' blackbox round reduction transformation on interactive proofs, and let $(\mathcal{P}', \mathcal{V}') = T(\mathcal{P}, \mathcal{V})$ be the resulting r'-round interactive proof for $\mathcal{L}_{\mathsf{MOD3}}$. Using $(\mathcal{P}', \mathcal{V}')$, we construct an r'-round ε-IPP for the language

$$\mathsf{Enc\text{-}MOD3} = \left\{\mathsf{LDE}_{\mathbb{F},H,m}(x) \ : \ x \in \mathcal{L}_{\mathsf{MOD3}}\right\}.$$

Recall that $\mathcal{V}$ only computes $\mathsf{LDE}_{\mathbb{F},H,m}(x)$ and queries it at a single point, and so each oracle call to $\mathcal{V}$ that $\mathcal{V}'$ makes can be emulated by making a single query to $\mathsf{LDE}_{\mathbb{F},H,m}(x)$. Therefore, we can view $(\mathcal{P}', \mathcal{V}')$ as an HIP, with respect to $\mathsf{LDE}_{\mathbb{F},H,m}$, for $\mathcal{L}_{\mathsf{MOD3}}$, with communication complexity c.

By applying Lemma 4.10 on $(\mathcal{P}', \mathcal{V}')$, we obtain an r'-round IPP for Enc-MOD3; denote its communication complexity by C and query complexity by Q. Finally, by Lemma 4.18 we have that:

$$\max(C, Q) = k^{\Omega(1/r')} = c^{\Omega(\sqrt{r}/r')}.$$

□

4.5.2 The Algebrization Barrier

The *relatization* framework, introduced by Baker, Gill, and Solovay [BGS75], tried to capture the intuition that we not understand how circuits operate and therefore we may as well treat them as black-boxes. Later on, the seminal result of [LFKN92, Sha92] showed that even without understanding how circuits operate, we can still do more than just evaluate them (i.e., treat them as oracles). Specifically, arithmetizing the circuit, allows us to evaluate points in a *low degree extension* of the function computed by the circuit. The latter cannot be done only via oracle access and has turned out to be incredibly useful.

The *algebrization* framework, introduced by Aaronson and Wigderson [AW09], tries to capture this additional power. Specifically, in this framework, rather than just giving oracle access to the given function, we give oracle access also to a low degree extension of the function. Results such as $\mathsf{IP} = \mathsf{PSPACE}$ can be showed to have "algebrizing" proofs. Despite the power that we obtain by having access to the low degree extension of the function, [AW09] also showed that some central questions in complexity theory cannot be proved within this framework (i.e., by "algebrizing") techniques.

Loosely speaking, for two complexity classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the inclusion $\mathcal{C}_1 \subseteq \mathcal{C}_2$ is said to $\mathsf{algebrize}$ if $\mathcal{C}_1^A \subseteq \mathcal{C}_2^{\tilde{A}}$ for every oracle A and every low-degree extension $\tilde{A}$ of A. (See [AW09] for the precise definition, discussions and many more details.) We say that proving the inclusion $\mathcal{C}_1 \subseteq \mathcal{C}_2$ requires non-algebrizing techniques, or cannot be proved via algebrizing techniques, if the inclusion does *not* algebrize.

Before stating our results, we point out that there is an intimate connection (or a high level equivalence) between the class the algebrized class $\mathsf{IP}^{\tilde{A}}$ (where $\tilde{A}$ is the low degree extension of some oracle A) and the notion of HIPs (with respect to the low degree extension encoding). Indeed, in both cases the verifier needs to verify a property of some string, given oracle access to its low degree extension and interaction with the prover. For an $\mathsf{IP}^{\tilde{A}}$ the string is the truth table of A and for HIPs the string is simply the input.

Using this relation, we use our hierarchy theorem to show that the inclusion $\#\mathcal{P} \subseteq \mathsf{AM}$, which is widely believed *not* to hold[24], cannot be proved via algebrizing techniques. As a matter of fact, the proof of Theorem 4.5 can be easily extended to show that even the containment of $\#\mathcal{P}$ in a powerful variant of AM in which, for inputs of length N, the verifier is allowed to run in time $2^{o(N)}$ and with $2^{o(N)}$ communication, cannot be proved via algebrizing techniques.

Theorem 4.5. *There exists an oracle A and a low-degree extension $\tilde{A}$ of A such that $\#\mathcal{P}^A \not\subseteq \mathsf{AM}^{\tilde{A}}$.*

Proof Sketch. Consider the problem $\#\mathsf{CSAT}$, which is the problem of counting the number of satisfying assignments of a given (Boolean) circuit C, and recall that $\#\mathsf{CSAT}$ is $\#\mathcal{P}$-complete. Let $A : \{0,1\}^N \to \{0,1\}$ be an oracle and consider an input circuit C that, given as input $x \in \{0,1\}^N$, just outputs $A(x)$. We associate A with its truth table,

[24] In particular it implies the collapse of the polynomial hierarchy to its second level.

which is a string of length 2^N. Let $\tilde{A} = \mathsf{LDE}_{\mathbb{F},H,m}(A)$, where $\mathbb{F}, H, m$ are defined as in Section 4.4.1, with respect to the parameter $k = 2^N$.

Observe that if $\#\mathcal{P}^A \subset \mathsf{AM}^{\tilde{A}}$, then there exists an AM proof system for computing the number of satisfying assignments of the circuit C, which is exactly the Hamming weight of A (viewed as an k-bit string), in which the communication complexity is $\mathsf{poly}(N)$ and in which the verifier only makes $\mathsf{poly}(N)$ oracle queries to $\tilde{A}$. Thus, following Lemma 4.10, we can obtain from this AM proof system a 1-round IPP for $\mathsf{Enc\text{-}MOD3}$ with communication and query complexities $\mathsf{poly}(N) = \mathsf{polylog}(k)$, which violates the lower bound in Lemma 4.18. □

Remark 4.21 (Using Prime Order Fields)**.** *We remark that the proof of Theorem 4.5 is strongly based on the fact that we take a low degree extension over a field that has characteristic* 2*. Our result can extend to other constant size characteristics but we do not know how to extend it to arbitrary fields. In fact, it is consistent with our result (however unlikely) that there is a proof that* $\#\mathcal{P} \subseteq \mathsf{AM}$ *based (in a crucial way) on taking the low degree extension of the circuit with respect to a large prime order field.*

We remark that we are unaware of any complexity class containments in the literature that are only known based on algebrization using prime *order fields.*[25]

[25]The original proof of the $\mathsf{IP} = \mathsf{PSPACE}$ theorem by Shamir [Sha92] does use prime order fields in an important way, however, the more recent proof of the same result by Goldwasser *et al.* [GKR08] can be based on fields of arbitrary characteristic (see also [Mei13] that gives a proof based on general tensor codes).

4.6 Appendices for Chapter 4

4.6.1 Miscellaneous Discussions

4.6.1.1 Why [GKR08] and not other Interactive Proofs?

One may wonder whether we could base our upper bound on other interactive proofs from the literature. Other than the protocols of [GKR08, KR09], two other general purpose interactive proof-systems that come to mind are Shamir's[26][Sha92] protocol for $\mathsf{IP} = \mathsf{PSPACE}$ and a recent protocol of Reingold, Rothblum and Rothblum [RRR16] that gives constant-round interactive proofs for bounded-space computations.

Shamir's protocol is not suitable for our needs both because it is not constant-round, and, perhaps more fundamentally, because the verifier in Shamir's protocol needs to access the low-degree extension of the input over a field that is only determined during the interaction (recall that the verifier in Shamir's protocol chooses a *random* prime p, and the players both work over the field of integers modulo p). For our purposes the field has to be fixed a priori (since we want the input for the IPP to be encoded under the LDE code corresponding to that field).

As for the protocol of [RRR16], the latter does actually yield a constant-round HIP for $\mathcal{L}_{\mathsf{MOD3}}$ (which can be modified to yield an IPP for Enc-MOD3 as above) but the tradeoff that it offers between rounds and the verifier's complexity is exponentially worse than what we obtain. More specifically, for every constant $r \geq 1$, the [RRR16] protocol yields a $2^{\tilde{O}(r)}$-round HIP for $\mathcal{L}_{\mathsf{MOD3}}$ with verification time roughly $2^{\tilde{O}(r)} \cdot k^{1/r}$. In contrast, we obtain an $O(r^2)$-round HIP with verification time roughly $\mathsf{poly}(r) \cdot k^{1/r}$.

4.6.1.2 An Alternative Candidate Language for the Round Hierarchy Theorem

The language for which we proved our round hierarchy consists of encodings of strings whose Hamming weight is divisible by 3. As described next, it seems as though a similar result can be obtained for a related language Enc-Maj that consists of encodings of strings $x \in \{0,1\}^k$ with $\mathsf{wt}(x) \geq k/2$, although there are some technical difficulties to overcome.

First note that the lower bound for Enc-Maj follows along the same lines as our lower bound for Enc-MOD3, where now we use the fact that $\mathsf{AC}_0[2]$ circuits cannot approximate the majority function [Raz87, Smo87]. In contrast, showing an upper bound (i.e., an IPP or HIP) introduces some new difficulties. As explained in Section 7.1 (and formalized in Section 4.4), our upper bound for Enc-MOD3 is based on the observation that computing the sum, modulo 3, of the bits of an input string can be done by a (highly uniform) NC^1 circuit. Given this observation, we based our protocol on a variant of the [GKR08] interactive proof for small-depth computations.

For Enc-Maj, we could similarly hope to base our protocol on an NC^1 circuit, but this time we need a circuit that computes the majority function. Obtaining such a circuit is

[26] Indeed, here we specifically refer to Shamir's [Sha92] protocol and not to the [LFKN92] protocol (on which [Sha92] builds).

less trivial and here we encounter some difficulties:

- Valiant [Val84] (see the presentation of Goldreich [Gol11b]) gave a *non-uniform* construction of an NC^1 circuit for majority. We could base our protocol on this result and obtain a *non-uniform* verifier (and in particular, its running time would be super-linear, although it would still have the desired query and communication complexities).
- The aforementioned construction of [Val84] can actually be shown to produce a (highly-uniform) *randomized* construction. That is, there exists a randomized logspace Turing machine that given as input 1^n, with all but exponentially vanishing probability, produces an NC^1 circuit, on n-bit strings, that computes the majority function correctly (on all inputs). We could have our verifier run this procedure to obtain the desired NC^1 circuit, but this would introduce an (exponentially small) completeness error, which we would like to avoid.
- Lastly, we mention that the celebrated [AKS83] sorting network of Ajtai, Komlós and Szemerédi gives rise to a uniform (and deterministic) construction of an NC^1 circuit for majority (by sorting the input bits and outputting the median). This construction is quite complex and in particular we have not verifed whether it satisfies the uniformity condition that is required for the [KR09] result.[27]

4.6.2 From HIPs to IPPs (Proof of Lemma 4.10)

The two main ingredients that we shall use to prove Lemma 4.10 are the well-known low (individual) degree test[28] for multivariate polynomials [RS96, Sud95, AS03], and the self-correction procedure for polynomials [GS92, Sud95].

Lemma 4.22 (Individual Degree Test)**.** *Let $d, m \in \mathbb{N}$ such that $dm < |\mathbb{F}|/10$ and $\varepsilon \in (0, 1/10)$. Denote by $\mathsf{Poly}_{d,m,\mathbb{F}}$ the set of all m-variate, individual degree d polynomials over $\mathbb{F}$. Then, there exists an ε-tester for $\mathsf{Poly}_{d,m,\mathbb{F}}$ with query complexity $dm \cdot \mathsf{poly}(1/\varepsilon)$.*

Lemma 4.23. *Let $\varepsilon < 1/3$ and $d, m \in \mathbb{N}$ such that $d \leq |\mathbb{F}|$. There exists an algorithm (corrector) that, given $x \in \mathbb{F}^m$ and oracle access to an m-variate function $f : \mathbb{F}^m \to \mathbb{F}$ that is ε-close to a polynomial p of individual degree d, makes $O(d \cdot m)$ queries and outputs $p(x)$ with probability $9/10$. Furthermore, if f has total degree d, the algorithm outputs $p(x)$ with probability 1.*

Given Lemmas 4.22 to 4.23, we can now describe the IPP (with respect to some proximity parameter ε) for $\mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L})$. Recall that the verifier is given oracle access to

[27] We note that other *partial*, but arguably simpler, de-randomization results of Valiant's formula have been obtained by [HMP06] and [CDI+13]. However, these partial derandomizations do not seem to suffice for our purposes.

[28] Actually, the cited works provide a test for *total* degree. A test for *individual* degree (which is implicit in [GS06, Section 5.4.2]) can be obtained via a simple reduction (see, e.g., [GR13b, Theorem A.8]).

a function $f : \mathbb{F}^m \to \mathbb{F}$ and the prover is given direct access to f. Assume, without loss of generality, that the HIP for $\mathcal{L}$ has soundness error $1/10$.[29]

First, the verifier and prover run the HIP protocol for $\mathcal{L}$ with respect to the input $f|_{H^m}$. (Recall that an HIP does not even query its input and therefore, so far, no queries have been made.) If the HIP verifier rejects then we immediately reject. Otherwise, the verifier outputs a pair $(z, \nu) \in \mathbb{F}^m \times \mathbb{F}$ (with the associated claim that $f(z) = \nu$). Then, the verifier runs the individual degree tester of Lemma 4.22 on f, with respect to proximity parameter ε, individual degree $|H| - 1$ (and soundness error $1/3$). If the low degree test rejects, the verifier immediately rejects. Lastly, the verifier decodes f at point z, using the self-correction procedure of Lemma 4.23, again with soundness error $1/10$. The procedure outputs a value ν'. The verifier accepts if $\nu = \nu'$ and otherwise it rejects.

Completeness follows from the perfect completeness of the HIP, the low degree test and the local self-correction. For soundness, let $f : \mathbb{F}^m \to \mathbb{F}$ be a function such that f is ε-far from $\mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L})$ and fix a cheating prover strategy $\mathcal{P}^*$. Consider first the case that f is ε-far from an individual degree $|H| - 1$ polynomial. In this case, by the low degree test, with probability at least $2/3$, the verifier rejects and we are done. Thus, we can assume that f is ε-close to some individual degree $|H| - 1$ polynomial $P : \mathbb{F}^m \to \mathbb{F}$. Observe that since f is ε-far from $\mathsf{LDE}_{\mathbb{F},H,m}(\mathcal{L})$ it must be the case that $P|_{H^m} \notin \mathcal{L}$.

We view the HIP as being applied to $P|_{H^m}$. By the soundness of the HIP, when the verifier interacts with any cheating prover (and in particular $\mathcal{P}^*$) with probability $9/10$ it either rejects (in which case we also reject) or it outputs a pair $(z, \nu) \in \mathbb{F}^m \times \mathbb{F}$ such that $P(z) \neq \nu$. The verifier reads the point z with self-correction and so, with probability at least $9/10$ it will obtain the actual value $\nu' = P(z)$ and reject when comparing ν' and ν. Thus, with probability $0.9^2 \geq 2/3$ our verifier rejects.

4.6.3 The Sumcheck Protocol (Proof of Lemma 4.9)

In this appendix we prove Lemma 4.9.

We use a variant of the sumcheck protocol that takes r rounds, where for simplicity we assume that r divides m. We maintain the invariant that before the i^{th} rounds begins, both the verifier and the prover agree on values $w_1, \ldots, w_{i-1} \in \mathbb{F}^{m/r}$ and $\nu_{i-1} \in \mathbb{F}$, where $\nu_0 \stackrel{\text{def}}{=} 0$. For every $i \in [r]$, the i^{th} round of the sumcheck protocol is as follows.

1. The prover sends to the verifier the individual degree $|H|-1$ polynomial $P_i : \mathbb{F}^{m/r} \to \mathbb{F}$ (by specifying its coefficients), defined as:

$$P_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1},\ldots,x_r \in H^{m/r}} P(w_1, \ldots, w_{i-1}, z, x_{i+1}, \ldots, x_r).$$

2. The verifier receives a polynomial $Q_i : \mathbb{F}^{m/r} \to \mathbb{F}$ (which is allegedly equal to P_i) and checks that $\sum_{z \in H^{m/r}} Q_i(z) = \nu_{i-1}$.

[29] Indeed, parallel repetition of IPPs decreases their soundness error at an exponential rate (see [GGR15, Appendix A] for details).

3. The verifier select uniformly at random $w_i \in \mathbb{F}^{m/r}$ and sends w_i to the prover.

4. Set $\nu_i \stackrel{\text{def}}{=} Q_i(w_i)$.

At the end of the protocol, the verifier outputs $((w_1, \ldots, w_r), \nu_r) \in \mathbb{F}^m \times \mathbb{F}$.

The running times and communication complexity of the protocol can be readily verified. We proceed to show that completeness and soundness hold.

Completeness. Let $P : \mathbb{F}^m \to \mathbb{F}$ be an individual degree $|H| - 1$ polynomial such that $\sum_{x \in H^m} P(x) = 0$. In this case, at every round $i \in [\rho]$, the prover sends the polynomial $Q_i \equiv P_i$. Hence, for every $i \in [r]$:

$$\begin{aligned}
\sum_{z \in H^{m/r}} Q_i(z) &= \sum_{z \in H^{m/r}} P_i(z) \\
&= \sum_{z \in H^{m/r}} \sum_{x_{i+1}, \ldots, x_r \in H^{m/r}} P(w_1, \ldots, w_{i-1}, z, x_{i+1}, \ldots, x_r) \\
&= P_{i-1}(w_{i-1}) \\
&= Q_{i-1}(w_{i-1}) \\
&= \nu_{i-1}
\end{aligned}$$

and so all of the verifier's checks pass. At the end of the protocol the verifier outputs $((w_1, \ldots, w_r), \nu_r) \in \mathbb{F}^m \times \mathbb{F}$ and $\nu_r = P_r(w_r) = P(w_1, \ldots, w_r))$ as required.

Soundness. Let $P : \mathbb{F}^m \to \mathbb{F}$ be an individual degree $|H| - 1$ polynomial such that $\sum_{x \in H^m} P(x) \neq 0$ and fix a cheating prover strategy $\mathcal{P}^*$.

The next two claims relate the polynomials Q_i sent by the prover to the corresponding polynomials P_i (recall that P_i was defined as $P_i(z) = \sum_{x_{i+1}, \ldots, x_r \in H^{m/r}} P(w_1, \ldots, w_{i-1}, z, x_{i+1}, \ldots, x_r)$). Recall that both polynomials depend only on $w_1, \ldots, w_{i-1}$.

Claim 4.23.1. *If $Q_1 \equiv P_1$, then the verifier rejects with probability 1.*

Proof. Observe that $\sum_{x_1 \in H^{m/r}} P_1(x_1) = \sum_{z \in H^m} P(z) \neq 0$, and so, if $Q_1 \equiv P_1$, then the verifier rejects when testing that $\sum_{z \in H^{m/r}} Q_1(z) = \nu_0 = 0$. □

Claim 4.23.2. *For every $i \in [r-1]$ and every $w_1, \ldots, w_{i-1} \in \mathbb{F}^{m/r}$, if $Q_i \not\equiv P_i$ then, with probability $1 - \frac{(m/r) \cdot |H|}{|\mathbb{F}|}$ over the choice of w_i, if $Q_{i+1} \equiv P_{i+1}$ then the verifier rejects.*

Proof. Since the (total degree $(m/r) \cdot (|H| - 1)$) polynomials Q_i and P_i differ, by the Shwartz-Zippel lemma (Lemma 4.3), with probability $1 - \frac{(m/r) \cdot |H|}{|\mathbb{F}|}$ over the choice of $w_i \in_R \mathbb{F}^{m/r}$, it holds that $Q_i(w_i) \neq P_i(w_i)$. If the latter event occurs and the prover sends $Q_{i+1} \equiv P_{i+1}$, then the verifier rejects when testing whether $\sum_{z \in H^{m/r}} Q_{i+1}(z) = \nu_i$, since

$$\nu_i = Q_i(w_i) \neq P_i(w_i) = \sum_{z \in H^{m/r}} P_{i+1}(z) = \sum_{z \in H^{m/r}} Q_{i+1}(z).$$

□

By Claims 4.23.1 and Claim 4.23.2 and an application of the union bound, with probability $1-(r-1)\cdot\frac{(m/r)\cdot|H}{|\mathbb{F}|}$, if there exists an $i\in[r-1]$ such that $Q_i\not\equiv P_i$ but $Q_{i+1}\equiv P_{i+1}$ then the verifier rejects. However, by Claim 4.23.1, we can assume that $Q_1\not\equiv P_1$ and so we get that with probability $1-(r-1)\cdot\frac{(m/r)\cdot|H}{|\mathbb{F}|}$ either the verifier rejects or $Q_r\not\equiv P_r$. Note that if $Q_r\not\equiv P_r$ then by the Shwartz Zippel Lemma with probability $1-\frac{(m/r)\cdot|H|}{|\mathbb{F}|}$ it holds that $Q_r(w_r)\neq P_r(w_r)$ and therefore:

$$\nu_r=Q_r(w_r)\neq P_r(w_r)=P(w_1,\ldots,w_r)$$

and so the soundness condition holds, with soundness error $(r-1)\cdot\frac{(m/r)\cdot|H}{|\mathbb{F}|}+\frac{(m/r)\cdot|H}{|\mathbb{F}|}=\frac{m\cdot|H}{|\mathbb{F}|}$.

4.6.4 Interactive Proof for Vanishing-Subcube (Proof of Proposition 4.13)

Let $\mathbb{F}$ be a constructible field ensemble, let $H\subseteq\mathbb{G}\subseteq\mathbb{F}$ be ensembles of subsets, and let $m\in\mathbb{N}$. Recall that $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,G}$ is the set of all functions $f:G^m\to\mathbb{F}$ that vanish on H^m (i.e., $f|_{H^m}\equiv 0$). We show that for every $r\in[m]$, there exists an $r+2$-round (public-coin) HIP for $\mathsf{Vanishing\text{-}Subcube}_{\mathbb{F},H,m,\mathbb{G}}$, with respect to the code $\mathsf{LDE}_{\mathbb{F},\mathbb{G},m}$.

Recall that in an HIP with respect to the code $\mathsf{LDE}_{\mathbb{F},\mathbb{G},m}$, the input should be thought of as an m-variate polynomial P with individual degree $|\mathbb{G}|-1$. The prover has direct access to P and the verifier needs to output a pair $(z,\nu)\in\mathbb{F}^m\times\mathbb{F}$, with the associated claim that $P(z)=\nu$.

For a given function $P:\mathbb{F}^m\to\mathbb{F}$, we define the polynomial $\tilde{P}(x)=\sum_{z\in H^m}\delta(z,x)\cdot P(z)$, where $\delta:\mathbb{F}^m\times\mathbb{F}^m\to\mathbb{F}$ is an individual degree $|H|-1$ polynomial such that for every $a,b\in H^m$, it holds that $\delta(a,b)=1$ if $a=b$ and $\delta(a,b)=0$ otherwise (and δ is arbitrary in $\mathbb{F}^{2m}\backslash H^{2m}$).[30]

To check that P is identically 0 in H^m, the verifier first chooses at random $r\in\mathbb{F}^m$ and sends r to the prover. Now, the prover and verifier run an interactive proof to check that $\tilde{P}(r)=0$, by invoking the sumcheck protocol with respect to the summation $\sum_{z\in H^m}\delta(z,r)\cdot P(z)=0$, where we observe that the polynomial $\delta(\cdot,r)\cdot P(\cdot)$ has individual degree $|H|+|\mathbb{G}|-1$. If the sumcheck verifier rejects, then we immediately reject. Otherwise, the sumcheck verifier outputs a pair $(z,\nu)\in\mathbb{F}^m\times\mathbb{F}$, and the prover then sends the value $\nu'=P(z)$. Finally, the verifier checks that $\delta(z,r)\cdot\nu'=\nu$ and if so outputs (z,ν').

For completeness, note that if P is identically 0 in H^m, then $\tilde{P}$ is identically 0 in $\mathbb{F}^m$. In particular, with probability 1 over the choice of r it holds that $\tilde{P}(r)=\sum_{z\in H^m}\delta(z,r)\cdot P(h)=0$. Thus, by the completeness of the sumcheck protocol, the sumcheck verifier outputs a pair (z,ν) such that $\delta(z,r)\cdot P(z)=0$. The prover now sends the value $\nu'=P(z)$, and so the verifier's check that $\delta(z,r)\cdot\nu'=\nu$ passes, and it outputs the claim (z,ν'), which is correct since $P(z)=\nu'$.

[30] We note that $\tilde{P}$ is in fact the low degree extension of the function P, when the latter is restricted to H^m.

As for soundness, if P is not identically 0 in H^m, then by definition, $\tilde{P}$ is not identically 0 in $\mathbb{F}^m$, and therefore by the Schwartz-Zippel lemma (see Lemma 4.3), with probability $1 - \frac{m\cdot(|H|-1)}{|\mathbb{F}|}$ over the choice of r, it holds that $\tilde{P}(r) \neq 0$. Thus, the sumcheck protocol is invoked on the sum $\sum_{z\in H^m} \delta(z,r)\cdot P(z) \neq 0$ and so, with probability $1 - \frac{m\cdot(|H|+|\mathbb{G}|-2)}{|\mathbb{F}|}$ either the sumcheck verifier rejects, or it outputs a claim (z,ν) such that $\delta(z,r)\cdot P(z) \neq \nu$. Assuming the latter happens, if the prover now sends $\nu' = P(z)$, then the verifier rejects. Hence, it must send $\nu' \neq P(z)$, and so the verifier outputs the incorrect claim (z,ν').

4.6.5 Efficiently Computing $\widetilde{\mathsf{MOD3}}_t$

Recall that $\widetilde{\mathsf{MOD3}}_t : \mathbb{K}^t \to \mathbb{K}$ was defined as the (unique) individual degree 2 polynomial such that for every $h \in \{0,1,2\}^t$ it holds that $\widetilde{\mathsf{MOD3}}_t(h) = \sum_{i\in[t]} h_i \pmod 3$. In this section we show that $\widetilde{\mathsf{MOD3}}$ is efficiently computable. Namely, that given a point $z \in \mathbb{K}^t$, one can compute $\widetilde{\mathsf{MOD3}}_t(z)$ in time $\mathsf{poly}(t, \log(|\mathbb{K}|))$.

Proposition 4.24. *Let $\mathbb{K}$ be a constructible field ensemble. There exists a $\mathsf{poly}(t,\log(|\mathbb{K}|))$-time algorithm that given a point $z \in \mathbb{K}^t$ outputs the value $\widetilde{\mathsf{MOD3}}_t(z)$.*

Proof. To prove Proposition 4.24, we first show that for every $\sigma \in \{0,1,2\}$ and $i \in [t]$, we can construct a size $\mathsf{poly}(i)$ uniform arithmetic circuit over $\mathbb{K}$ that computes the function $F_i^{(\sigma)} : \mathbb{K}^i \to \mathbb{K}$, which is defined as the unique individual degree 2 polynomial such that:

$$\forall h \in \{0,1,2\}^i, \quad F_i^{(\sigma)}(h) = \begin{cases} 1 & \text{if } \sum_{i\in[t]} h_i = \sigma \pmod 3 \\ 0 & \text{otherwise} \end{cases}.$$

where the summation is over integers modulo 3. Despite their similarity, note that $\widetilde{\mathsf{MOD3}}_t$ is the low degree extension of a function that *computes* the sum modulo 3 of its input, whereas $F_t^{(\sigma)}$ is the low degree extension of a function that *indicates* whether the sum modulo 3 is congruent to σ.

Given arithmetic circuits that compute $\mathbb{F}_t^{(\sigma)}$, we can now compute $\widetilde{\mathsf{MOD3}}_t : \mathbb{K}^t \to \mathbb{K}$ as:

$$\widetilde{\mathsf{MOD3}}_t(z) = \sum_{\sigma\in\{0,1,2\}} \sigma \cdot F_i^{(\sigma)}(z), \tag{4.10}$$

where here the arithmetic is over the field $\mathbb{K}$, and the equality follows from the fact that both sides of the equation are polynomials of individual degree 2 that agree on $\{0,1,2\}^t$ and therefore must agree on $\mathbb{K}^t$. Thus, it remains to prove the following claim.

Claim 4.24.1. *For every $\sigma \in \{0,1,2\}$ and $i \in \mathbb{N}$, there exists an arithmetic circuit of size $O\left(i^{\log_2(6)}\right)$ over $\mathbb{K}$ that computes $F_i^{(\sigma)}$.*

Proof. We prove the proposition for i's that are powers of two and note that the general case follows easily (e.g., by using a circuit of size that is the nearest power of two and fixing some of its inputs to 0).

The proof is by induction on i, where the base case $i = 1$, is trivial. Fix i (that is a power of two) and suppose that we have constructed arithmetic circuits for computing $F_i^{(\sigma)}$ for every $\sigma \in \{0, 1, 2\}$.

Fix $\tau \in \{0, 1, 2\}$. The main observation is that for every $z_1, z_2 \in \mathbb{K}^i$ it holds that

$$F_{2i}^{(\tau)}(z_1, z_2) = \sum_{\sigma \in \{0,1,2\}} F_i^{(\sigma)}(z_1) \cdot F_i^{(\tau - \sigma \bmod 3)}(z_2), \tag{4.11}$$

where the equality follows from the fact that that both sides of the equation are polynomials of individual degree 2 that agree on $\{0, 1, 2\}^i$ and therefore must agree on $\mathbb{K}^{2i}$.

Denoting by S_i the size of the arithmetic circuit that Eq. (4.11) yields for $F_i^{(\sigma)}$, it holds that:

$$S_{2i} = 6 \cdot S_i + c = \cdots = 6^{\log(2i)} \cdot S_1 + c \cdot \sum_{j=0}^{i-1} 6^j = O\left((2i)^{\log_2(6)}\right),$$

where $c \leq 10$ is the constant overhead that arises from Eq. (4.11). This concludes the proof of Claim 4.24.1. □

Proposition 4.24 now follows by combining Eq. (4.10) and Claim 4.24.1. □

Chapter 5

Strong Locally Testable Codes with Relaxed Local Decoders

5.1 Introduction

Locally testable codes (LTCs) are error-correcting codes that can be tested very efficiently. Specifically, a code is said to be an LTC if there exists a probabilistic algorithm, called a tester, that is given a *proximity parameter* $\varepsilon > 0$ and oracle access to an input string (an alleged codeword), makes a small number (e.g., $\mathsf{poly}(1/\varepsilon)$) of queries to the input and is required to accept valid codewords, and reject with high probability input strings that are ε-far from being a codeword (i.e., reject strings that disagree with any codeword on ε fraction of the bits). The systematic study of LTCs was initiated by Goldreich and Sudan [GS06], though the notion was mentioned, in passing, a few years earlier by Friedl and Sudan [FS95] and Rubinfeld and Sudan [RS96].

A natural strengthening of the notion of locally testable codes (LTCs) is known as strong-LTCs. While LTCs (also referred to as weak-LTCs) allow for a different behavior of the tester for different values of the proximity parameter, strong-LTCs are required to satisfy a strong *uniformity* condition over all values of the proximity parameter. In more detail, the tester of a strong-LTC does *not* get a proximity parameter as an input, and is instead required to make only a *constant* number of queries and reject non-codewords with probability that is related to their distance from the code. See [GS06, Gol10c] for a discussion on both types of local testability. We note that from a property testing point of view, strong-LTCs can be thought of as codes that can be tested by a *proximity-oblivious tester* (see [GR09]).

The two most fundamental parameters of error-correcting codes (and strong-LTCs in particular) are the *distance* and the codeword *length*. Throughout this chapter we will only consider codes with constant relative distance, and so our main parameter of interest is the length, which measures the amount of redundancy of information in each codeword. By this criterion, constructing a strong-LTC with linear length (and constant relative distance) is the holy grail of designing efficient locally testable codes. Although recently some progress was made towards showing the impossibility of such linear length

LTCs [DK11, BV12], there are known constructions of strong-LTCs with relatively good parameters: Goldreich and Sudan [GS06] constructed a strong-LTC with constant relative distance and nearly-linear length, where throughout this chapter a code of dimension k is said to have nearly-linear length if its codewords are of length $k^{1+\alpha}$ for an arbitrarily small constant $\alpha > 0$. Furthermore, recently Viderman [Vid13] constructed a strong-LTC with constant relative distance and quasilinear length (i.e., length $k \cdot \mathsf{polylog} k$).

Another natural local property of codes is *local decodability*. A code is said to be a locally decodable code (LDC) if it allows for a highly efficient recovery of any individual bit of the message encoded in a *somewhat corrupted* codeword. That is, there exists a probabilistic algorithm, called a decoder, that is given a location i and oracle access to an input string w that is promised to be sufficiently close to a codeword. The decoder is allowed to make a small (usually constant) number of queries to the input w and is required to decode the i^{th} bit of the information that corresponds to the codeword that w is closest to. Following the work of Katz and Trevisan [KT00] that formally defined the notion of LDCs, these codes received much attention and found numerous applications (see e.g., [Tre04, Yek12] and references therein). They are also related to *private information retrieval* protocols [CGKS98] (see [Gas04] for a survey).

Despite much attention that LDCs received in recent years, the best known LDCs are of super-polynomial length (cf. [Efr12], building on [Yek08]). While the best known lower bound (cf. [KT00]) only shows that any q-query LDC must be of length $\Omega\left(k^{1+\frac{1}{q-1}}\right)$ (where k is the dimension of the code), the existence of a constant-query LDC with *polynomial* length remains a major open problem.

In an attempt to bypass this barrier, Ben-Sasson*et al.*[BSGH+06] introduced a natural relaxation of the notion of local decodability, known as relaxed-LDCs. This relaxation requires local recovery of most (or nearly all) individual information-bits, yet allows for recovery-failure (but not error) on the rest. Specifically, a code is said to be a relaxed-LDC if there exists an algorithm, called a (relaxed) decoder, that has oracle access to an input string that is promised to be sufficiently close to a codeword. Similarly to LDCs, the decoder is allowed to make few queries to the input in attempt to decode a given location in the message. However, unlike LDCs, the relaxed decoder is allowed to output an abort symbol on a small fraction of the locations, which indicates that the decoder detected a corruption in the codeword and is unable to decode this specific information-bit. Note that the decoder must still avoid errors (with high probability).

Throughout this chapter, unless explicitly stated otherwise, when we say that a code is a relaxed-LDC, we actually mean that it is a relaxed-LDC with *constant* query complexity.

Ben-Sasson*et al.*[BSGH+06] constructed a relaxed-LDC with nearly-linear length. More generally, they showed that for every constant $\alpha > 0$ there exists a relaxed-LDC (with constant relative distance) that maps k-bit messages to $k^{1+\alpha}$-bit codewords and has query complexity $O\left(1/\alpha^2\right)$. While these relaxed-LDCs are dramatically shorter than any known LDC, they do not break the currently known lower bound on LDCs (cf. [KT00]), and hence it is it still an open question whether relaxed-LDC are a strict relaxation of LDCs.

5.1.1 Obtaining Local Testability and Decodability Simultaneously

In this chapter, we are interested in short codes that are both (strongly) locally testable and (relaxed) locally decodable.[1] The motivation behind such codes is very natural, as the notion of local decodability is complimentary to the notion of local testability: The success of the decoding procedure of a locally decodable code is pending on the promise that the input is sufficiently close to a valid codeword. If the locally decodable code is also locally testable, then this promise can be verified by the testing procedure. However, recall that there are no known constant-query LDCs with even polynomial length, let alone such that are also locally testable. Hence, we focus on relaxed-LDCs.[2]

There are a couple of known constructions of codes that are both locally testable and relaxed decodable (with constant query complexity). Ben-Sasson *et al.*[BSGH+06] observed that their relaxed-LDC can be modified to also be a weak-LTC (i.e., an LTC that is not strong), while keeping its length nearly-linear. However, the local testability of their code is inherently *weak* (see Section 5.1.3 for details). In a recent development, Gur and Rothblum [GR13c] constructed a relaxed-LDC that is also a strong-LTC, albeit with *polynomial length*.

In this chapter, we improve upon the aforementioned results of [BSGH+06] and [GR13c], achieving the best of both worlds. That is, we construct a code that is both a strong-LTC and a relaxed-LDC with *nearly-linear* length.

Theorem 5.1 (informal)**.** *There exists a binary linear code that is a relaxed-*LDC *and a (one-sided error)* strong-LTC *with constant relative distance and nearly-linear length.*

A formal statement of Theorem 5.1 is given in Section 5.3. We remark that we actually prove a slightly stronger claim; namely, that any good linear code can be augmented (by appending additional bits to each codeword) into a code that is both a relaxed-LDC and a strong-LTC, at the cost of increasing the codeword length from linear to nearly-linear.

On Invoking Testers Prior to Decoders. Recall that for a code that is both locally testable and decodable, the promise (that the input is close to a codeword) required by the decoder can be eliminated by invoking the tester first. However, doing so can potentially hamper the decodability, since the tester is allowed to reject codewords that are only slightly corrupted. Fortunately, our tester is smooth (i.e., it queries each of the n bits of a codeword with probability $\Theta(1/n)$), and thus invoking the strong-tester a carefully chosen number of times (rejecting if one of the invocations rejected) will result in a tolerant tester (see [GR05, PRR06]). Such a tester will reject inputs that do not

[1] Note that although the notion of local testability and decodability are related, LTCs do not imply LDCs (i.e., there are LTCs that are not LDCs) and vice-versa. (See [KV10].)

[2] A different possible approach to solve this problem is to settle for codes with *long length*. Indeed, there are codes with *exponential* length that are both (constant-query) LDCs and LTCs, e.g., the Hadamard code. Another approach to solve this problem is to settle for codes with *large query complexity*. In a recent work, Guo, Kopparty, and Sudan [GKS13] constructed very short length codes that are both locally testable and locally decodable, albeit with large (yet needless to say, sub-linear) query complexity.

satisfy the promise of the decoder, yet still accept slightly-corrupted codewords (with high probability).

5.1.2 Strong Canonical PCPs of Proximity

The notion of PCPs of proximity plays a major role in many constructions of LTCs and relaxed-LDCs, as well as in our own. Loosely speaking, PCPs of proximity (PCPPs) are a variant of PCP proof systems, which can be thought of as the PCP analogue of *property testing*. Recall that a standard PCP is given explicit access to a statement (i.e., an input that is supposedly in some NP language) and oracle access to a proof (i.e., a "probabilistically checkable" NP witness). The PCP verifier is required to probabilistically verify whether the (explicitly given) statement is correct, by making few queries to the alleged proof. In contrast, a PCPP is given oracle access to a statement and to a proof, and is only allowed to make a small number of queries to both the statement and the proof. Since a PCPP verifier only sees a small part of the statement (typically, only a constant number of bits), it cannot be expected to verify the statement precisely. Instead, it is required only to accept correct statements and reject statements that are far from being correct (i.e., far in Hamming distance from any valid statement).

PCPs of proximity were first studied by Ben-Sasson*et al.*[BSGH+06] and by Dinur and Reingold [DR06] (wherein they are called assignment testers). The main parameters of interest in a PCPP system for some language L are its *query complexity* (i.e., the total number of queries to the input and to the proof that the PCPP verifier makes in order to determine membership in L) and its proof *length*, which can be thought as measuring the amount of redundancy of information in the proof. Ben-Sasson*et al.*[BSGH+06] showed a PCPP for any language in NP, with constant query complexity and nearly-linear length (in fact, the length is $n^{1+o(1)}$, where n is the length of the corresponding NP-witness).

As we have already noted, PCPPs have a central theoretical significance as the property testing analogue of PCP proof-systems. Moreover, PCPPs were shown to be useful in various applications, e.g., for PCP composition and alphabet reduction [BSGH+06, DR06], and for locally testable and locally decodable codes [BSGH+06, GS06, GR13c]. Further information regarding the latter application follows.

The notion of locally testable codes and PCPs of proximity are tightly connected. Not only that PCPPs (and PCPs in general) can be thought of as the computational analogue of the (combinatorial) notion of LTCs, but also any code can be made locally testable by using an adequate PCPP. Specifically, Ben-Sasson*et al.*[BSGH+06] showed that any linear code can be transformed to a (weak) LTC by appending each codeword with a PCPP proof that ascertains that the codeword is indeed properly encoded.[3] However, since there is no guarantee that every two different proofs for the same statement are far (in Hamming distance), in order to prevent deterioration of the distance of the code two additional steps are taken: Firstly, the appended PCPP proof should be uniquely determined per codeword (i.e., each codeword has a *canonical* proof), and secondly, each

[3]Note that membership in any linear code is in P, and so, the efficient PCPP for NP of Ben-Sasson*et al.*[BSGH+06] can handle these statements.

codeword is repeated many times so that the PCPP part constitutes only a small fraction of the total length.

The drawback of the foregoing approach is that it results in locally testable codes that are *inherently* weak (i.e., codes that do not allow for proximity-oblivious testing). To see this, note that PCPPs only guarantee that false assertions are rejected (with high probability), while true assertions can be accepted even if the proof is incorrect. Hence, corruptions in the PCPP part are not necessarily detectable and the *canonicity* of the PCPP proofs may not be verified, ruling out the possibility of a (strong) tester that is uniform over all possible values of the proximity parameter.[4] Moreover, when trying to build strong-LTCs, an additional problem that arises is that, by definition, PCPPs do not necessarily provide strong soundness, i.e., reject false proofs with probability that depends only on their distance from a correct proof.

Motivated by constructing *strong* locally testable codes, Goldreich and Sudan [GS06, Section 5.3] considered a natural strengthening of the notion of PCPPs, known as strong canonical PCPs of proximity (hereafter scPCPP), which addresses the aforementioned issues. Loosely speaking, scPCPP are PCPPs with *strong soundness* that are required to reject "wrong" proofs, even for correct statements. Moreover, they require that each correct statement will only have one acceptable proof. In more detail, scPCPP are PCPP with two additional requirements: (1) *canonicity*: for every true statement there exists a unique proof (called the canonical proof) that the verifier is required to accept, and any other proof (even for a correct statement) must be rejected, and (2) *strong soundness*: the scPCPP verifier is required to be proximity oblivious and reject any pair of statement and proof with probability that is related to its distance from a true statement and its corresponding canonical proof. A formal definition of scPCPPs can be found in Section 5.2.4.

Given a construction of adequate scPCPPs, the aforementioned strategy of appending each codeword with an efficient scPCPP (which ascertains membership in a code) will allow to transform any code to a strong-LTC. Unfortunately, unlike standard PCPPs, for which there are efficient constructions for any language in NP, there are no known constructions of general-purpose scPCPPs. Yet, Goldreich and Sudan constructed a mechanism, called *linear inner proof systems* (LIPS), which is closely related to some special-purpose scPCPPs. Loosely speaking, the LIPS mechanism allows to transform linear strong locally testable codes over a large alphabet into strong locally testable codes over a smaller alphabet (see [GS06, Section 5.2] for further details). By a highly non-trivial construction and usage of the LIPS mechanism, Goldreich and Sudan showed efficient constructions of strong-LTCs. Unfourtunately, their constructions do not meet our needs. Nevertheless, building upon their techniques, we show *strong canonical* PCPs *of proximity*

[4] In contrast, note that for *weak* LTCs this problem can be ignored by simply making the PCPPs themselves a sufficiently small part of the codewords. Recall that *weak* LTCs are allowed to only work for values of the proximity parameter that are sufficiently large to ensure that the concatenation of a corrupted codeword and its corresponding PCPP sequence will include (significant) corruption in the codeword part. Thus, there is no need to verify the canonicity or even validity of the PCPP proof. However, when we seek to achieve the stronger definition of LTCs (i.e., strong-LTCs), this problem becomes relevant (and cannot be ignored).

with polynomial length for any good linear code.

Theorem 5.2 (scPCPP for good codes — informal)**.** *Let C be a linear code with constant relative distance and linear length. Then, there exists a* scPCPP *with polynomial proof length for membership in the set of all codewords of C.*

In fact, we actually prove a slightly stronger statement. Specifically, our scPCPPs satisfy two additional properties that will be useful for our main construction: The scPCPP proofs are linear (over GF(2)), and the queries that the verifier makes are roughly uniform. We remark that not only that the scPCPPs in Theorem 5.2 are crucial to our construction (see Section 5.1.4 for details), we also view these scPCPPs as interesting on their own. A formal statement of Theorem 5.2 and its proof are presented in Section 5.6.

5.1.3 Previous Works and Techniques

In this subsection, we survey the previous works and techniques regarding relaxed-LDCs upon which we build. We start by recalling the construction of the (nearly) quadratic length relaxed-LDC of Ben-Sasson*et al.*[BSGH+06, Section 4.2]. The core idea that underlies their construction is to partition each codeword into three parts: The first providing the distance property, the second allowing for "local decodability", and the third ascertaining the consistency of the first two parts. The natural decoder for such a code will verify the consistency of the first two parts via the third part and decode according to the second part in case it detects no consistency error. Details follow.

Let C be any good linear code (i.e., a code with constant relative distance and linear length). Ben-Sasson*et al.*construct a new code C' whose codewords consist of three parts of equal length: (1) repetitions of a good codeword $C(x)$ that encodes the message x; (2) repetitions of the explicitly given message x; and (3) PCPPs that ascertain the consistency of each individual bit in the message x (which is explicitly given in the second part) with the codeword $C(x)$ (which is explicitly given in the first part). We remark that since the total length of the PCPPs is significantly longer than the statements they ascertain, the desired length proportions are obtained by the repetitions in the first two parts. Observe that the first part grants the new code C' good distance (although it may *not* be locally decodable), the second part allows for a highly efficient decoding of the message (at the cost of reducing the distance), and the third part is needed in order to guarantee that the first two parts refer to the same message. The (relaxed) decoder for C' will use the PCPPs in the third part in order to verify that the first part (the codeword $C(x)$) is consistent with the bit we wish to decode in the second part (the message x). If the PCPP verifier detects no error, the decoder returns the relevant bit in the second part; otherwise, it returns an abort symbol.

In order to implement the aforementioned relaxed-LDC, an adequate PCPP is needed; that is, an efficient PCPP for verifying the consistency of each individual bit in a message x with the codeword $C(x)$. We note that such statements are in P. Recall that Ben-Sasson*et al.*[BSGH+06, Section 3] showed PCPPs with nearly-linear length for *any* language in NP. Hence, the consistency of each message bit with a codeword of C can

be guarantied by a PCPP of length that is nearly-linear in the length of C. Since C' is obtained by augmenting a good linear code C with a single PCPP proof per every message bit (claiming consistency between that bit and the codeword of C), the length of C' is (nearly) quadratic (i.e., length $k^{2+\alpha}$ for an arbitrarily small constant $\alpha > 0$, where k is the dimension of the code). We note that Ben-Sasson*et al.*showed that the length of C' can be improved to nearly-linear by, roughly speaking, partitioning the message into blocks of various lengths and decoding based on a chain of consistent blocks.[5]

Recall that *any* code can be transformed to a weak locally testable code by appending adequate PCPPs to it (See [BSGH+06, Section 4.1]). Applying this transformation to the relaxed-LDC does not hamper the relaxed decodability of the code, and only increase its length by a moderate amount (since the PCPPs are of nearly-linear length); hence this transformation yields a (constant query) relaxed-LDC with nearly-linear length that is also a (weak) LTC. We stress that the aforementioned transformation yields local testability that is *inherently weak* due to the fact that it uses standard PCPPs. However, if the PCPPs in use were actually scPCPPs (of nearly-linear length), then the foregoing code would have been strongly testable.

In a recent work, Gur and Rothblum [GR13c] constructed scPCPPs with polynomial length for the particular family of linear length statements that are needed for the [BSGH+06] relaxed-LDC. By using these scPCPPs in the construction of [BSGH+06], they obtained a relaxed-LDC that is also a strong-LTC, albeit with polynomial length (due to the length of their scPCPPs). While we conjecture it is feasible to construct nearly-linear length scPCPPs for P (which contains the family of statements we wish to have scPCPPs for) and even for unique-NP (also known as the class US),[6] we do not obtain such scPCPPs here. Instead, we take an alternative approach, which circumvents this challenge, as described in the next subsection.

5.1.4 Our Techniques

In this subsection, we present our main techniques and ideas for constructing a relaxed-LDC with nearly-linear length that is also a strong-LTC. Our starting point is the (*weakly* testable) relaxed-LDC construction of Ben-Sasson*et al.*[BSGH+06]. However, we wish to replace the PCPPs that they use with scPCPPs, in order to achieve *strong* local testability.

Since we do not have general-purpose scPCPPs (let alone of near-linear length), we

[5]To obtain length $k^{1.5+\alpha}$, the message is partitioned into $\sqrt{k}$ blocks, each of length $\sqrt{k}$. Then, the original message, as well as each of the smaller blocks is encoded by an error-correcting code. For each of the encoded smaller blocks, the following PCPPs are added: (1) a PCPP that ascertains the consistency of the encoded block with the encoded original-message; and (2) PCPPs that ascertains the consistency of each bit in the encoded block with the entire encoded block. Observe that the total encoding length decreased, since there are $\sqrt{k}$ proofs of statements of length $O(k)$ and k proofs of statements of length $O(\sqrt{k})$, thus, the total length is nearly-linear in $k^{3/2}$. By repeating this process, we can reduce the length of the code to $k^{1+\alpha}$ for an arbitrarily small constant $\alpha > 0$ (see [BSGH+06, Section 4.2] for details).

[6]We note that the class unique-NP(i.e., the class of NP problems with unique witnesses) seems more likely to have scPCPPs than NP. This is because a language in NP may have many witnesses per instance, and it is not clear how to recognize the "canonical" NP-witness.

construct special-purpose scPCPPs that allow us to ascertain the particular statements we are interested in (see Theorem 5.2). It is crucial to note that the scPCPPs we are able to construct are with *polynomial* proof length (and not nearly-linear length, as we would have hoped). Recall that the statements that are needed for the construction of Ben-Sasson*et al.*(i.e., ascertaining the consistency of each bit of the message with the entire codeword for *decodability*, and ascertaining the validity of the codeword for *testability*) are linear in the length of the message. Therefore, applying our scPCPPs in a naive way (i.e., replacing the PCPPs in the construction of Ben-Sasson*et al.*with our scPCPPs) would yield codes with *polynomial* length, whereas we are aiming for nearly-linear length. Instead, we use an alternative approach.

The key idea is to provide scPCPPs that only refer to sufficiently short statements such that even with the polynomial blow-up of the scPCPP, the length of each proof would still be sub-linear. Specifically, instead of providing proofs for the validity of the entire codeword and the consistency of each message bit with the entire codeword (as in [BSGH+06]), we provide proofs for the consistency of each message bit with "small" parts of the code and for the validity of these small parts. If each part is sufficiently small (i.e., of length k^α for an arbitrarily small constant $\alpha > 0$, where k is the length of the message), then we can still obtain a code with nearly-linear length, even when providing polynomial length proofs for all of the small parts.

The caveat, however, is that proving that each message bit is consistent with a small part (or *local view*) of a codeword does *not* necessary imply that the message bit is consistent with the entire codeword. Similarly, partitioning a codeword into small parts and proving the validity of each part does not imply the validity of the entire codeword. Therefore, we need the base code (to which we append scPCPPs) to be highly structured so that, loosely speaking, the *local* consistency and validity we are able to ascertain can be used to enforce *global* consistency and validity. Concretely, the strategy we employ is using *tensor codes* and proving that this family of codes has features that allow us to overcome the aforementioned caveat. Details follow.

Given a linear code $C : \{0,1\}^k \to \{0,1\}^n$, the tensor code $C \otimes C : \{0,1\}^{k^2} \to \{0,1\}^{n^2}$ consists of all $n \times n$ matrices whose rows and columns are codewords of C. Similarly, the d-dimensional tensor code $C^{\otimes d} = \underbrace{C \otimes C \otimes \cdots \otimes C}_{\text{d times}} : \{0,1\}^{k^d} \to \{0,1\}^{n^d}$ is defined in the natural way. Namely, $C^{\otimes d}$ consists of all $\underbrace{n \times n \times \cdots \times n}_{\text{d times}}$-dimensional tensors such that each (axis-parallel) line in the tensor is a codeword of C.[7] (See Section 5.2.3 for the exact definitions.)

Taking a brief pause, we mention that the use of tensor products towards the construction of local testable codes was suggested by Ben-Sasson and Sudan [BS06] who initiated the study of tensor codes in the context of local testability. As hinted above and will become evident next, tensor codes are a key ingredient in our constructions, and we rely on a recent analysis of the testability of these codes (provided recently by

[7] Axis-parallel lines in high-dimensional tensors simply generalize the notion of rows and columns in $n \times n$ matrices.

Viderman [Vid12], which builds upon Ben-Sasson and Sudan [BS06]).

Towards obtaining relaxed local decodability, we show that tensor codes satisfy a feature, which we call *local propagation*, that allows us to verify *global* consistency statements (such as the ones that are used in the [BSGH+06] relaxed-LDC) by verifying *local* consistency statements, which we can afford to prove with *polynomial* length scPCPPs; the *local propagation* feature of tensor codes is discussed in Section 5.4. Hence, we can ascertain that the value at each point in the tensor is consistent with the entire codeword by verifying the consistency of a constant number of randomly selected statements regarding small parts of the tensor (specifically, statements of consistency between the value at a point in the tensor and a line that passes through it). We remark that Theorem 5.2 can be used to derive polynomial-length scPCPPs for such statements (see Section 5.6). Therefore, we can replace the nearly-linear length PCPPs that are used in [BSGH+06] with our polynomial length scPCPPs, while preserving the functionality of relaxed local decoding and keeping the total length of the construction nearly-linear. (See Section 5.4 for a more detailed high-level description of our approach, followed by a full proof in Section 5.4.2.)

Recapping, so far our construction is as follows. Let C be a good linear code and $d \in \mathbb{N}$ be a sufficiently large constant. Each codeword of our code consists of the following equal-length parts: (1) repetitions of the tensor codeword $C^{\otimes d}(x)$ that encodes the message x; (2) repetitions of the explicitly given message x; and (3) scPCPPs for small statements (specifically, regarding the consistency of each point in the tensor $C^{\otimes d}(x)$ with each line that passes through it), which are used to ascertain the consistency of each individual bit in the message x with the codeword $C^{\otimes d}(x)$.[8]

Finally, we augment the aforementioned construction with a forth and last part that allows us to obtain *strong* local testability. The naive approach is to append a scPCPP that ascertains the validity of all three parts of our code. However, since the length of our scPCPPs is polynomial in the length of the statement, this approach would yield codes with long (polynomial) length. Instead, recall that we can (strongly) test the consistency of the first two parts via the third part (which is also strongly testable, since it is a scPCPP). Thus, in order to obtain strong local testability it suffices to ascertain that the first part is a valid codeword of $C^{\otimes d}$ using scPCPPs. Luckily, tensor codes also satisfy the *robustness* feature, which allows us to ascertain the validity of an entire codeword of $C^{\otimes d}$ by ascertaining the validity of small parts of the codeword. Detail follows.

Loosely speaking, a code is said to be *robust* if the corruption in a random "local view" of a codeword is proportional to the corruption in the entire codeword [BS06]. In more detail, we use a recent result of Viderman [Vid12] (building on Ben-Sasson and Sudan [BS06]) that states that the corruption in a random 2-dimensional (axis-parallel) plane of a corrupted codeword of a binary tensor code $C^{\otimes d}$ (where $d \geq 3$) is proportional to the corruption in the entire codeword. This feature allows us to ascertain the validity of the first part (i.e., the tensor codeword $C^{\otimes d}(x)$) by only providing scPCPPs for short

[8] We remark that the actual construction differs slightly from the above in that, for convenience, we use *systematic* tensor codes that contain the message explicitly in the encoding, instead of providing repetitions of the message as a part of the code.

statements that refer to 2-dimensional planes in $C^{\otimes d}(x)$. (See Section 5.5 for a more detailed high-level description, followed by a full proof.)

We remark that, for simplicity, we use the foregoing result in [Vid12] rather than [BS06], because the latter requires the base code C to have a large relative distance, which can only be obtained by codes with a sufficiently large alphabet. In contrast, we rely on the fact that our codes are binary in many places throughout our construction, most notably in our PCPPs (Theorem 5.6). Thus, it is convenient to use the stronger result in [Vid12], which allows us to work with binary codes throughout the construction.

5.1.5 Applications to Property Testing

As an application of our main result (Theorem 5.1) we improve on the best known separation result (due to [GR13c]) between the complexity of *decision* and *verification* in the setting of *property testing*.

The study of property testing, initiated by Rubinfeld and Sudan [RS96] and Goldreich, Goldwasser and Ron [GGR98], considers highly-efficient randomized algorithms that solve approximate decision problems, while only inspecting a small fraction of the input. Recently, Gur and Rothblum [GR13c] initiated the study of MA proofs of proximity (hereafter MAPs), which can be viewed as the NP analogue of property testing. They reduced the task of separating the power of property testers and MAPs to the design of very local codes, both in terms of testability and decodability. Furthermore, they noticed that for such a separation, *relaxed decodability* would suffice.

Gur and Rothblum used several weaker codes to obtain weaker separation results than the one we obtain here. Specifically, they either show a smaller gap between the query complexity of testers and MAPs, or show a separation for a limited range of the proximity parameter. In contrast, by plugging-in the code of Theorem 5.1, we obtain the best known (exponential) separation result between the power of MAPs and property testers.

Theorem 5.3 (Informal)**.** *There exists a property that requires $n^{0.999}$ queries for every property tester but has an* MAP *that uses a proof of* logarithmic *length and makes* $\mathsf{poly}(1/\varepsilon)$ *queries.*

For more information regarding this application, we refer the reader to Section 5.7.

5.1.6 Organization.

In Section 5.2 we provide the preliminaries. In Section 5.3 we describe the construction of the codes that establish Theorem 5.1. In Section 5.4 and Section 5.5 we establish the relaxed local decodability and strong local testability (respectively) of the codes. In Section 5.6 we construct the scPCPPs needed for our construction, and finally, in Section 5.7 we present an application of our codes for property testing.

5.2 Preliminaries

We start with some general notation. We denote by $[n]$ the set of numbers $\{1, 2, \ldots, n\}$. For $i \in [n]$ and for $x \in \{0,1\}^n$, denote by x_i the i^{th} bit of x. For $x, y \in \{0,1\}^n$, we denote by $\Delta(x, y)$ the Hamming distance between x and y, and denote by $\delta(x, y)$ the *relative* (Hamming) distance between x and y, i.e., $\delta(x, y) = \Delta(x, y)/n$. We say that x is δ-close to (respectively, δ-far from) y if the relative distance between x and y is at most δ (respectively, at least δ).

Given a set S, we denote by $s \in_R S$ the distribution that is obtained by selecting uniformly at random $s \in S$. For a randomized algorithm A, we write $\Pr_A[\cdot]$ (or $\mathbf{E}_A[\cdot]$) to state that the probability (or expectation) is over the internal randomness of the algorithm A.

(Non) Uniformity. Throughout this chapter, for the simplification of the presentation, we formally treat algorithms (testers, decoders, and verifiers) as (non-uniform) *polynomial-size circuits*. We note, however, that all of our algorithms can be made uniform by making straightforward modifications. Furthermore, it will be convenient for us to view the length $n \in \mathbb{N}$ of objects as fixed. We note that although we fix n, it should be viewed as a generic parameter, and so we allow ourselves to write asymptotic expressions such as $\mathsf{poly}(n)$, $O(n)$, etc. In contrast, when we say that something is a constant, we mean that it is independent of the length parameter n.

5.2.1 Error Correcting Codes

Let $k, n \in \mathbb{N}$. A binary linear code $C : \{0,1\}^k \to \{0,1\}^n$ of distance d is a linear mapping over $\mathsf{GF}(2)$, which maps messages to codewords, such that the Hamming distance between any two codewords is at least $\mathsf{d} = \mathsf{d}(n)$. The relative distance of C, denoted by $\delta(C)$, is given by d/n. The length of a code is $n = n(k)$. By slightly abusing notation, we say that we can construct a code C with nearly linear length if for any constant $\alpha > 0$ we can construct a code $C\colon \{0,1\}^k \to \{0,1\}^n$, where $n = k^{1+\alpha}$. For any $x \in \{0,1\}^n$, denote the relative distance of x to the code C by $\delta_C(x) = \min_{y \in C}\{\delta(x, y)\}$.

We say that C is systematic, if the first k bits of every codeword of C contain the message; that is, if for every $x \in \{0,1\}^k$ and every $i \in [k]$ it holds that $C(x)_i = x_i$. Since C is a linear code, we may assume without loss of generality that it is systematic.

5.2.2 Local Testability and Decodability

Following the discussion in the introduction, *strong* locally testable codes are defined as follows.

Definition 5.1 (strong-LTC)**.** *A code $C : \{0,1\}^k \to \{0,1\}^n$ is a* strong-LTC*, if there exists a probabilistic algorithm (tester) T that, given oracle access to $w \in \{0,1\}^n$, makes $O(1)$ queries to w, and satisfies:*

1. *Completeness: For any codeword w of C it holds that $T^w = 1$.*
2. *Strong Soundness: For all $w \in \{0,1\}^n$,*

$$\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_C(w)\big).$$

We say that a tester makes nearly-uniform *queries if it queries each bit in the (alleged) codeword input $w \in \{0,1\}^n$ with probability $\Theta(1/n)$.*

Following the discussion in the introduction, *relaxed* locally decodable codes are defined as follows.

Definition 5.2 (relaxed-LDC)**.** *A code $C : \{0,1\}^k \to \{0,1\}^n$ is a relaxed-*LDC *if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$, a constant $\rho > 0$ and a probabilistic algorithm (decoder) D that, given oracle access to $w \in \{0,1\}^n$ and explicit input $i \in [k]$, makes $O(1)$ queries to w, and satisfies:*

1. *Completeness: For any $i \in [k]$ and $x \in \{0,1\}^k$ it holds that $D^{C(x)}(i) = x_i$.*
2. *Relaxed Soundness: For any $i \in [k]$ and any $w \in \{0,1\}^n$ that is δ_{radius}-close to a codeword $C(x)$,[9] it holds that*

$$\Pr_D[D^w(i) \in \{x_i, \perp\}] \geq 2/3.$$

3. *Success Rate: For every $w \in \{0,1\}^n$ that is δ_{radius}-close to a codeword $C(x)$, and for at least a ρ fraction of the indices $i \in [k]$, with probability at least $2/3$ the decoder D outputs the i^{th} bit of x. That is, there exists a set $I_w \subseteq [l]$ of size at least ρk such that for every $i \in I_w$ it holds that $\Pr_D[D^w(i) = x_i] \geq 2/3$.*

We remark that our definition is slightly *stronger* than the one given in [BSGH+06] as we require *prefect* completeness (i.e., that the decoder *always* outputs the correct value given oracle access to a valid codeword of the code C).

5.2.3 Tensor Codes

Tensor codes are defined as follows.

Definition 5.3. *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code. The tensor code $C \otimes C : \{0,1\}^{k^2} \to \{0,1\}^{n^2}$ is the code whose codewords consists of all $n \times n$ matrices such that each axis-parallel line (i.e., a row or a column) in the matrix is a codeword of C. Similarly, given $d \in \mathbb{N}$, the tensor code $C^{\otimes d} : \{0,1\}^{k^d} \to \{0,1\}^{n^d}$ is the code whose codewords consists of all d-dimensional tensors such that each axis-parallel line in the tensor is a codeword of C.*

[9] Note that since $\delta_{\mathsf{radius}} < \delta(C)/2$, for every $x \in \{0,1\}^n$ that is δ_{radius}-close to C there exists a *unique* codeword x' of C such that x is $\delta_{C'}(x)$-close to x'.

It is well-known that for every $d \in \mathbb{N}$ the tensor code $C^{\otimes d}$ is a linear code with relative distance $\delta(C)^d$ (see e.g., [BS06]). Given a message $x \in \{0,1\}^{k^d}$ and coordinate $\bar{\imath} = (\bar{\imath}_1, \ldots, \bar{\imath}_d) \in [n]^d$, we denote the value of $C^{\otimes d}(x)$ at coordinate $\bar{\imath}$ by $C^{\otimes d}(x)_{\bar{\imath}}$.

Remark 5.4. *By the definition of tensor codes, if a linear code C is systematic, then the tensor code $C^{\otimes d}$ is also a systematic code;*[10] *that is, for every $x \in \{0,1\}^{k^d}$ and $\bar{\imath} \in [k]^d$ it holds that $C^{\otimes d}(x)_{\bar{\imath}} = x_{\bar{\imath}}$.*

Next, we provide notations for the restriction of tensors to lines and planes. We start by defining axis-parallel lines.

Definition 5.5 (Axis-Parallel Lines)**.** *For $j \in [d]$ and $\bar{\imath} = (i_1, \ldots, i_d) \in [n]^d$, we denote by $\ell_{j,\bar{\imath}}$ the j^{th} axis-parallel line passing through $\bar{\imath}$. That is,*

$$\ell_{j,\bar{\imath}} = \{(i_1, \ldots, i_{j-1}, x, i_{j+1}, \ldots, i_d)\}_{x \in [n]} .$$

We denote by $\mathsf{Lines}(n,d)$ *the* multi-set *that contains all axis-parallel lines that pass through each point $\bar{\imath} \in [n]^d$.*[11] *That is,* $\mathsf{Lines}(n,d) = \{\ell_{j,\bar{\imath}}\}_{\bar{\imath} \in [n]^d, j \in [d]}$. *Lastly, given a tensor $w \in \{0,1\}^{n^d}$ we denote by $w|_{\ell_{i,j}} \in \{0,1\}^n$ the* restriction *of w to the line $\ell_{i,j}$, i.e., the j^{th} axis-parallel line that passes through $\bar{\imath}$.*

Next, we define axis-parallel planes.

Definition 5.6 (Axis-Parallel (2-dimensional) Planes)**.** *For $j_1 < j_2 \in [d]$ and $\bar{\imath} = (i_1, \ldots, i_d) \in [n]^d$, we denote by $\mathsf{p}_{j_1,j_2,\bar{\imath}}$ the $(j_1, j_2)^{th}$ axis-parallel plane passing through the point $\bar{\imath}$. That is*

$$\mathsf{p}_{j_1,j_2,\bar{\imath}} = \{(i_1, \ldots, i_{j_1-1}, x_1, i_{j_1+1}, \ldots, i_{j_2-1}, x_2, i_{j_2+1}, \ldots, i_d)\}_{x_1,x_2 \in [n]} .$$

We denote by $\mathsf{Planes}(n,d)$ *the* set *of all (distinct) axis-parallel planes in all directions in $\{0,1\}^{n^d}$.*[12] *Lastly, for a tensor $w \in \{0,1\}^{n^d}$ and a plane $\mathsf{p} \in \mathsf{Planes}(n,d)$ we denote by $w|_{\mathsf{p}} \in \{0,1\}^{n^2}$ the* restriction *of w to the coordinates in the plane p.*

Throughout this chapter we deal with axis-parallel lines (respectively, axis-parallel planes); hence, for brevity, we will sometimes refer to an *axis-parallel line* (respectively, *axis-parallel plane*) simply as a *line* (respectively, *plane*). We remark that the multi-set $\mathsf{Lines}(n,d)$ contains $d \cdot n^d$ lines and the set $\mathsf{Planes}(n,d)$ contains $\binom{d}{2} \cdot n^{d-2}$ planes. We omit the parameters n and d when they are clear from the context.

[10] We view the restriction of the tensor $C^{\otimes d}$ to the coordinates in $[k]^d$ as the prefix of $C^{\otimes d}$.

[11] Note that each axis-parallel line in $\{0,1\}^{n^d}$ appears n times in $\mathsf{Lines}(n,d)$.

[12] Unlike the *multi-set* $\mathsf{Lines}(n,d)$, which contains n copies of each line, there is no redundancy in the *set* $\mathsf{Planes}(n,d)$.

Testing Tensor Codes. The next theorem, which is implicit in [Vid12], shows that for every $d \geq 3$ and every linear code C, testing the tensor-code $C^{\otimes d}$ can be reduced to testing whether a random plane in C is a codeword of $C^{\otimes 2}$.

Theorem 5.4. *Let C be a linear binary code and $d \geq 3$ an integer. Then, there exists a constant $c_{\mathsf{robust}} \in (0,1)$ such that for every tensor $w \in \{0,1\}^{n^d}$ it holds that*

$$\mathop{\mathbf{E}}_{\mathsf{p} \in_R \mathsf{Planes}} \left[\delta\left(w|_{\mathsf{p}}, C^{\otimes 2} \right) \right] > c_{\mathsf{robust}} \cdot \delta_{C^{\otimes d}}(w).$$

Specifically, in [Vid12, Theorem A.5] it is shown that for $d \geq 3$, if a codeword w of a tensor code $C^{\otimes d}$ is corrupted, then the corruption in a random $(d-1)$-dimensional subplane of w is proportional to the corruption in the entire tensor w. By applying this result recursively (a constant number of times), we obtain Theorem 5.4. For completeness, we provide the proof of Theorem 5.4 in Section 5.8.3.

5.2.4 PCPs of Proximity

Strong canonical PCPs of proximity were defined as follows in [GS06, Section 5.3].

Definition 5.7 (scPCPPs)**.** *Let V be a probabilistic algorithm (verifier) that is given* oracle *access to an* input *$x \in \{0,1\}^n$ and* oracle *access to a* proof *$\pi \in \{0,1\}^{\ell(n)}$, where $\ell : \mathbb{N} \to \mathbb{N}$ satisfies $\ell(n) \leq \mathsf{exp}(\mathsf{poly}(n))$. We say that V is a* strong (canonical) PCPP verifier *for language L if it makes $O(1)$ queries and satisfies the following two conditions:*

- *Canonical Completeness: For all $x \in L$, there exists a* unique canonical *proof for x, denoted $\pi_{\mathsf{canonical}}(x)$, such that the verifier always accepts the pair $(x, \pi_{\mathsf{canonical}}(x))$; i.e., $V^{x,\pi_{\mathsf{canonical}}(x)} = 1$.*
- *Strong Canonical Soundness: For any input $x' \in \{0,1\}^n$ and proof $\pi' \in \{0,1\}^{\ell(|x|)}$ the verifier rejects with probability at least $\mathsf{poly}\big(\delta_{\mathsf{PCPP}}(x', \pi')\big)$, where*

$$\delta_{\mathsf{PCPP}}(x', \pi') \triangleq \min_{x \in \{0,1\}^n} \left\{ \max\left(\frac{\Delta(x, x')}{n} \; ; \; \frac{\Delta(\pi_{\mathsf{canonical}}(x), \pi')}{\ell(n)} \right) \right\}, \tag{5.1}$$

  where for any $x \notin L$ we define $\pi_{\mathsf{canonical}}(x) = \lambda$ and say that any π' is 1-far from λ.

We say that a scPCPP *verifier makes* nearly-uniform *queries if it queries each bit in the input x with probability $\Theta(1/|x|)$ and queries each bit in the proof $\pi(x)$ with probability $\Theta(1/|\pi|)$.*

We stress that these scPCPPs have *one-sided error* (i.e., they always accept inputs in L coupled with their canonical proofs). Note that the *canonical* aspect is reflected in the dependence of $\delta_{\mathsf{PCPP}}(x', \pi')$ on $\Delta(\pi_{\mathsf{canonical}}(x), \pi')$, whereas the *strong-soundness* aspect is reflected in the tight relation between the rejection probability and $\delta_{\mathsf{PCPP}}(x', \pi')$.

5.3 The Main Construction

In this section we describe our construction of a family of binary linear codes that are both (constant-query) relaxed-LDCs and strong-LTCs with constant relative distance and nearly-linear length. Our codes rely heavily on special-purpose *strong canonical* PCPs *of proximity* (with polynomial proof length), which we construct in Section 5.6, and so, we start by stating these scPCPPs. Our first family of scPCPPs is for good linear codes.

Theorem 5.5 (scPCPPs for good codes)**.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code with constant relative distance and linear length. Then, there exists a* scPCPP *for codewords of C (i.e., for the set $\{ C(x) \}_{x \in \{0,1\}^k}$). Furthermore, the proof length of the* scPCPP *is* $\mathsf{poly}(n)$*, the* scPCPP *verifier makes nearly-uniform queries, and the canonical* scPCPP *proofs are linear (over* $\mathsf{GF}(2)$*).*

As a corollary of Theorem 5.5, we obtain a family of scPCPPs for *half-spaces* of any good linear code. That is, scPCPPs that ascertain membership in the set of all codewords wherein one given location is set to a specific value (for example, all codewords that have 1 in their first location).

Theorem 5.6 (scPCPPs for half-spaces of good codes)**.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code with constant relative distance and linear length. Let $i \in [k]$ be a location in a message and $b \in \{0,1\}$ a bit. Then, there exists a* scPCPP *for $C_{i,b}$, where $C_{i,b}$ is the set of all codewords w of C such that the i^{th}-bit of w equals b (i.e., $w_i = b$). Furthermore, the proof length of the* scPCPP *is* $\mathsf{poly}(n)$*, the* scPCPP *verifier makes nearly-uniform queries, and the* scPCPP *proofs are linear (over* $\mathsf{GF}(2)$*).*

See Section 5.6 for the full proofs of Theorems 5.5 and 5.6. Equipped with the foregoing scPCPPs, we describe the construction of our code, which consists of three parts. (See Section 5.2 for relevant notation.)

Tensor code part. Let $C_0 : \{0,1\}^k \to \{0,1\}^n$ be a systematic linear code with linear length (i.e., $n = \Theta(k)$) and constant relative distance $0 < \delta(C_0) < 1$. Let $d \geq 3$ be a sufficiently large constant (to be determined later). Let $C \triangleq (C_0)^{\otimes d} : \{0,1\}^{k^d} \to \{0,1\}^{n^d}$ be the d-tensor product of C_0. By Remark 5.4, since C_0 is systematic, then C is also systematic. Recall that $\delta(C) = \delta(C_0)^d$, hence $\delta(C)$ is a constant.

We augment the code C with scPCPPs that ascertain the validity of each *plane* in C (using Theorem 5.5) and scPCPPs that ascertain the consistency of each bit in C with each line that passes through it (using Theorem 5.6). Details follow.

Plane scPCPPs part. Let $C(x)$ be a codeword of the tensor code C. For every plane p in the tensor $C(x)$ we use our scPCPP*s for good codes* to prove that the restriction of $C(x)$ to the plane p (denoted by $C(x)|_{\mathsf{p}}$) is a codeword of $C_0^{\otimes 2}$. Specifically, for a codeword w of $C_0^{\otimes 2}$ we denote by $\pi_{\mathsf{plane}}(w)$ the corresponding canonical proof for the scPCPP verifier

of Theorem 5.5. Then, for every message $x \in \{0,1\}^{k^d}$ we define $\pi_{\mathsf{planes}}(x)$ as the sequence of the canonical proofs for *all* planes in $C(x)$; that is,

$$\pi_{\mathsf{planes}}(x) = \{\pi_{\mathsf{plane}}(C(x)|_{\mathsf{p}})\}_{\mathsf{p}\in\mathsf{Planes}},$$

where Planes is the set of *all* (2-dimensional) axis-parallel planes in $\{0,1\}^{n^d}$ (see Definition 5.6).

We append $\pi_{\mathsf{planes}}(x)$ to the codeword $C(x)$. Note that $|\pi_{\mathsf{planes}}(x)| = \binom{d}{2} n^{d-2} \cdot |\pi_{\mathsf{plane}}(C(x)|_{\mathsf{p}})| \leq n^{d+O(1)}$. We stress that the constant in the $O(1)$ notation does *not* depend on d. These scPCPPs will be used for the local testability of our code (see Section 5.5).

Point-line scPCPPs part. Let $C(x)$ be a codeword of the tensor code C. For every point $\bar{\imath} = (i_1, \ldots, i_d) \in [n]^d$ and every direction $j \in [d]$ we use our scPCPP*s for half-spaces of good codes* to prove that the restriction of $C(x)$ to the line that passes through point $\bar{\imath}$ in direction j (denoted by $C(x)|_{\ell_{j,\bar{\imath}}}$) is a codeword of C_0 that is consistent with value of $C(x)$ at point $\bar{\imath}$.[13] Specifically, for a codeword w of C_0 and index $s \in [n]$ we denote by $\pi_{\mathsf{line}}(w, s)$ the canonical proof for the scPCPP verifier of Theorem 5.6 (which corresponds to codewords of C_0 whose s^{th}-bit equals to w_s). Then, for every message $x \in \{0,1\}^{k^d}$ we define $\pi_{\mathsf{lines}}(x)$ as the set of the canonical proofs for *all* lines passing through each point in $C(x)$; that is,

$$\pi_{\mathsf{lines}}(x) = \{\pi_{\mathsf{line}}(C(x)|_{\ell_{j,\bar{\imath}}}, i_j)\}_{\ell_{j,\bar{\imath}}\in\mathsf{Lines}},$$

where $\mathsf{Lines} = \{\ell_{j,\bar{\imath}}\}_{\bar{\imath}\in[n]^d, j\in[d]}$, as in Definition 5.5 (i.e., the set Lines contains all axis-parallel lines that pass through each point $\bar{\imath} \in [n]^d$).

We append $\pi_{\mathsf{lines}}(x)$ to the codeword $C(x)$. Note that $|\pi_{\mathsf{lines}}(x)| = d \cdot n^d \cdot |\pi_{\mathsf{line}}(C(x)|_{\ell})| \leq n^{d+O(1)}$, where the constant in the $O(1)$ notation does *not* depend on d. These scPCPPs will be used for the relaxed local decodability of our code (see Section 5.4).

Putting it all together. Our construction is obtained by combining the tensor codeword $C(x)$ with $\pi_{\mathsf{lines}}(x)$ and $\pi_{\mathsf{planes}}(x)$, while ensuring that the three parts are of equal length. That is, for $k' = k^d$ define $C' : \{0,1\}^{k'} \to \{0,1\}^{n'}$ as follows.

$$C'(x) \triangleq \left((C(x))^{t_1} \ , \ (\pi_{\mathsf{lines}}(x))^{t_2} \ , \ (\pi_{\mathsf{planes}}(x))^{t_3} \right)$$

where t_1, t_2 and t_3 are the *minimal* integers such that $|C(w)|^{t_1} = |\pi_{\mathsf{lines}}(w)|^{t_2} = |\pi_{\mathsf{planes}}(w)|^{t_3}$.[14]

Length and relative-distance of C'. For sufficiently large d the length of C' is nearly-linear. To this end, observe that for every $x \in \{0,1\}^{k^d}$ it holds that $|C(x)| = n^d$,

[13] Note that the $\bar{\imath}^{\text{th}}$-bit of $C(x)$ is, in fact, the $i_j{}^{\text{th}}$-bit of the line $C(x)|_{\ell_{j,\bar{\imath}}}$.

[14] Ignoring integrality issues, we can say that we "blow" the lengths of the two shorter parts to match the length of the longest part, which (in case of our implementation of the scPCPPs) is the part of the *plane* scPCPPs. Hence, actually, $t_3 = 1$.

$|\pi_{\mathsf{lines}}(x)| \leq \mathsf{poly}(n)$ and $|\pi_{\mathsf{planes}}(x)| \leq \mathsf{poly}(n^2)$. Hence, for every constant $\alpha > 0$, there exists some constant $d > 0$ so that

$$n' = n^{d+O(1)} = (O(1) \cdot k)^{d+O(1)} \leq (k')^{1+\alpha}.$$

The code C' has constant relative distance since the relative distance of C (denoted by $\delta(C)$) is constant, and since repetitions of C constitute a third of the length of C'; that is, $\delta(C') \geq \frac{\delta(C)}{3}$. In the next sections we prove the following theorem.

Theorem 5.7. *thm:code[restated] For every constant $\alpha > 0$, there exists some constant $d \geq 0$ so that the code $C' : \{0,1\}^{k'} \to \{0,1\}^{n'}$, as defined above, is a linear binary code that is a relaxed-*LDC *and a* strong-LTC *with constant relative distance.*

Specifically, in Section 5.4 we prove the relaxed-LDC feature of C', and in Section 5.5 we prove the strong-LTC feature of C'.

(Alleged) Codeword Notations. Consider an arbitrary string $w \in \{0,1\}^{n'}$ (which we think of as an alleged codeword). We view w as a string composed of three parts (analogous to the three parts of the construction above):

1. $\bar{c} = (c_1, \ldots, c_{t_1})$: the t_1 alleged repetitions of the tensor code part.
2. $\bar{p}^{\mathsf{lines}} = \left(\bar{p}_1^{\mathsf{lines}}, \ldots, \bar{p}_{t_2}^{\mathsf{lines}}\right)$: the t_2 alleged repetitions of the scPCPP proofs for all the point-line pairs (i.e., lines passing through all coordinates in all directions). For every $i \in [t_2]$, the string $\bar{p}_i^{\mathsf{lines}}$ consists of scPCPP proofs for every point-line pair, i.e., $\bar{p}_i^{\mathsf{lines}} = \{p_i^{\ell}\}_{\ell \in \mathsf{Lines}}$.
3. $\bar{p}^{\mathsf{planes}} = \left(\bar{p}_1^{\mathsf{planes}}, \ldots, \bar{p}_{t_3}^{\mathsf{planes}}\right)$: the t_3 alleged repetitions of the scPCPP proofs for all the (2-dimensional) planes. For every $i \in [t_3]$, the string $\bar{p}_i^{\mathsf{planes}}$ consists of scPCPP proofs for every plane, i.e., $\bar{p}_i^{\mathsf{planes}} = \{p_i^{\mathsf{p}}\}_{\mathsf{p} \in \mathsf{Planes}}$.

5.4 Establishing the Relaxed-LDC Property

In this section we prove that the code C', which was defined in Section 5.3, is a relaxed locally decodable code.

Theorem 5.8. *The code $C' : \{0,1\}^{k'} \to \{0,1\}^{n'}$ is a relaxed-*LDC.

In order to prove Theorem 5.8, it would be convenient to use an alternative definition of relaxed-LDCs, which implies the standard definition (Definition 6.3) by applying known transformations. Specifically, in Section 5.8.4 (following [BSGH+06, Section 4.2]) we show that it suffices to relax the soundness parameter in Definition 6.3 to $\Omega(1)$ (instead of 2/3), and replace the *success rate* condition with the following *average smoothness* condition. Loosely speaking, *average smoothness* requires that the decoder makes nearly uniform queries on average (over all indices to be decoded). By the foregoing, to prove Theorem 5.8 it suffices to show that the code C' satisfies the following definition.

Definition 5.8 (*Modified* relaxed-LDCs)**.** *A code $C : \{0,1\}^k \to \{0,1\}^n$ is a* modified *relaxed-*LDC *if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$ and a probabilistic algorithm (decoder) D that, given oracle access to $w \in \{0,1\}^n$ and explicit input $i \in [k]$, makes $q = O(1)$ queries to w, and satisfies:*

1. *Completeness: For any $i \in [k]$ and $x \in \{0,1\}^k$ it holds that $D^{C(x)}(i) = x_i$.*
2. *Modified Relaxed Soundness: For any $i \in [k]$ and any $w \in \{0,1\}^n$ that is δ_{radius}-close to a codeword $C(x)$ it holds that*

$$\Pr_D[D^w(i) \in \{x_i, \perp\}] = \Omega(1).$$

   where $\delta_{\mathsf{radius}} \in (0, \delta(C')/2)$, the decoding radius of C, is a universal constant, to be determined later.

3. *Average Smoothness: for every $w \in \{0,1\}^n$ and $v \in [n]$,*

$$\Pr_{i,j,r}\left[\mathcal{D}^w(i,j,r) = v\right] < \frac{2}{n},$$

   where $\mathcal{D}^w(i,j,r)$ denotes the distribution of the j^{th} query of the decoder D^w on coordinate i and coin tosses r, where the probability is taken uniformly over all possible choices of $i \in [k]$, $j \in [q]$, and coin tosses r.

We remark that in [BSGH+06, Section 4.2], the definition of average smoothness also requires a matching lower bound, i.e., the decoder should satisfy $\frac{1}{2n} < \Pr_{i,j,r}\left[\mathcal{D}^w(i,j,r) = v\right] < \frac{2}{n}$. However, for our applications it suffices to only require the upper bound. We note that the lower bound can be easily obtained by adding (random) dummy queries.

We start by showing a decoder that satisfies the first two aforementioned conditions (i.e., the *completeness* condition and the *modified relaxed soundness*). Next, in Section 5.4.3 we show how to obtain a related decoder that also satisfies the *average smoothness* condition.

The Setting. Consider an arbitrary input $w \in \{0,1\}^{n'}$ such that $0 \leq \delta_{C'}(w) < \delta_{\mathsf{radius}}$. We view w as a string composed of three parts as in Section 5.3, i.e., $w = (\bar{c}, \bar{p}^{\mathsf{lines}}, \bar{p}^{\mathsf{planes}})$. We stress that any part of w might suffer from corruptions, and so, we have to be able to decode correctly assuming that not too many corruptions have occurred (i.e., less than δ_{radius} fraction). Denote by x the unique string such that w is $\delta_{C'}(w)$-close to $C(x)$ (see Footnote 9).

High-Level Idea. Recall that a valid codeword of C' consists of three (repeated) parts: (1) a systematic tensor code C, (2) point-line scPCPPs, and (3) plane scPCPPs. Our general approach is to decode according to the prefix of the first part (which allegedly contains the message x explicitly (since we use a systematic code), and to use the second part to ensure that each bit in message x is consistent with the rest of the (tensor)

codeword $C(x)$. (The third part is not used here; it is only used for the testability of the code.) Thus, the task of (relaxed) decoding the i^{th} bit of the message is reduced to verifying that the explicitly given value of the i^{th} bit of the message is consistent with the rest of the codeword.

Towards this end, recall that the second part of each codeword contains scPCPPs that ascertain the consistency of each bit in the tensor with each line that passes through it, but not consistency with the entire tensor. Therefore, in order to verify the consistency of each message bit with the entire codeword, our decoder uses a feature of tensor codes, which we call *local propagation*. This feature allows us to verify the consistency of a single message bit with the entire codeword by verifying the consistency of a carefully chosen sequence of d point-line pairs (using the *point-line* scPCPP). Details follow.

Loosely speaking, the *local propagation* feature of tensor codes implies that if one corrupts a single point in a codeword and attempts to keep most local views (say, lines in the tensor) consistent with this corruption, then a chain of highly structured modifications must be made that causes the "corruption" to propagate throughout the entire tensor. This is best exemplified by our decoder, which is tailored to take advantage of the foregoing phenomena.

Our decoder is given a coordinate $\bar{\imath} = (i_1, \ldots, i_d) \in [k]^d$ and oracle access to an alleged codeword w as above. The decoder looks for "inconsistencies" in w and if it finds any, it outputs $\perp$. Otherwise, it simply output $w_{\bar{\imath}}$ (which should contain the $\bar{\imath}^{\text{th}}$ bit of the message). Since our base code C_0 has constant relative distance, in order to "corrupt" the point $\bar{\imath}$ in the tensor code without causing the lines that pass through $\bar{\imath}$ to be inconsistent with the corrupted value at $\bar{\imath}$, one has to corrupt a *constant fraction* of each line on which $\bar{\imath}$ resides. Thus, our decoder uses the scPCPPs to verify that a line ℓ that passes through $\bar{\imath}$ is consistent with the value at $\bar{\imath}$, assuring that a constant fraction of many lines on which $\bar{\imath}$ resides is corrupted.

Similarly, in order to "corrupt" a constant fraction of the line ℓ in the tensor codeword without causing inconsistency between the corrupted points in ℓ and the lines that pass through these corrupted points, one has to change a *constant fraction* of each line that passes through a corrupted point in ℓ (therefore, corrupting a *constant fraction* of each *plane* wherein the line ℓ resides). Thus, our decoder uses the scPCPPs to verify that the line that passes through a random point $\bar{\imath}'$ in ℓ (which is corrupted with probability $\Omega(1)$) is consistent with the value at $\bar{\imath}'$, assuring that a constant fraction of many planes on which line ℓ resides were corrupted.

Thus, if the $\bar{\imath}^{\text{th}}$ point of the tensor codeword (i.e., the bit we wish to decode) is *corrupted*, then by iteratively continuing this procedure d times, and only performing d point-line consistency tests, the decoder can detect the corruption in $\bar{\imath}$ with high probability, unless a large fraction of the codeword is corrupted (i.e., the corruption at a single point, $\bar{\imath}$, propagated to the entire tensor).

We remark that in the proof that C' is a relaxed-LDC we do not use the *strongness* and *canonicity* properties of the scPCPPs (they are only used to prove that C' is a strong-LTC). Furthermore, since in the following we only wish to present a decoder satisfies Condition 1 and 2 of Definition 5.8, we can allow the decoder to output a "don't-

know" symbol whenever the codeword is corrupted.[15] Thus, we are not concerned with corruptions in the scPCPP parts, since a corruption in these parts can only increase the rejection probability for strings that are not codewords. Regarding inputs that are legal codewords, there are no corruptions and hence, no "inconsistencies". Thus, for legal codewords our tester will always output the correct value.

5.4.1 Warm-up: Two-Dimensional Tensors

Before we proceed to prove Theorem 5.8, we sketch a proof for two-dimensional tensor codes; that is, when we set $d = 2$ in the construction that appears in Section 5.3. In this warm-up, towards the end of simplifying the presentation, we make the following assumptions: We omit the third part of the codeword (i.e., the plane scPCPPs), and we omit the repetitions of the first and second parts of the code (i.e., the tensor code, and the point-line scPCPPs) and assume instead that the lengths of the first and the second parts are equal. We note that both assumptions can be easily removed (see Section 5.4.2 for details).

Let $w = (c, p)$ be an alleged codeword that consists of two parts of equal length: (1) c, an alleged 2-dimensional tensor code $C_0^{\otimes 2} : \{0,1\}^{k^2} \to \{0,1\}^{n^2}$, and (2) p, a sequence of alleged scPCPPs for every pair of point $\bar{\imath}$ in $[n]^2$ and line ℓ in $C_0^{\otimes 2}$ that passes through $\bar{\imath}$; each scPCPP ascertains that the line ℓ is a codeword of C_0 that is consistent with the value at the point $\bar{\imath}$.

Given a point $\bar{\imath} = (i_1, i_2) \in [k]^2$, the decoder first runs the point-line scPCPP that corresponds to $\bar{\imath}$ and the line $\ell_{1,\bar{\imath}} = \{(x, i_2)\}_{x \in [n]}$ passing through $\bar{\imath}$ in direction "1" (i.e., parallel to the first axis), and outputs $\perp$ if the scPCPP verifier rejected. Otherwise, the decoder picks a random point $\bar{\imath}' = (i_1', i_2')$ on the line $\ell_{1,\bar{\imath}}$, runs the corresponding scPCPP for $\bar{\imath}'$ and the line $\ell_{2,\bar{\imath}'} = \{(i_1', x)\}_{x \in [n]}$ that passes through $\bar{\imath}'$ in direction "2", and output $\perp$ if the scPCPP verifier rejected. If none of the scPCPP verifiers rejected, the verifier outputs $c_{\bar{\imath}}$.

For the *completeness* condition, assume that the decoder is given a valid codeword. In this case, the first part is indeed a valid copy of $C_0^{\otimes 2}(x)$, and the second part consists of the canonical proofs for $C_0^{\otimes 2}(x)$. Hence, all of the scPCPP verifiers accept, and since $C_0^{\otimes 2}(x)_{\bar{\imath}} = x_{\bar{\imath}}$, the decoder succeeds in decoding $x_{\bar{\imath}}$.

For the (modified) *relaxed soundness* condition, assume that the decoder is given a corrupted codeword $w = (c, p)$ that is δ-close to a valid codeword $C_0^{\otimes 2}(x)$, where $\delta \leq \delta_{\mathsf{radius}}$ for a sufficiently small (constant) *decoding radius* δ_{radius}. Note that if $c_{\bar{\imath}} = x_{\bar{\imath}}$, then the decoder satisfies the soundness condition (since it always outputs either x_i or $\perp$); hence, we assume that $c_{\bar{\imath}} \neq x_{\bar{\imath}}$. In this case, when the decoder runs the scPCPP verifier for $\bar{\imath}$ and (the restriction of c to) $\ell_{1,\bar{\imath}}$ it does not reject (with high probability) only if $C|_{\ell_{1,\bar{\imath}}}$ is "close" to a codeword of C_0 that disagrees with c on $\bar{\imath}$ (since the $i_2{}^{\text{th}}$ bit of this codeword

[15] Recall that the *completeness* condition of Definition 5.8 requires the decoder to successfully decode valid codeword, and the *modified relaxed soundness* condition requires that the decoder does not make a mistake in the decoding with probability at least $\Omega(1)$. However, the decoder is allowed to output a "don't-know" symbol with arbitrary probability on any (even on only slightly) corrupted codeword.

of C_0 must be different than $x_{\bar{\imath}}$). Since C_0 is a code with constant relative distance, this implies that a constant fraction of the line $\ell_{1,\bar{\imath}}$ must be corrupted (i.e., the restriction of c to the line $\ell_{1,\bar{\imath}}$ is $\Omega(1)$-far from its corresponding line in $C(x)$) for the scPCPP verifier to accept. Finally, if the decoder selected $\bar{\imath}'$ that is one of the $\Omega(n)$ corrupted points on $\ell_{1,\bar{\imath}}$, then by the same argument, a constant fraction points on the restriction of c to the line $\ell_{2,\bar{\imath}'}$ (that passes through $\bar{\imath}'$) must be corrupted. We deduce that in order to both scPCPP verifiers to accept (and hence defy the soundness condition), c must contain $\Omega(n^2)$ corrupted points, i.e., c should be β-far from $C_0^{\otimes 2}(x)$ for some constant β. By fixing $\delta_{\mathsf{radius}} < \beta$, we prevent this possibility.

5.4.2 The General Case

We proceed with the full proof that C' has a decoder that satisfies the first two conditions in the definition of a relaxed-LDC (i.e., the *completeness* and *(modified) relaxed soundness* conditions of Definition 5.8). We generalize the decoder of Section 5.4.1 to d-dimensional tensors and ensure it works without the assumptions that were made there for simplicity. The decoder D is formally described in Figure 5.1.

Let $\bar{\imath} \in [k]^d$. The *completeness* of the decoder is immediate from the construction: If the input is a codeword, i.e., $w = C'(x)$ and all of the scPCPPs proofs are the canonical proofs for $C'(x)$ (i.e., $\bar{p}^{\mathsf{lines}}$ and $\bar{p}^{\mathsf{planes}}$), then all of the executions of the scPCPP verifiers accept (since the scPCPP verifiers are with one-sided error). Recalling that, by definition, $C(x)_{\bar{\imath}} = C_0^{\otimes d}(x)_{\bar{\imath}} = x_{\bar{\imath}}$, the decoding procedure $D^w(\bar{\imath})$ returns $x_{\bar{\imath}}$ with probability 1, as required.

Next, we prove the *(modified) relaxed soundness* of the decoder. Let $w \in \{0,1\}^n$ be a corrupted codeword that is δ_{radius}-close to a codeword $C(x)$, where δ_{radius} is a sufficiently small constant, to be determined later. We partition the analysis into three cases (Claim 5.8.1, Claim 5.8.2, and Lemma 5.9) that we analyze in the rest of this section. We begin with the following two simple claims.

The first claim shows that probability $\Omega(1)$, the random copy $\mathbf{c}$ in $(c_1, \ldots, c_{t_1})$ that is chosen in Step 1 cannot be "too far" from the codeword $C(x)$.

Claim 5.8.1. *With probability at least* $1/4$*, the random copy* $\mathbf{c}$ *is* $4\delta_{C'}(w)$*-close to* $C(x)$*, where* $\mathbf{c}$ *is chosen uniformly at random from* $\bar{c}$*. That is,*

$$\Pr_{\mathbf{c} \in_R (c_1, \ldots, c_{t_1})} [\delta_C(\mathbf{c}) \leq 4\delta_{C'}(w)] \geq \frac{1}{4}.$$

Proof. Since $|\bar{c}| = \left|\bar{p}^{\mathsf{lines}}\right| = \left|\bar{p}^{\mathsf{planes}}\right|$, then $\bar{c} = (c_1, \ldots, c_{t_1})$ is $3\delta_{C'}(w)$-close to $C(x)^{t_1}$. This means that the expected relative distance of a random $\mathbf{c} \in \{c_1, \ldots, c_{t_1}\}$ from $C(x)$ is at most $3\delta_{C'}(w)$. Hence, by Markov's inequality, $\mathbf{c}$ is $4\delta_{C'}(w)$-far from $C(x)$ with probability at most $3/4$. □

Therefore, throughout the rest of the proof we fix a random copy $\mathbf{c}$ and assume that it is $4\delta_{C'}(w)$-close to $C(x)$. This only costs us at most a constant factor in the success probability of the decoder. Having fixed $\mathbf{c}$, recall that for $\bar{\imath} \in [n]^d$, the notation $\mathbf{c}_{\bar{\imath}}$ refers

The relaxed-LDC Procedure for C'

Input: a coordinate $\bar{\imath} \in [k]^d$ and an oracle access to a string $w = (\bar{c}, \bar{p}^{\mathsf{lines}}, \bar{p}^{\mathsf{planes}})$.

For $s \in [n]$ and $b \in \{0,1\}$ let $V_{s,b}$ be a scPCPP verifier that refers to an input of the form $z \in \{0,1\}^n$, and asserts that there exists $y \in C_0$ such that $z = y$ and $z_s = b$.

1. Choose a random copy of a tensor code $\mathbf{c}$ in $\bar{c}$ and a random copy of a set of point-line proofs $\bar{p}$ in $\bar{p}^{\mathsf{lines}}$. That is, choose uniformly at random $\mathsf{r} \in [t_1]$ and $\mathsf{r}' \in [t_2]$, and set $\mathbf{c} \triangleq c_{\mathsf{r}}$ and $\bar{p} \triangleq \{p^{j,\bar{\imath}}\}_{\bar{\imath} \in [n]^d,\, j \in [d]} \triangleq \bar{p}^{\mathsf{lines}}_{\mathsf{r}'}$.
2. Initialize a set of points P_1 to contain the singleton $\bar{\imath}$; i.e., $P_1 = \{\bar{\imath}\}$.
3. For $j = 1$ until $j = d$:
   (a) Select uniformly at random a point $\bar{u} = (u_1, \ldots, u_d)$ from the set P_j.
   (b) Verify that the j^{th}-axis-parallel line passing through $\bar{u}$ is a legal codeword of C_0 and that it is consistent with the value at $\mathbf{c}_{\bar{u}}$. That is, run the scPCPP verifier $V_{s,\mathbf{c}_{\bar{u}}}$, where $s \triangleq u_j$, with proof oracle $p^{j,\bar{u}}$ and input that consists of the j^{th}-axis-parallel line passing through $\bar{u}$ in $\mathbf{c}$. In other words, we run $V_{s,\mathbf{c}_{\bar{u}}}$ on input $\mathbf{c}|_{\ell_{j,\bar{u}}}$ and proof $p^{j,\bar{u}}$.
   (c) If V rejects, output $\perp$ and halt.
   (d) If $j < d$, fix P_{j+1} to be a set of points in $[n]^d$ that reside on the j^{th}-axis-parallel lines passing through the points in P_j. That is, $P_{j+1} = \{\ell_{j,\bar{z}}\}_{\bar{z} \in P_j}$, where $\ell_{j,\bar{z}}$ (defined in Definition 5.5) is the j^{th} axis-parallel line passing through the point $\bar{z}$.
4. Query $\mathbf{c}_{\bar{\imath}}$ and return its value.

Figure 5.1: Relaxed local decoder D for C'

to the value of $\mathbf{c}$ at point $\bar{\imath}$. The next claim shows that if the bit we are trying to decode is not "corrupted" (in the random copy $\mathbf{c}$), then the decoder D never outputs a mistake.

Claim 5.8.2. *If $\mathbf{c}_{\bar{\imath}} = x_{\bar{\imath}}$, then $\Pr_D[D^w(\bar{\imath}) \in \{x_{\bar{\imath}}, \perp\}] = 1$.*

Proof. By the definition of the decoder (see Figure 5.1), regardless of the rest of the values in the input, D always outputs either $\mathbf{c}_{\bar{\imath}}$ or $\perp$. □

The main part of the analysis takes place in the next lemma, where we assume that $\mathbf{c}_{\bar{\imath}} \neq x_{\bar{\imath}}$ and $\mathbf{c}$ is close to $C(x)$, and prove that the decoder succeeds with constant probability, as required. Recall that $\delta_{C'}(w) < \delta_{\mathsf{radius}}$, where δ_{radius} is a sufficiently small constant, to be determined later.

Lemma 5.9. *Suppose that* $\mathbf{c}$ *is* $4\delta_{C'}(w)$*-close to* $C(x)$ *and that* $\mathbf{c}_{\bar{\imath}} \neq x_{\bar{\imath}}$*. Then,*

$$\Pr_{D}[D^w(\bar{\imath}) \in \{x_{\bar{\imath}}, \perp\}] = \Omega(1).$$

Proof. We say that a point $\bar{u} \in [n]^d$ in the tensor code $\mathbf{c}$ is corrupted if $\mathbf{c}_{\bar{u}} \neq C(x)_{\bar{u}}$. Since we assume that $\mathbf{c}$ is corrupted in the point $\bar{\imath}$ (which we wish to decode), by the definition of the decoder, the probability that D makes a mistake is equal to the probability that D reaches Step 4 and outputs $\mathbf{c}_{\bar{\imath}}$.

Recall that P_j is the set of points that we consider in the j^{th} iteration of the decoder. The set P_1 is the singleton that contains $\bar{\imath}$; i.e., $P_1 = \{\bar{\imath}\}$ and for every $j \in \{2, \ldots, d+1\}$ we recursively define P_j as the set of all points that reside on the $(j-1)$-axis-parallel lines that pass through points in P_{j-1} (see Step 3d). Note that for every $j \in [d]$ the cardinality of P_j is equal to the number of points in a codeword of $C_0^{\otimes j-1}$; that is, $|P_j| = n^{j-1}$. Hence, the number of points in all lines that pass through points in P_j (i.e., n^j) equals the number of points in a codeword of $C_0^{\otimes j}$. We will show that in order to corrupt $\mathbf{c}_{\bar{\imath}}$ without being detected by the scPCPPs, one has to corrupt a constant fraction of a large portion of the lines that pass through points in P_d, which in turn implies that one has to corrupt a constant fraction of the tensor code C, in contradiction to our assumption that $\delta_{C'}(w) < \delta_{\mathsf{radius}}$, for a sufficiently small constant δ_{radius}.

Consider the first iteration of Step 3 (where $j = 1$). Denote by $s \triangleq i_1$ the index of the bit that we wish to decode in the line $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$, and denote by $b \triangleq \mathbf{c}_{\bar{\imath}}$ the value of $\mathbf{c}$ at $\bar{\imath}$.

We verify that the line that passes through $\bar{\imath}$ in the 1-direction is a codeword of C_0 that is consistent with the value of $\mathbf{c}$ at $\bar{\imath}$. This is done by running the verifier $V_{s,b}$ on input $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$ and proof $p^{1,\bar{\imath}}$. Recall that the relative distance of C_0 (i.e., $\delta(C_0)$) is a constant. Since $\bar{\imath}$ is corrupted (i.e., $b = \mathbf{c}_{\bar{\imath}} \neq C(x)_{\bar{\imath}}$), if the line $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$ is $\delta(C_0)/2$-close to the line $C(x)|_{\ell_{1,\bar{\imath}}}$ (which is a codeword of C_0 that is *inconsistent* with $\mathbf{c}_{\bar{\imath}}$), then $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$ is $\delta(C_0)/2$-far from any codeword $y \in C_0$ that is *consistent* with $\mathbf{c}_{\bar{\imath}}$ (i.e., such that $y_s \neq C(x)_{\bar{\imath}}$). In this case, the verifier $V_{s,b}$ rejects $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$ with probability at least $\mathsf{poly}(\delta(C_0)/2) = \Omega(1)$ (regardless of the corresponding proof), as required. Hence, in the following we assume that the line $\mathbf{c}|_{\ell_{1,\bar{\imath}}}$ is $\delta(C_0)/2$-far from $C(x)|_{\ell_{1,\bar{\imath}}}$, and therefore P_2 contains a constant fraction of at least $\beta_1 \triangleq \delta(C_0)/2$ corrupted points.

We proceed by induction. Consider the j^{th} iteration, where $2 \leq j \leq d$. We show that if the set of points that we consider in the j^{th} iteration (the set P_j) contains a constant fraction of corrupted points, then either the decoder rejects with constant probability in the j^{th} iteration, or P_{j+1} contains a constant fraction of corrupted points (we denote this probability by β_{j+1}).

Claim 5.9.1. *Let* $2 \leq j \leq d$ *and let* $0 < \beta_j \leq 1$ *be a constant. If* P_j *contains a at least a* β_j *fraction of corrupted points, then either:*

1. *The decoder rejects with probability at least* $\Omega(1)$ *in the* j^{th} *iteration; or,*
2. P_{j+1} *contains at least* $\beta_{j+1} \triangleq \frac{\beta_j \cdot \delta(C_0)}{4}$ *fraction of corrupted points.*

Proof of Claim 5.9.1. Consider the j^{th} iteration of Step 3. The decoder selects uniformly at random a point $\bar{u} = (u_1, \ldots, u_d) \in P_j$. Denote by $s = u_j$ the index of the bit that we wish to decode on the line $\mathbf{c}|_{\ell_{j,\bar{u}}}$ (which passes through $\bar{u}$ in the j^{th}-direction), and denote by $b \triangleq \mathbf{c}_{\bar{u}}$ the value of $\mathbf{c}$ at $\bar{u}$. By the hypothesis, $\bar{u}$ is corrupted with probability at least β_j.

Next, the verifier $V_{s,b}$ is executed on input $\mathbf{c}|_{\ell_{j,\bar{u}}}$ and proof $p^{j,\bar{u}}$. Observe that if a fraction of *at most* $\beta_j/2$ of the j-axis-parallel lines that pass through points in P_j (i.e., $\left\{\mathbf{c}|_{\ell_{j,\bar{z}}}\right\}_{\bar{z}\in P_j}$) are $\delta(C_0)/2$-far (each) from their corresponding lines in $C(x)$, then the decoder outputs $\perp$ with probability at least $\beta_j/2 \cdot \mathsf{poly}\left(\delta(C_0)/2\right) = \Omega(1)$, as required. This is because in this case, with probability at least $\beta_j/2$, we hit a line that is $\delta(C_0)/2$-close to its corresponding line in $C(x)$ (but the value of this line in u_j differs from $C(x)_{\bar{u}}$). As in the first iteration, this implies that this line is $\delta(C_0)/2$-far from any codeword $y \in C_0$ such that $y_s \neq C(x)_{\bar{u}}$, and hence the verifier $V_{s,b}$ rejects $\mathbf{c}|_{\ell_{j,\bar{u}}}$ with probability at least $\mathsf{poly}\left(\delta(C_0)/2\right)$ (regardless of the corresponding proof).

Otherwise (i.e., if the above case does not hold), *at least* $\beta_j/2$ of the lines in $\left\{\mathbf{c}|_{\ell_{j,\bar{z}}}\right\}_{\bar{z}\in P_j}$ are $\delta(C_0)/2$-far (each) from their corresponding lines in $C(x)$. Therefore, P_{j+1} contains at least a $\frac{\beta_j \cdot \delta(C_0)}{4}$ fraction of corrupted points. □

Note that P_{d+1} is the set of all points in $[n]^d$. By solving the recurrence relation, we get that $\beta_{d+1} = \frac{\delta(C_0)^d}{2^{2d-1}}$.[16] Recall that according to the hypothesis of the lemma, $\mathbf{c}$ is $4\delta_{\mathsf{radius}}$-close to $C(x)$. Fix the decoding radius δ_{radius} to a sufficiently small constant such that $4\delta_{\mathsf{radius}} < \beta_{d+1}$. Thus, Claim 5.9.1 implies that in one of the iterations the decoder must reject with probability at least $\Omega(1)$, as required. □

Remarks. The codewords of C' are of the form $w = (\bar{c}, \bar{p}^{\mathsf{lines}}, \bar{p}^{\mathsf{planes}})$, where the three parts are of equal length. The fact that the length of each of the three parts is proportional to the others is critical. The length of $\bar{c}$ must be proportional to the length of w in order for our code to have constant relative distance (recall that there is no guarantee on the distance of the scPCPPs). Moreover, the length of each of the scPCPP parts, $\bar{c}$ and $\bar{p}^{\mathsf{lines}}$, should be proportional to the length of w in order to obtain the average smoothness requirement (see Section 5.4.3).

We remark that we chose our tensor code to be *systematic* only for the sake of convenience. Instead, we could have added the message itself (repeated to obtain the proper length) as a fourth part to the code C'.[17].

Next, we note that for the proof that our code C' is a relaxed-LDC we only use the point-line scPCPPs and ignore the plane scPCPPs (i.e., the third part of w). Furthermore, we do not use the fact that the point-line scPCPPs are neither *strong* nor *canonical*. That is, to get only a relaxed-LDC with nearly-linear length it is enough to augment a good

[16] Recall that the fraction of corrupted points in P_2 is at least $\delta(C_0)/2$, and that for $2 \leq j \leq d$ the fraction of corrupted points in P_{j+1} (which we denote by β_{j+1}) is at least $\frac{\beta_j \cdot \delta(C_0)}{4}$.

[17] Actually, this approach (of adding the message itself to the output of the code) was taken in previous constructions of relaxed-LDC (see [BSGH+06, GR13c]). By using a systematic tensor code, we circumvented this unnecessary complication.

systematic tensor code (i.e., a tensor product of a systematic linear code with constant rate and constant relative distance) with a "regular" PCPP. However, the plane scPCPPs and the *strongness* and *canonicity* of the PCPPs will be heavily used in the proof that C' is also a strong-LTC (see Section 5.5).

5.4.3 Obtaining Average Smoothness

In this subsection, we conclude the proof that $C' : \{0,1\}^{k'} \to \{0,1\}^{n'}$ is a relaxed-LDC. Recall that in Section 5.4.2 we showed a decoder D for C' (described in Figure 5.1) that satisfies the first two conditions of Definition 5.8, i.e., the *completeness* and *(modified) relaxed soundness* conditions. Next, we show that D can be modified such that it also satisfies the third and final condition of Definition 5.8, i.e., the *average smoothness* condition (which, roughly speaking, requires that the decoder makes nearly-uniform queries on average).

Denote by $\mathcal{D}^w(i,j,r)$ the j^{th} query of the decoder D on coordinate $i \in [k']$, coin tosses r, and input oracle w. Recall that D satisfies the *average smoothness* condition if for every $w \in \{0,1\}^{n'}$ and $v \in [n']$, it holds that

$$\Pr_{i,j,r}\left[\mathcal{D}^w(i,j,r) = v\right] < \frac{2}{n'}, \tag{5.2}$$

where the probability is taken uniformly over all possible choices of $i \in [k']$, $j \in [q]$ (where q is the number of queries that D makes), and coin tosses r.

Firstly, we can relax the condition in Equation (5.2) and replace it with the condition

$$\Pr_{i,j,r}\left[\mathcal{D}^w(i,j,r) = v\right] = O\left(\frac{1}{n'}\right). \tag{5.3}$$

To see this, note that if the decoder D (which makes $q = O(1)$ queries) satisfies Equation (5.3), then we can obtain a decoder D' that makes $q' = O(q)$ queries and satisfy Equation (5.2) simply by running D and adding $O(q)$ uniformly distributed "dummy" queries (whose answers the decoder ignores).

Secondly, note that by the construction of D (of Figure 5.1), each of the scPCPPs verifiers that are being emulated by D makes nearly-uniform queries (see Theorems 5.5 and 5.6) to the statement it refers to and to its corresponding proof. Observe that on a random index $\bar{u} \in [k]^d$ the decoder D invokes the verifier of the *point-line* scPCPP on uniformly selected lines in a uniformly selected copy of the tensor code. Since the length of the first and second part of each codeword of C' (i.e., the tensor code and the *point-line* scPCPPs) constitutes a constant fraction of the length of each codeword of C', the decoder D satisfies Equation (5.3). Finally, by the foregoing discussion, D can be modified to satisfy Equation (5.2).

5.5 Establishing the Strong-LTC Property

In this section we prove that the code C', which was defined in Section 5.3, is a strong locally testable code.

Theorem 5.9. *The code $C' : \{0,1\}^{k'} \to \{0,1\}^{n'}$ as defined in Section 5.3 is a* strong-LTC. *Furthermore, it has a tester that makes nearly-uniform queries.*

In order to prove Theorem 5.9 we need to present a tester T that is given an oracle access to $w \in \{0,1\}^{n'}$, makes $O(1)$ queries to w, and satisfies the following: For all $w \in C$ it holds that $T^w = 1$, and for all $w \notin C$ it holds that $\Pr_T[T^w = 0] \geq \mathsf{poly}\left(\delta_{C'}(w)\right)$.

5.5.1 Outline of the Tester and its Analysis

Recall that each codeword of C' consists of three parts: (1) an alleged d-dimensional tensor code $C = C_0^{\otimes d} : \{0,1\}^{k^d} \to \{0,1\}^{n^d}$, (2) alleged scPCPPs for every 2-dimensional plane in C; each scPCPP ascertains that the given plane is consistent with C, and (3) alleged scPCPPs for every pair of point $\bar{\imath}$ in C and line ℓ in C that passes through $\bar{\imath}$; each scPCPP ascertains that a line ℓ is a codeword of C_0 that is consistent with the value at a point $\bar{\imath}$.

For the simplicity of the exposition, we omit the repetitions of the three parts of the code (i.e., the tensor code, the *point-line* scPCPPs, and the *plane* scPCPP*s*) and assume instead that the length of the each part is equal. We note that this assumption can be easily removed by using an additional consistency test. See the full details in Section 5.5.2.

The key idea is that by the *robustness* property of tensor codes, the corruption rate of a codeword is proportional to the corruption rate of a random plane in the codeword. Hence, in order to ensure that the tensor code part of C' is valid, our tester use the *plane* scPCPPs to ascertain that a random plane is close to being valid. We note that for the tester, we do not need the *point-line* scPCPPs (which we only need for the decoder); however, since we need to ensure that also the point-line scPCPPs part is not corrupted, our tester also verifies a random *point-line* scPCPPs.

Clearly, this tester always accepts valid codewords. To analyze what happens with non-codewords consider a string that is somewhat far from C'. In this case, one of the following three cases must hold:

1. The tensor code part is far from a legal codeword of $C^{\otimes d}$.

2. The tensor code part is close to a legal codeword of $C^{\otimes d}$ but the *plane* scPCPP proofs part is far from the corresponding canonical proofs.

3. The tensor code part is close to a legal codeword of $C^{\otimes d}$ but the *point-line* scPCPP proofs part is far from the corresponding canonical proofs.

To ensure that in the first case the tester succeeds (i.e., rejects with sufficiently high probability), it is enough to test that a random plane in $\mathbf{c}$ is close to a codeword of $C^{\otimes 2}$. To accomplish this, we choose uniformly at random a (2-dimensional, axis-parallel) plane and run the corresponding *plane* scPCPP verifier. This suffices, since Theorem 5.4 asserts that if a tensor $\mathbf{c}$ is far from a legal codeword of $C^{\otimes d}$, then a random (2-dimensional, axis-parallel) plane in $\mathbf{c}$ must also be far from a legal codeword of $C^{\otimes 2}$.

The second and third cases are similar, and so, we only sketch how to handle the second case. Assume that the tensor is close to a codeword but the *plane* scPCPPs are far from the corresponding canonical proofs. From this assumption we can deduce that there are many planes that are close to legal codewords of $C^{\otimes 2}$, but whose corresponding scPCPPs are far from the canonical proofs. Thus, choosing a random plane and running the corresponding *plane* scPCPP verifier ensures that the tester rejects with a sufficiently high probability. This is due to the *strongness* and *canonicity* features of our scPCPPs.

To conclude, the tester consists of three parts: (1) a repetition test, wherein we verify the repetition structure of the tensor, (2) *plane* scPCPP consistency test, wherein we verify that a random plane in the tensor is a legal codeword; this test ensures that both the tensor code part consists of valid codewords and its *plane* scPCPPs are the corresponding canonical proofs, and (3) *point-line* scPCPP consistency test, which we perform only to verify that the *point-line* scPCPPs consists of the canonical proofs that corresponds to the tensor part of the code.

5.5.2 The Full Proof

We proceed with the full proof of Theorem 5.9, which formalizes the intuition given in the previous section. We show a strong-LTC procedure for C'. The tester T is formally described in Figure 5.2. Note that since both the *point-line* and *plane* scPCPP verifiers make nearly-uniform queries (and the three parts of each codeword are of equal length), then the tester T also makes nearly-uniform queries.

Consider an arbitrary input $w \in \{0,1\}^{n'}$ such that $\delta_{C'}(w) \geq 0$. We view w as a string composed of three parts as in Section 5.3, i.e., $w = (\bar{c}, \bar{p}^{\mathsf{lines}}, \bar{p}^{\mathsf{planes}})$. The *completeness* of the tester is immediate: Indeed, if the input is a codeword, i.e., $w = C'(x)$, then the first part of w consists of identical copies of a tensor code, and hence the codeword repetition test accepts with probability 1. Similarly, the second and third parts consists of the canonical *point-line* and *plane* scPCPP proofs for the aforementioned tensor code, respectively; hence the (one-sided error) scPCPP verifiers will accept with probability 1.

Next, we prove the *soundness* of the tester. We partition the analysis into three cases (Claim 5.9.2 and Lemmas 5.10 and 5.11), which we analyze in the rest of this section.

Let $\hat{c} \in \{0,1\}^{n^d}$ be a tensor that is closest on average to the tensors in $\bar{c}$, i.e., a string that minimizes $\Delta(\bar{c}, \hat{c}^{t_1}) = \sum_{i=1}^{t_1} \Delta(c_i, \hat{c})$. The first (and standard) claim shows that if $\bar{c}$ is far from consisting of t_1 identical tensors, then the repetition test (of Step 1) rejects with high probability. Let γ be a constant set to $\delta(C)/(24d)$ (for the purpose of Lemma 5.11).

Claim 5.9.2. *If* $\delta(\bar{c}, \hat{c}^{t_1}) \geq \frac{\gamma}{5} \cdot \delta_{C'}(w)$, *then* $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.

Proof. Suppose that $\delta(\bar{c}, \hat{c}^{t_1}) \geq \frac{\gamma}{5} \cdot \delta_{C'}(w)$. The codeword repetition test rejects with

The strong-LTC Procedure for C'

Input: oracle access to a string $w = (\bar{c}, \bar{p}^{\mathsf{lines}}, \bar{p}^{\mathsf{planes}})$.

For $s \in [n]$ and $b \in \{0,1\}$ let $V^{\mathsf{line}}(s,b)$ be a scPCPP verifier that refers to an input of the form $z \in \{0,1\}^n$ and asserts that there exists $y \in C_0$ such that $z = y$ and $z_s = b$.

Let V^{plane} be a scPCPP verifier that refers to an input of the form $z \in \{0,1\}^{n^2}$ and asserts that there exists $y \in C_0^{\otimes 2}$ such that $z = y$.

Choose a random copy of each of the three replicated parts of w. That is, choose uniformly at random a copy $\mathbf{c}$ in $\bar{c}$, a copy $\bar{p}_{\mathsf{line}} = \{p^{j,\bar{\imath}}\}_{\{\bar{\imath}\in[n]^d,\, j\in[d]\}}$ in $\bar{p}^{\mathsf{lines}}$, and a copy $\bar{p}_{\mathsf{plane}} = \{p^{\mathsf{p}}\}_{\{\mathsf{p}\in\mathsf{Planes}\}}$ in $\bar{p}^{\mathsf{planes}}$.

Accept if none of the following tests reject:

1. **The repetition test:** We query two random copies from the tensor part of w and check if they agree on a random location. More accurately, we select uniformly at random $\mathsf{r}, \mathsf{r}' \in [t_1]$ and reject if and only if c_{r} and $c_{\mathsf{r}'}$ disagree on a *random* coordinate.
2. **The *plane* scPCPP consistency test:** Choose a uniformly at random a plane $\mathsf{p} \in \mathsf{Planes}$. Reject if the verifier V^{plane} rejects on the plane p (i.e., input $\mathbf{c}|_{\mathsf{p}}$) and the proof p^{p}.
3. **The *point-line* scPCPP consistency test:** Choose uniformly at random a coordinate $\bar{u} = (u_1, \ldots, u_d) \in [n]^d$ and a direction $j \in [d]$ in $\mathbf{c}$. Reject if the verifier $V^{\mathsf{line}}(u_j, \mathbf{c}_{\bar{u}})$ rejects on the line passing through $\bar{u}$ in direction j and the proof $p^{j,\bar{u}}$. In other words, we reject if $V^{\mathsf{line}}(u_j, \mathbf{c}_{\bar{u}})$ rejects on input $\mathbf{c}|_{\ell_{j,\bar{u}}}$ and proof $p^{j,\bar{u}}$.

Figure 5.2: Strong local tester for C'

probability at least

$$\mathop{\mathbf{E}}_{\mathsf{r},\mathsf{r}'\in_R[t_1]}\left[\frac{\Delta(c_{\mathsf{r}}, c_{\mathsf{r}'})}{n^d}\right] \geq \mathop{\mathbf{E}}_{\mathsf{r}\in_R[t_1]}\left[\frac{\Delta(c_{\mathsf{r}}, \hat{c})}{n^d}\right] = \frac{\Delta(\bar{c}, \hat{c}^{t_1})}{t_1 n^d}.$$

Therefore, $\Pr_T[T^w = 0] \geq \frac{\gamma}{5} \cdot \delta_{C'}(w) \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$. □

The following lemma shows that if $\bar{c}$ consists of t_1 nearly identical tensors that are far from a codeword of C, then due to the *robustness* feature of tensor codes, a random plane in a random copy in $\bar{c}$ will be far from valid, and hence, Step 2 of the tester rejects with high probability.

Lemma 5.10. *Assume $\delta(\bar{c}, \hat{c}^{t_1}) < \frac{\gamma}{5} \cdot \delta_{C'}(w)$. If $\bar{c}$ is $\gamma \cdot \delta_{C'}(w)$-far from C^{t_1}, then* $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.

Proof. Observe that a random copy $\mathbf{c}$ of a tensor code in $\bar{c}$ is $\Omega\big(\delta_{C'}(w)\big)$-far from C with high probability. This is because $\delta_{C^{t_1}}(\bar{c}) \leq \delta_{C^{t_1}}(\hat{c}^{t_1}) + \delta(\hat{c}^{t_1}, \bar{c})$, which implies $\delta_C(\hat{c}) > \frac{4\gamma}{5} \cdot \delta_{C'}(w)$. Since at least $2/3$ of the c_i's are $3 \cdot \frac{\gamma}{5} \cdot \delta_{C'}(w)$-close to $\hat{c}$, these c_i's are $\frac{\gamma}{5} \cdot \delta_{C'}(w)$-far from C.

Next, by the *robustness* feature of tensor codes, we deduce that if the randomly selected tensor code $\mathbf{c}$ is $\Omega\big(\delta_{C'}(w)\big)$-far from being valid, then a random plane of $\mathbf{c}$ is also $\Omega\big(\delta_{C'}(w)\big)$-far from being valid. Specifically, by Theorem 5.4, there exists a constant $c_{\mathsf{robust}} \in (0,1)$ such that for every tensor $w \in \{0,1\}^{n^d}$ we have

$$\mathop{\mathbf{E}}_{\mathsf{p} \in_R \mathsf{Planes}} \left[\delta\left(w|_{\mathsf{p}}, C^{\otimes 2}\right)\right] > c_{\mathsf{robust}} \cdot \delta_{C^{\otimes d}}(w).$$

Hence, by an averaging argument,

$$\Pr_{\mathsf{p} \in \mathsf{Planes}} \left[\delta_{C^{\otimes 2}}(c|_{\mathsf{p}}) > \frac{c_{\mathsf{robust}}}{2} \cdot \frac{\gamma}{5} \cdot \delta_{C'}(w)\right] > \frac{c_{\mathsf{robust}}}{2} \cdot \frac{\gamma}{5} \cdot \delta_{C'}(w). \tag{5.4}$$

Note that, by Equation (5.4), with probability $\Omega\big(\delta_{C'}(w)\big)$ we select a plane that is $\Omega\big(\delta_{C'}(w)\big)$-far from a codeword of $C_0^{\otimes 2}$. Given such plane, the scPCPP verifier V^{plane} rejects with probability $\Omega\big(\delta_{C'}(w)\big)$. Thus, the tester T rejects with probability $\mathsf{poly}(\delta_{C'}(w))$ over the internal randomness of T. □

In the next lemma, we complete the analysis by assuming that $\bar{c}$ is sufficiently close to a codeword of C^{t_1}, and showing that in this case most of the "corruption" takes place in the parts of the scPCPP proofs, and hence the scPCPP consistency tests will reject with high probability.

Lemma 5.11. *If $\bar{c}$ is $\gamma \cdot \delta_{C'}(w)$-close to being a codeword of C^{t_1}, then $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.*

Proof. Recall that $\gamma = \frac{\delta(C)}{24d} < \frac{\delta(C)}{2}$. Therefore, our assumption that $\bar{c}$ is $\gamma \cdot \delta_{C'}(w)$-close to being a codeword of C^{t_1} implies that there exists a *unique* codeword c' of C that minimizes the distance of $\bar{c}' \triangleq (c')^{t_1}$ from $\bar{c}$. Let w' be the codeword of C' that consists of repetitions of the tensor code c' and its canonical scPCPP proofs; that is, Let $w' = \left(\bar{c}', (\pi_{\mathsf{lines}}(c'))^{t_2}, (\pi_{\mathsf{planes}}(c'))^{t_3}\right)$ be a codeword of C'. Denote by x the inverse of w' (i.e., , $w' = C'(x)$).

It is convenient to introduce notations for the fraction of corruptions in each part of C'. Towards this end, denote the fraction of errors in the first part of the code (the copies of the tensor code) by $\delta_{\bar{c}} = \delta(\bar{c}, \bar{c}')$. Analogously, denote by $\delta_{\bar{p}^{\mathsf{lines}}}$ and $\delta_{\bar{p}^{\mathsf{planes}}}$ the fraction of errors in the second and third parts of w (*point-line* scPCPPs and *plane* scPCPPs), respectively. Denote by $\delta_{\bar{p}^{\mathsf{total}}} = \left(\delta_{\bar{p}^{\mathsf{lines}}} + \delta_{\bar{p}^{\mathsf{planes}}}\right)/2$ the total fraction of errors in the second and third part of w together.

Observe that assuming the hypothesis of Lemma 5.11 (i.e., $\bar{c}$ is sufficiently close to $\bar{c}'$), the scPCPPs part (i.e., $\bar{p}^{\mathsf{lines}}$ and $\bar{p}^{\mathsf{planes}}$) must be somewhat far from the corresponding set of canonical scPCPP proofs; that is, assuming $\delta_{\bar{c}} < \delta_{C'}(w)$, then $\delta_{\bar{p}^{\mathsf{total}}} \geq \delta_{C'}(w)$.

Therefore, since $\delta_{\bar{c}} \leq \gamma \cdot \delta_{C'}(w) < \delta_{C'}(w)$, we may assume that either: (1) the *plane* scPCPPs are sufficiently corrupted, i.e., $\delta_{\bar{p}^{\mathsf{planes}}} > \delta_{C'}(w)$, or (2) the *point-line* scPCPPs are sufficiently corrupted, i.e., $\delta_{\bar{p}^{\mathsf{lines}}} > \delta_{C'}(w)$. We claim that in the first case the *plane* scPCPP consistency test will reject with high probability, and in the second case the *point-line* scPCPP consistency test will reject with high probability. We prove this in the following two claims, from which Lemma 5.11 follows.

Claim 5.11.1. *Assuming $\bar{c}$ is $\gamma\cdot\delta_{C'}(w)$-close to being a codeword of C^{t_1}, if $\delta_{\bar{p}^{\mathsf{planes}}} > \delta_{C'}(w)$, then* $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.

Claim 5.11.2. *Assuming $\bar{c}$ is $\gamma\cdot\delta_{C'}(w)$-close to being a codeword of C^{t_1}, if $\delta_{\bar{p}^{\mathsf{lines}}} > \delta_{C'}(w)$, then* $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.

Claim 5.11.1 and Claim 5.11.2 follow immediately from the *canonicity* and *strong soundness* features of the scPCPPs (along with averaging arguments). Since the proofs of Claim 5.11.1 andClaim 5.11.2 are similar, we conclude the proof of Lemma 5.11 by showing Claim 5.11.1 and defer the proof of Claim 5.11.2 to Section 5.8.5.

Proof of Claim 5.11.1. Loosely speaking, the hypothesis of the claim guarantees that: (1) $\bar{c}$ is close to being a *unique* codeword $C(x)^{t_1}$, and hence (by averaging arguments), most restrictions of a random copy $\mathbf{c}$ in $\bar{c} = (c_1, \ldots, c_{t_1})$ to a plane cannot be significantly corrupted; (2) the *plane* scPCPPs are far, on average, from the canonical proofs that corresponds to $C(x)$, and thus many *plane* scPCPPs are far from the canonical proofs for the planes of $C(x)$ they correspond to. By the foregoing, we conclude that there are many planes in $\mathbf{c}$ that are close to planes of $C(x)$ but their alleged *plane* scPCPP proofs are far from their canonical proofs. Thus, by the *canonicity* and *strong soundness* features of the scPCPPs, the verifier will reject with high probability. Details follow.

By the claim's hypothesis, $\bar{c}$ is $\delta_{\bar{c}}$-close to $C(x)^{t_1}$, where $\delta_{\bar{c}} \leq \gamma \cdot \delta_{C'}(w)$. Hence, by an averaging argument, with probability at least $2/3$ the random copy $\mathbf{c}$ is $3\delta_{\bar{c}}$-close to $C(x)$. Assume from now on that this is indeed the case. We say that a point $\bar{\imath} \in [n]^d$ in $\mathbf{c}$ is corrupted if $\mathbf{c}_{\bar{\imath}} \neq C(x)_{\bar{\imath}}$, and so, there are at most $3\delta_{\bar{c}} n^d$ corrupted points in $\mathbf{c}$. Since there are $\binom{d}{2} n^{d-2}$ axis-parallel planes in $\mathbf{c}$, then on average, the number of corrupted points in a random axis-parallel plane in $\mathbf{c}$ is at most $\frac{3\delta_{\bar{c}} n^d}{\binom{d}{2} n^{d-2}} < 3\delta_{\bar{c}} n^2$. Thus, by an averaging argument, we obtain that at most $\frac{\delta_{\bar{p}}}{4}$ fraction of the axis-parallel planes in $\mathbf{c}$ contain at least $\frac{4}{\delta_{\bar{p}}} \cdot 3\delta_{\bar{c}} n^2$ corrupted points.

Secondly, we note that a random copy of the *plane* scPCPP proofs contains a fraction of $\Omega\big(\delta_{C'}(w)\big)$ corrupted points with probability $\Omega\big(\delta_{C'}(w)\big)$. That is, by an averaging argument, with probability at least $\delta_{\bar{p}} \triangleq \delta_{\bar{p}^{\mathsf{planes}}}/2$ the random copy $\bar{p}$ in $\bar{p}^{\mathsf{planes}}$ is $\delta_{\bar{p}}$-far from its corresponding set of canonical proofs, $\pi_{\mathsf{planes}}(x) = \{\pi_{\mathsf{plane}}(C(x)|_{\mathsf{p}})\}_{\mathsf{p}\in\mathsf{Planes}}$. Assume from now on that $\bar{p}$ is $\delta_{\bar{p}}$-far from $\pi_{\mathsf{planes}}(x)$. Then, by an averaging argument, we obtain that at least $\delta_{\bar{p}}/2$ fraction of the proofs in $\bar{p} = \{p^{\mathsf{p}}\}_{\mathsf{p}\in\mathsf{Planes}}$ are $\delta_{\bar{p}}/2$-far from their corresponding (canonical) proofs $\pi_{\mathsf{planes}}(x)$.

By combining the conclusions of the last two paragraphs, we deduce that $\Omega\big(\delta_{C'}(w)\big)$-fraction of the planes p in $\mathbf{c}$ are both $\delta(C_0^{\otimes 2})/2$-close to the restriction of the tensor

codeword $C(x)$ to p, and their corresponding proofs are $\Omega\big(\delta_{C'}(w)\big)$-corrupted; that is, a fraction of at least $\frac{\delta_{\bar{p}}}{4}$ of the axis-parallel planes p in $\mathbf{c}$ are $\delta(C_0^{\otimes 2})/2$-close to $C(x)|_{\mathsf{p}}$ (recall that $\frac{4}{\delta_{\bar{p}}} \cdot 3\delta_{\bar{c}} < 12\gamma < \delta(C)/2 \leq \delta(C_0^{\otimes 2})$), and in addition, their corresponding (alleged) *plane* scPCPP proofs in $\{p^{\mathsf{p}}\}_{\mathsf{p}\in\mathsf{Planes}}$ are $\delta_{\bar{p}}/2$-far from their (correct) canonical proofs in $\pi_{\mathsf{planes}}(x)$. Denote the set of planes that satisfy the foregoing condition by BAD.

Observe that for every plane $\mathsf{p} \in$ BAD, in order for input $\mathbf{c}|_{\mathsf{p}}$ and proof p^{p} to be a valid claim (for the input-proof language that V^{plane} verifies), one must make at least one of the following changes: (1) change a fraction of at least $\frac{\delta_{\bar{p}}}{2}$ of the proof p^{p} such that it matches $\pi_{\mathsf{plane}}\left(C(x)|_{\mathsf{p}}\right)$, or (2) change a fraction of at least $\delta(C_0^{\otimes 2})/2$ of $\mathbf{c}|_{\mathsf{p}}$ (since p^{p} might be a valid proof for input $C_0^{\otimes 2}(y) \neq \mathbf{c}|_{\mathsf{p}}$). Thus, for every $\mathsf{p} \in$ BAD, the probability that V^{plane} rejects input $c|_{\mathsf{p}}$ and proof p^{p} is at least polynomial in $\delta_{C'}(w)$.

Putting it all together, with probability $2/3$ we hit a random copy $\mathbf{c}$ of the tensor code that is $3\delta_{\bar{c}}$-close to $C(x)$. Furthermore, with probability at least $\delta_{\bar{p}}$ we hit a random copy $\bar{p}$ that is $\delta_{\bar{p}}$-corrupted, and subsequently, with probability $\delta_{\bar{p}}/2$ we hit a *plane* scPCPP proof that is $\delta_{\bar{p}}/2$-corrupted. Finally, assuming the foregoing, the scPCPP verifier V^{plane} rejects with probability $\mathsf{poly}\left(\delta_{C'}(w)\right)$. Therefore,

$$\Pr_T[T^w = 0] \geq \frac{2}{3} \cdot \delta_{\bar{p}} \cdot \frac{\delta_{\bar{p}}}{2} \cdot \mathsf{poly}\left(\delta_{C'}(w)\right) \geq \mathsf{poly}\big(\delta_{C'}(w)\big).$$

□

This concludes the proof of Lemma 5.11. □

5.6 Strong Canonical PCPs of Proximity

In this section we construct scPCPPs with *polynomial* proof length for any good linear code (see Theorem 5.5) and for any *half-space* of a any good linear code (see Theorem 5.6). Our starting point (see Corollary 5.11) is the following result of [GR13c],[18] which in turn builds upon [GS06, Section 5.2]: For any good code $C : \{0,1\}^k \to \{0,1\}^{ck}$, there exists a strong-LTC $C' : \{0,1\}^k \to \{0,1\}^{\mathsf{poly}(k)}$ such that the first half of $C'(x)$ consists of c blocks, each depending only on a k-bit long block of $C(x)$. Using this result, we construct a scPCPP for any good code C, where this construction applies the above result to several auxiliary codes that are derived from C.

5.6.1 scPCPPs for Good Codes

The main technical tool upon which we rely (when proving Theorem 5.5) is the *linear inner proof systems* (hereafter, LIPS), constructed by Goldreich and Sudan. Loosely speaking, the LIPS mechanism allows to transform linear strong locally testable codes over a large alphabet into strong locally testable codes over a smaller alphabet (see [GS06, Section 5.2]). We encapsulate our usage of the LIPS mechanism in the following theorem,

[18] Actually, Corollary 5.11 is a straightforward generalization of [GR13c, Corollary B.3].

which generalizes [GS06, Theorem 5.20] and [GS06, Proposition 5.21]. Throughout this section, denote $\mathbb{F} = \mathsf{GF}(2)$.

Theorem 5.10. *Let $\Sigma = \mathbb{F}^b$. For infinitely many k, there exists $n = \mathsf{poly}(k)$ and a linear code $E : \Sigma \to \mathbb{F}^n$ with constant relative distance such that the following holds. Suppose that $C : \Sigma^K \to \Sigma^N$ is a* strong-LTC *that is linear over $\mathbb{F}$ and has a (non-adaptive) tester that uses r random bits and makes nearly-uniform queries. Then, there exists $\ell = \mathsf{poly}(k)$ such that ℓ is a multiple of n, and a linear* strong-LTC *$C'' : \mathbb{F}^{bk} \to \mathbb{F}^{2^{r+1} \cdot \ell}$ such that the $2^r \cdot \ell$-bit long prefix of $C''(x)$ equals $\big(E(C(x)_1), \ldots, E(C(x)_N)\big)^{2^r \ell/(Nn)}$. Moreover, the tester of C'' makes nearly-uniform queries.*

As a corollary of Theorem 5.10, we obtain that any good linear code can be augmented to a linear strong-LTC with *polynomial* length, such that the prefix of the new code is closely related to that of the original code (but is *not* equal to the original code). This is done by viewing the good linear code as a trivial strong-LTC over a sufficiently large alphabet.

Corollary 5.11 (our starting point)**.** *Let $C : \{0,1\}^k \to \{0,1\}^{ck}$ be a good linear code with constant relative distance, where $c \in \mathbb{N}$ is a constant. Then, for some $M, m = \mathsf{poly}(k)$, there exists a linear* strong-LTC *$C' : \{0,1\}^k \to \{0,1\}^{2M}$ and a linear code $E : \{0,1\}^k \to \{0,1\}^m$, which has constant relative distance, such that the M-bit long prefix of $C'(x)$ equals $\big(E(C(x)[1]), \ldots, E(C(x)[c])\big)^{M/(c \cdot m)}$, where $C(x)[i]$ is the i^{th} block of length k in $C(x)$. Furthermore, the (strong) tester of C' makes nearly-uniform queries.*

We remark that Theorem 5.10 and Corollary 5.11 are straightforward generalization of [GR13c, Theorem B.2] and [GR13c, Corollary B.3] (respectively), and we defer their proofs to Section 5.8.1.

The Plan. Let $C : \{0,1\}^k \to \{0,1\}^{ck}$ be a good linear code, where $c \in \mathbb{N}$ is a constant. We construct a strong-LTC C' such that a constant fraction of each codeword $C'(x)$ contains copies of $C(x)$. This, in turn, implies a scPCPP for C (see Proposition 5.14). Note that by applying Corollary 5.11 to C we obtain a strong-LTC C' such that a constant fraction of each codeword $C'(x)$ contains copies of $\big(E(C(x)[1]), \ldots, E(C(x)[c])\big)$, but not of $C(x)$. This does not seem to suffice for obtaining a scPCPP, and so we use a different approach.

We start by using Corollary 5.11 to obtain a family of linear strong-LTCs $\big\{C_i : \{0,1\}^k \to \{0,1\}^n\big\}_{i \in [ck]}$ where $n = \mathsf{poly}(k)$, with constant relative distance such that the prefix of each codeword $C_i(x)$ contains a linear number of copies of the i^{th}-bit of $C(x)$ (as well as other structural features that will be useful for us). This is done via the next lemma, which uses techniques from [GR13c].

Lemma 5.12 (obtaining auxiliary codes C_i)**.** *Let $C : \{0,1\}^k \to \{0,1\}^{ck}$ be a good linear code, where $c \in \mathbb{N}$ is a constant. There exist a constant $\alpha \in (0,1)$, a polynomial value $n = \mathsf{poly}(k)$, and a linear code $\hat{C} : \{0,1\}^k \to \{0,1\}^{cn}$ with constant relative distance, which*

satisfy the following: For every $i \in [ck]$, there exists a function $\pi_i : \{0,1\}^k \to \{0,1\}^{(c+1)n}$ such that the code $C_i : \{0,1\}^k \to \{0,1\}^{\alpha n+cn+(c+1)n}$, given by

$$C_i(x) = \big((C(x)_i)^{\alpha n}, \hat{C}(x), \pi_i(x)\big),$$

is a linear strong-LTC *with constant relative distance. Moreover, for every $i \in [ck]$ the (strong) tester of C_i makes nearly-uniform queries.*

We stress that the code $\hat{C}$ (which is common to all C_i's) is independent of i and constitutes a constant fraction of the length of each C_i.

Proof of Lemma 5.12. For every $j \in [c]$, we denote by $C(x)[j]$ the j^{th} block of length k of $C(x)$. For every $i \in [ck]$, consider the code $C'_i : \{0,1\}^k \to \{0,1\}^{(c+1)k}$ given by

$$C'_i(x) \triangleq \big((C(x)_i)^k, C(x)\big) = \big((C(x)_i)^k, C(x)[1], \ldots, C(x)[c]\big).$$

Note that C'_i is a good linear code.

For every $i \in [ck]$, we apply Corollary 5.11 to C'_i and obtain a linear strong-LTC $C''_i : \{0,1\}^k \to \{0,1\}^{2(c+1)\cdot n}$ with constant relative distance, which is (up to a permutation of its bit locations) of the form

$$C''_i(x) = \left(\left(E\big((C(x)_i)^k\big)\right)^t, \left(E\big(C(x)[1]\big)\right)^t, \ldots, \left(E\big(C(x)[c]\big)\right)^t, \pi_i(x)\right)$$

where $m, n = \mathsf{poly}(k)$, the function $E : \{0,1\}^k \to \{0,1\}^m$ is a linear code with constant relative distance, $t = n/m$, and $\pi_i(x) \in \{0,1\}^{(c+1)n}$ is some string. Moreover, the (strong) tester of C''_i makes nearly-uniform queries.

Denote by $\hat{C} : \{0,1\}^k \to \{0,1\}^{cn}$ the linear code (with constant relative distance) that is given by $\hat{C}(x) = \left(\left(E\big(C(x)[1]\big)\right)^t, \ldots, \left(E\big(C(x)[c]\big)\right)^t\right)$. Since E is a linear code with constant relative distance, then $E(0^k) = 0^m$ and $\Delta\big(E\left(1^k\right), 0^m\big) \geq \alpha m$ for some constant $\alpha \in (0,1)$. Now, for every $i \in [ck]$, consider the code $C_i : \{0,1\}^k \to \{0,1\}^{\alpha n+cn+(c+1)n}$, given by $C_i(x) = \big((C(x)_i)^{\alpha n}, \hat{C}(x), \pi_i(x)\big)$, which is obtained from C''_i by simply removing coordinates on which $E(0^k)$ and $E(1^k)$ agree, in each of the t copies in the first part (i.e., $E\big(C(x)_i\big)^k$).

Note that C_i has constant relative distance. Furthermore, since C''_i is linear and since we only removed coordinates on which the value is 0, the code C_i is also a linear code. Finally, by emulating the execution of the tester of C''_i on an (alleged) codeword of C_i (which can be done by returning 0 whenever a coordinate that was omitted is being queried), we obtain that $C_i(x)$, which is of the required form of the hypothesis, is a strong-LTC with a (strong) tester that makes nearly-uniform queries. □

In the actual proof of Theorem 5.5, we will construct a code C' that encodes a message x by concatenating the encodings of x by all of the strong-LTCs in $\big\{C_i : \{0,1\}^k \to \{0,1\}^n\big\}_{i\in[ck]}$ (i.e., $C'(x) \triangleq \big(C_1(x), \ldots, C_{ck}(x)\big)$). Thus, we will obtain a strong-LTC that (up to a permutation of the bit locations) contains copies of the entire codeword $C(x)$ in its prefix. We

remark that, in general, the concatenation of strong-LTCs is *not* a strong-LTC. However, the structure of the aforementioned family of codes (specifically, the fact that all codes in the family contains a *common* sub-code) implies that the concatenation of codes in $\left\{C_i : \{0,1\}^k \to \{0,1\}^n\right\}_{i\in[ck]}$ yields a strong-LTC. The next proposition shows a sufficient condition for obtaining strong-LTCs via concatenation of strong-LTCs.

Proposition 5.13 (concatenating multiple encodings of strong-LTCs with a common sub-code)**.** *Let $C_1,\ldots,C_t : \{0,1\}^k \to \{0,1\}^n$ be strong-LTCs with constant relative distance. Let $I \subseteq [n]$ such that $|I| = \Omega(n)$, and let $\widehat{C} : \{0,1\}^k \to \{0,1\}^{|I|}$ be a code with constant relative distance. If $\widehat{C}(x) = C_1(x)|_I = C_2(x)|_I = \ldots = C_t(x)|_I$ for every $x \in \{0,1\}^k$, where $C_i(x)|_I$ denotes the restriction of $C_i(x)$ to I, then the code $C'(x) \triangleq \big(C_1(x),\ldots,C_t(x)\big)$ is a* strong-LTC *with constant relative distance. Moreover, if the (strong) testers of $C_1,\ldots,C_t$ make nearly-uniform queries, then the (strong) tester of C' also makes nearly-uniform queries*

Proposition 5.13 follows by using a tester that (1) emulates the strong-LTC tester of a randomly selected concatenated code C_i (to ascertain that each concatenated codeword is valid), and (2) tests the consistency of the common code $\hat{C}$ in two randomly selected concatenated codes (to assure that all of the concatenated codewords encode the same message). The analysis is quite straightforward and is deferred to Section 5.8.2.

The last tool we shall need in order to prove Theorem 5.5 is the following proposition, which allows us to transform strong-LTCs to scPCPPs for prefixes of the strong-LTCs' codewords.

Proposition 5.14 (from strong-LTCs to scPCPPs for related codewords)**.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code, and let $C' : \{0,1\}^k \to \{0,1\}^{n'}$ be a linear* strong-LTC*. If there exists $I \subseteq [n']$ where $|I| = \Omega(n')$ and $n' - |I| = \Omega(n')$ such that $C'(x)|_I = \big(C(x)\big)^{|I|/n}$, then there exists a* scPCPP *for C (i.e., for the set of codewords $\{C(x)\}_{x\in\{0,1\}^k}$) with proof length $O(n')$. Moreover, the canonical* scPCPP *proofs are linear, and if the (strong) tester of C' makes nearly-uniform queries, then the verifier of the* scPCPP *for C also makes nearly-uniform queries.*

Proof. Let C, C', and I be as in the hypothesis. Assume, without loss of generality, that $I = \{1,\ldots,|I|\}$. Denote the (strong) tester of C' by T. We use T in a black-box manner in order to construct a scPCPP for the set $\{C(x)\}_{x\in\{0,1\}^k}$.

Given a codeword $C(x)$, the canonical scPCPP proof for $C(x)$ is given by $\pi(x) \triangleq C'(x)|_{[n']\setminus I}$, where $C'(x)|_{[n']\setminus I}$ is the restriction of $C(x)$ to the coordinates outside of I. Let V be the scPCPP verifier that gets oracle access to an alleged codeword $w \in \{0,1\}^n$ and oracle access to a proof oracle p of length $n' - |I|$. Let $t = |I|/n$. The verifier V emulates the execution of T on (w^t, p) as follows: Each query that T makes to the first part (which are allegedly $C(x)^t$) is simulated by a corresponding query to the input oracle w,[19] and each query that T makes to the other coordinates (which is allegedly $\pi(x)$) is

[19]Note that the tester expects t copies of $C(x)$, while the input oracle consists of a single copy. Hence, the emulation is done simply by directing the query of the i^{th} bit of the j^{th} copy to the i^{th} bit of the input oracle, for every i, j.

simulated by a corresponding query to the proof oracle. The verifier V accepts if and only if the emulated run of T on (w^t, p) accepted. Note that if T makes nearly-uniform queries, then V also makes nearly-uniform queries.

The *completeness* of V is immediate: If w is a codeword $C(x)$ and $p = \pi(x)$, then $\big(w^t, p\big)$ is a codeword of C'. We conclude the proof by showing the *soundness* of V. Note that V gets as input a pair of an alleged codeword w and an alleged canonical proof p. Suppose that $\delta_{\mathsf{PCPP}}(w,p) \triangleq \min_{x\in\{0,1\}^n}\left\{\max\left(\delta(x,w)\ ;\ \delta(\pi_{\mathsf{canonical}}(x),p)\right)\right\} > 0$.

For every $x \in \{0,1\}^n$, either the alleged proof p is $\delta_{\mathsf{PCPP}}(w,p)$-far from $\pi_{\mathsf{canonical}}(x)$, or the alleged codeword is $\delta_{\mathsf{PCPP}}(w,p)$-far from $C(x)$. In the former case, since $|p| = n' - |I| = \Omega(n')$, it holds that $\delta\big((w^t,p),(x^t,\pi_{\mathsf{canonical}}(x))\big) = \Omega\big(\delta_{\mathsf{PCPP}}(w,p)\big)$. In the latter case, since $\delta_{C^t}(w^t) = \delta_C(w)$ and $|w^t| = \Omega(n')$, it holds that $\delta\big((w^t,p),(x^t,\pi_{\mathsf{canonical}}(x))\big) = \Omega\big(\delta_{\mathsf{PCPP}}(w,p)\big)$. Therefore $\delta_{C'}\big((w^t,p)\big) = \Omega\big(\delta_{\mathsf{PCPP}}(w,p)\big)$, and thus the tester of C', and subsequently the verifier V, will reject with probability $\mathsf{poly}(\delta_{\mathsf{PCPP}}(w,p))$ as required. □

Using Lemma 5.12 and Propositions 5.13 and 5.14, we proceed with the proof of Theorem 5.5.

Proof of Theorem 5.5. Let $c \in \mathbb{N}$ be a constant and $C : \{0,1\}^k \to \{0,1\}^{ck}$ be a linear code with constant relative distance. We show a scPCPP, with polynomial proof length, for the language of all codewords of C.

First, we apply Lemma 5.12 on C and get that there exists a linear code $\hat{C} : \{0,1\}^k \to \{0,1\}^{cn}$ with constant relative distance and a set of codes $\big\{C_i : \{0,1\}^k \to \{0,1\}^{\alpha n+cn+(c+1)n}\big\}_{\{i\in[ck]\}}$ such that each C_i is a linear code with constant relative distance that is given by

$$C_i(x) = \big((C(x)_i)^{\alpha n}, \hat{C}(x), \pi_i(x)\big),$$

where $\alpha \in (0,1)$, $n = \mathsf{poly}(k)$ and $\pi_i : \{0,1\}^k \to \{0,1\}^{(c+1)n}$. Moreover, each C_i makes nearly-uniform queries.

Next, we consider the code $C'(x) \triangleq \big(C_1(x), \ldots, C_{ck}(x)\big)$. Observe that, up to a permutation of the indices, C' has the form

$$C'(x) = \big(C(x)^{\alpha n}, \hat{C}(x)^{ck}, \pi(x)\big),$$

where $\pi(x) = \pi_1(x), \ldots, \pi_{ck}(x)$. Note that $\left|\hat{C}(x)^{ck}\right| = ck \cdot cn$, which is a constant fraction of $|C'(x)|$. By Proposition 5.13, the code C' is a strong-LTC with constant relative distance that makes nearly-uniform queries.

Finally, the theorem follows by applying Proposition 5.14 to the code C' with $I = [\alpha n \cdot ck]$, where the code C is repeated $\alpha n = |I|/(ck)$ times. (Indeed, we use the fact that $|I|$ is a constant fraction of $|C'(x)|$.) Note that the scPCPP proof we obtain (namely, $\big(\hat{C}(x)^{ck}, \pi(x)\big)$) is of length $\mathsf{poly}(k)$. □

5.6.2 scPCPPs for Half-Spaces of Good Codes

Theorem 5.6 is obtained by using Theorem 5.5 in a black-box manner. Specifically, note that in case $b = 0$, the code $C_{i,0}$ is linear, and thus we can apply Theorem 5.5 directly.

On the other hand, in case $b = 1$, the code $C_{i,1}$ is *not* linear, but we can "shift" it (by a fixed codeword of $C_{i,1}$) and apply Theorem 5.5.

Proof of Theorem 5.6. In light of the above, we focus on the case in which $b = 1$. Assume, without loss of generality, that there exists a codeword $c^{(i)}$ of C such that that the i^{th}-bit of $c^{(i)}$ is 1 (otherwise, we can always reject). Consider a verifier, $V_{i,1}$, that gets oracle access to an input string w and a proof π, and proceeds as follows. The verifier $V_{i,1}$ emulates the execution of $V_{i,0}$ (obtained via Theorem 5.5) on input oracle $w + c^{(i)}$ (where the summation is point-wise over $\mathsf{GF}(2)$) and its proof oracle π (which should be the canonical proof for $w + c^{(i)} \in C_{i,0}$). Note that the verifier $V_{i,0}$ makes nearly-uniform queries, and so $V_{i,1}$ also makes nearly-uniform queries. We show that $V_{i,1}$ is a scPCPP for $C_{i,1}$

The *completeness* is immediate: Recall that if w is a codeword of $C_{i,1}$, then $w = C(x)$ such that $w_i = 1$. By the linearity of C, $w + c^{(i)}$ is a codeword of C such that its i^{th} bit is 0 (i.e., $\left(w + c^{(i)}\right)_i = 0$). Therefore, we actually invoke $V_{i,0}$ on a codeword of $C_{i,0}$. For the *soundness* condition, assume that $\delta_{C_{i,1}}(w) > 0$. Observe that

$$\delta_{C_{i,0}}\left(w + c^{(i)}\right) = \min_{w' \in C_{i,0}} \delta\left(w', w + c^{(i)}\right) = \min_{w' \in C_{i,0}} \delta\left(w' + c^{(i)}, w\right) = \delta_{C_{i,1}}(w).$$

Therefore, the verifier $V_{i,1}$ will reject the input $w + c^{(i)}$ (given the corresponding canonical proof) with probability at least $\mathsf{poly}\left(\delta_{C_{i,1}}(w)\right)$, as required. □

5.7 Application to Property Testing

In this section we give an application of our main result (Theorem 5.1) to the area of *property testing*. Specifically, we improve on the best known separation result, due to Gur and Rothblum [GR13c], between the complexity of *decision* versus *verification* in the property testing model. Details follow.

The study of property testing, initiated by Rubinfeld and Sudan [RS96] and Goldreich, Goldwasser and Ron [GGR98], considers highly-efficient randomized algorithms that solve approximate decision problems, while only inspecting a small fraction of the input. Such algorithms, commonly referred to as *testers*, are given oracle access to some object, and are required to determine whether the object has some predetermined property or is far (say, in Hamming distance) from every object that has the property.

Remarkably, it turns out that many natural properties can be tested by making relatively few queries to the object. However, there are also many natural properties that *no* tester can test efficiently. In fact, "almost all" properties require a very large query complexity to be tested. Motivated by this limitation, Gur and Rothblum [GR13c] initiated the study of MA proofs of proximity (hereafter MAPs), which can be viewed as the NP proof-system analogue of property testing.

Loosely speaking, an MAP is a probabilistic proof system that augments the property testing framework by allowing the tester full and free access to an (alleged) proof. That is, such a proof-aided tester for a property Π is given *oracle* access to an input x and

free access to a proof string w, and should distinguish between the case that $x \in \Pi$ and the case that x is far from Π, while only making a sublinear number of queries. More precisely, given a *proximity parameter* $\varepsilon > 0$, we require that for inputs $x \in \Pi$, there exist a proof that the tester accepts with high probability, and for inputs x that are ε-far from Π no proof will make the tester accept, except with some small probability of error. For formal definitions we refer to [GR13c, Section 2].

As observed by [GR13c], given an MAP proof of length that is linear in the size of the object (specifically, a proof that fully describes the object), every property can be tested by only making $O(1/\varepsilon)$ queries to the object, simply by verifying the proof's consistency with the object. Hence, it is natural to measure the complexity of an MAP by both the length of the proof and the number of queries made in order to decide whether $x \in \Pi$ or ε-far from it. We note that a property tester can be viewed as an MAP that uses a proof of length 0.

Gur and Rothblum [GR13c] showed that the task of separating the power of property testers and MAPs can be reduced to the task of designing a code that is both locally testable and locally decodable. Furthermore, they noticed that for such a separation, *relaxed decodability* suffices. Unable to construct a code as in Theorem 5.1, Gur and Rothblum used several weaker codes to obtain partial separation results. Specifically, they proved the following theorem.

Theorem 5.12 (Theorems 3.1, 3.2 and 3.3 in [GR13c])**.** *In all items, n denotes the length of the main input being tested.*

1. *For every constant $\alpha > 0$, there exists a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(1/\varepsilon)$ queries for every $\varepsilon > 1/\mathsf{polylog}(n)$, but for which every* property tester *must make $\Omega(n^{1-\alpha})$ queries.*

2. *For every constant $\alpha > 0$, there exists a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(\log n, 1/\varepsilon)$ queries, but for which every* property tester *must make $\Omega(n^{1-\alpha})$ queries.*

3. *There exists a universal constant $c \in (0,1)$ and a property Π that has an* MAP *that uses a proof of length $O(\log n)$ and makes $\mathsf{poly}(1/\varepsilon)$ queries (without limitation on ε), but for which every* property tester *must make n^c queries.*

Furthermore, each of the above MAP*s has one-sided error.*

Note that each of these separation results has a drawback: The first separation works only for sufficiently large values of the proximity parameter, the second separation has non-constant query complexity for the MAPs, and the third separation does not require property testers to make nearly-linear number of queries.

Plugging in the code C' from Theorem 5.1 into the framework developed by [GR13c, Lemmas 3.4 and 3.5], we achieve the best of all the aforementioned results; that is, a separation for all values of the proximity parameter, with constant query complexity for the MAPs, and nearly-linear query complexity for testers. Formally, we obtain the following separation result between MAPs and property testers.

Theorem 5.13 (Restated)**.** *For every constant $\alpha > 0$, there a property Π_α that has an* MAP *that uses a proof of length $O(\log n)$ and makes* poly$(1/\varepsilon)$ *queries (without limitation on ε), but for which every* property tester *must make $n^{1-\alpha}$ queries. Furthermore, the* MAP *has one-sided error.*

Acknowledgments

We would like to thank Or Meir and Madhu Sudan for helpful discussions regarding the robustness of tensor codes and its relation to local testability, and Michael Ben-Or for raising the issue of tolerant testing. The third author would like to thank his advisor Moni Naor for his support and encouragement.

5.8 Appendices for Chapter 5

5.8.1 Obtaining Strong LTCs from LIPS

In this appendix, we provide tools that allow us to use the *linear inner proof systems* (hereafter, LIPS), constructed by Goldreich and Sudan [GS06], to obtain families of strong-LTCs with several features that we take advantage of in Section 5.6. Specifically, we prove Theorem 5.10 and Corollary 5.11. Throughout this section, denote $\mathbb{F} = \mathsf{GF}(2)$. Recall the statement of Theorem 5.10.

Theorem 5.14 (restated)**.** *Let $\Sigma = \mathbb{F}^b$. For infinitely many k, there exists $n = \mathsf{poly}(k)$ and a linear code $E : \Sigma \to \mathbb{F}^n$ such that the following holds. Suppose that $C : \Sigma^K \to \Sigma^N$ is a* strong-LTC *that is linear over $\mathbb{F}$ and has a (non-adaptive) tester that uses r random bits and makes nearly-uniform queries. Then, there exists $\ell = \mathsf{poly}(k)$ such that ℓ is a multiple of n, and a linear* strong-LTC *$C'' : \mathbb{F}^{bk} \to \mathbb{F}^{2^{r+1}\cdot\ell}$ such that the $2^r \cdot \ell$-bit long prefix of $C''(x)$ equals $\big(E(C(x)_1), \ldots, E(C(x)_N)\big)^{2^r \ell/Nn}$. Moreover, the tester of C'' makes nearly-uniform queries.*

Proof. We follow the proof of [GS06, Theorem 5.20], while using the code C of the theorem's hypothesis instead of the third ingredient in that proof. In addition, following [GS06, Proposition 5.21], we use composition theorems (i.e., [GS06, Theorem 5.15] and [GS06, Theorem 5.17]) that preserve the nearly-uniform distribution of the queries the verifiers (or tester) make, thus ascertaining that $C''(x)$ has a tester that queries each location with probability $\Theta(1/N)$. We note that in our settings, the overhead of replacing the "vanilla" composition theorems (which are used in [GS06, Theorem 5.20]) with the composition theorems that preserve the nearly-uniform queries is insignificant. Details follow.

In the following description, all references refer to [GS06]. Recall some basics regarding the terminology used in [GS06]. By Definitions 5.8 and 5.9, a $\big(\mathbb{F}, (q, b) \to (p, a), \delta, \gamma\big)$-LIPS refers to input oracles $X_1, ..., X_q : [n] \to \mathbb{F}^a$ and a proof oracle $X_{q+1} : [\ell] \to \mathbb{F}^a$, where the input oracles provide an n-long encoding (over $\mathbb{F}^a$) of a single symbol in the (much) bigger alphabet $\mathbb{F}^b$ (i.e., this encoding is denoted $E : \mathbb{F}^b \to (\mathbb{F}^a)^n$). (In addition δ is the relative distance of the encoding used, and γ is the detection ratio in strong soundness. In the following, both parameters will be small constants.)

The proof of Theorem 5.20 starts with an overview (page 79), and then lists three ingredients (page 80) that will be used: (1) The Hadamard based $\big(\mathbb{F}, (p_H, k_H) \to (p_H + 5, 1), 1/2, 1/8\big)$-LIPS (for any choice of p_H and k_H) of Proposition 5.18, (2) The Reed-Muller based $\big(\mathbb{F}, (p_{RM}, k_{RM}) \to (p_{RM} + 4, \mathsf{poly}(\log p_{RM} k_{RM})), 1/2, \Omega(1)\big)$-LIPS (for any choice of p_{RM} and k_{RM}) of Proposition 5.18, and (3) a specific strong-LTC (namely, the strong-LTC in Part 1 of Theorem 2.4). We shall use the very same first two ingredients,[20] but use the code C in place of the third. Assume, without loss of generality, that the randomness complexity r of the strong (tester) of C satisfies that 2^r is

[20] We remark that while these two LIPSs are presented in [GS06] as if they are non-uniform, it can be verified that they can be presented in uniform terms (i.e., computable by Turing machines rather than by circuits).

a multiple of N. (We remark that all three ingredients have verifiers or testers that make nearly-uniform queries, and that we compose these ingredients via the composition theorems that preserve this distribution of queries.) Specifically, the second paragraph following the ingredients-list asserts that for any desired p'' and k'', an $\big(\mathbb{F},(p'',k'')\to(p''+13,1),\Omega(1),\Omega(1/p'')^2\big)$-LIPS with randomness $O(p''\log k'')$, input length $\mathsf{poly}(p''k'')$, and proof length that are $\mathsf{poly}(p''k'')$. We shall use $p''=O(1)$ and $k''=b$, where the $O(1)$ stands for the query complexity of the codeword tester for C. Thus the above simplifies to asserting an $\big(\mathbb{F},(O(1),b)\to(O(1),1),\Omega(1),\Omega(1)\big)$-LIPS with randomness $O(\log b)$ and input/proof lengths (i.e., n and ℓ) that are $\mathsf{poly}(b)$. Without loss of generality, we may assume that ℓ is a multiple of n.

Next, we wish to compose C with the above LIPS via Theorem 5.15 (instead of via Theorem 5.13, which does not preserve the nearly-uniform distribution of the queries). It follows that in Item 1 of Theorem 5.15 we use K,N and r as provided by the hypothesis and $q=O(1)$. For Item 2, we use b as provided by the hypothesis, ($q=O(1)$ as above), $p=O(1)$ and $a=1$, and $n,\ell=\mathsf{poly}(b)$ (all fitting the LIPS above). So we have $\Gamma=F$, and get a strong-LTC mapping $\mathbb{F}^{bK}$ to $\mathbb{F}^{2^{r+1}\cdot\ell}$, which makes nearly-uniform queries. In particular, for $t=2^r\ell/Nn$ (i.e., $tNn=2^r\ell$), as shown on top of page 56 (see Equation (32)), the first half of the codewords of the resulting code have the form $\big(E(C(x)_1)),...,E(C(x)_N)\big)^t$, where $x\in\mathbb{F}^{bK}$ is viewed as an element of Σ^K. The theorem follows. □

Next, recall the statement of Corollary 5.11.

Corollary 5.15 (restated)**.** *Let $C:\{0,1\}^k\to\{0,1\}^{ck}$ be a good linear code with constant relative distance, where $c\in\mathbb{N}$ is a constant. Then, for some $M,m=\mathsf{poly}(k)$, there exists a linear* strong-LTC *$C':\{0,1\}^k\to\{0,1\}^{2M}$ and a linear code $E:\{0,1\}^k\to\{0,1\}^m$, which has constant relative distance, such that the M-bit long prefix of $C'(x)$ equals $\big(E(C(x)[1]),...,E(C(x)[c])\big)^{M/cm}$, where $C(x)[i]$ is the i^{th} block of length k in $C(x)$. Furthermore, the (strong) tester of C' makes nearly-uniform queries.*

Proof. Let $C:\mathbb{F}^k\to\mathbb{F}^{ck}$ be a good linear code. Viewing C as a mapping from $\Sigma=\mathbb{F}^k$ to Σ^c, note that C is a strong-LTC, which is (trivially) checked by reading all c symbols (and hence, by definition, it makes uniform queries). The claim follows by instantiating Theorem 5.10 using the code C and taking $b=k$, $K=1$, $N=c=O(1)$, and $r=0$. □

5.8.2 Concatenating Multiple Encodings of Strong LTCs

In this appendix, we show a sufficient condition for obtaining strong-LTCs via concatenation of strong-LTCs. We prove Proposition 5.13.

Proof. Let $|I|=\alpha\cdot n$ for constant $0\le\alpha\le 1$. Assume, without loss of generality, that $I=\{1,\ldots,\alpha\cdot n\}$. For every $i\in[t]$, we refer to an alleged (n-bit) codeword $C_i(x)$ as the pair of strings $(y_i,z_i)\in\{0,1\}^{\alpha\cdot n}\times\{0,1\}^{(1-\alpha)\cdot n}$, so that y_i is the common codeword $\widehat{C}(x)$ and z_i is the rest of the codeword.

We show a (strong) tester that, given oracle access to a binary string $w = \big((y_1, z_1), \ldots, (y_t, z_t)\big)$, where $(y_i, z_i) \in \{0,1\}^n$ for every $i \in [t]$, accepts every codeword of C' and rejects non-codewords of C' with probability that is polynomial in their relative distance from C'. The strong-LTC procedure for C' is described in Figure 5.3.

The strong-LTC Procedure for C'

Input: a string $\big((y_1, z_1), \ldots, (y_t, z_t)\big) \in \{0,1\}^{n \cdot t}$.

1. **The inner strong-LTC test:** Select at random $i \in [t]$, and run the strong-LTC tester of C_i on (y_i, z_i).
2. **The common codeword consistency test:** Select at random $i_1, i_2 \in [t]$ and $j \in [n]$, and reject if the j^{th} bit of y_{i_1} and y_{i_2} differs.

Figure 5.3: Strong local tester for C'

Note that Step 1 of the tester T invokes the tester of a uniformly selected inner code (C_i), and so, if the testers of $C_1, \ldots, C_t$ make nearly-uniform queries, then Step 1 of T also makes nearly-uniform queries. As for Step 2 of T (which queries a uniformly selected bit in two uniformly selected y_i's), note that by adding two dummy queries to the second part of each inner code (i.e., query a uniformly selected bit in two uniformly selected z_i's) we ensure that the first test also makes nearly-uniform queries.

The *completeness* of the tester is straightforward. If $\big((y_1, z_1), \ldots, (y_t, z_t)\big)$ is equal to $C'(x)$ for some $x \in \{0,1\}^k$, then: (1) for every $i_1, i_2 \in [t]$ it holds that $y_{i_1} = y_{i_2}$, and (2) for every $i \in [t]$ it holds that (y_i, z_i) is equal to $C_i(x)$. Thus the tester accepts.

Next, we show the *soundness* of the tester. Let $w = \big((y_1, z_1), \ldots, (y_t, z_t)\big)$ be $\delta_{C'}(w)$-far from the code C', let $u \in \{0,1\}^n$ be a string that minimizes the value of $\Delta\big((y_1, \ldots, y_t), u^t\big)$, and let $\gamma = \delta(\widehat{C})/36$. Suppose that $(y_1, \ldots, y_t)$ is $\gamma \cdot \delta_{C'}(w)$-far from u^t. In this case, the "common codeword consistency test" rejects with probability

$$\mathop{\mathbf{E}}_{i_1, i_2 \in_R [t]} \left[\frac{\Delta(y_{i_1}, y_{i_2})}{n} \right] \geq \mathop{\mathbf{E}}_{i_1 \in_R [t]} \left[\frac{\Delta(y_{i_1}, u)}{n} \right] = \frac{\Delta\big((y_1, \ldots, y_t), u^t\big)}{n \cdot t} = \gamma \cdot \delta_{C'}(w).$$

Thus, in the sequel, we assume that $(y_1, \ldots, y_t)$ is $\gamma \cdot \delta_{C'}(w)$-close to u^t.

Suppose that u is $3\gamma \cdot \delta_{C'}(w)$-far from $\widehat{C}$. Since $(y_1, \ldots, y_t)$ is $\gamma \cdot \delta_{C'}(w)$-close to u^t, at least half of the y_i's must be $2\gamma \cdot \delta_{C'}(w)$-close to u, so these y_i's are $\gamma \cdot \delta_{C'}(w)$-far from $\hat{C}$. Thus, in the invocation of the strong-LTC test of a random C_i, with probability $1/2$, the test is invoked on a string (y_i, z_i) such that y_i is $\gamma \cdot \delta_{C'}(w)$-far from the codewords of $\widehat{C}$. Since $|I| = |y_i|$ the tester will reject with probability $\Omega(\delta_{C'}(w))$. Hence, in the sequel, we assume that u is $3\gamma \cdot \delta_{C'}(w)$-close to a codeword of $\widehat{C}$. Since we also assume that $(y_1, \ldots, y_t)$ is $\gamma \cdot \delta_{C'}(w)$-close to u^t, then by the triangle inequality, the string $(y_1, \ldots, y_t)$

is $4\gamma \cdot \delta_{C'}(w)$-close to a (unique, since $4\gamma < \delta(\widehat{C})/2$) codeword $\widehat{C}^t(x)$. Furthermore, by an averaging argument, at most $\delta_{C'}(w)/8$ fraction of the y_i's are $\delta(\widehat{C})/2$-far from $\widehat{C}(x)$.

Since $|\hat{C}(x)|^t = \alpha \cdot |C'(x)|$ for a constant $\alpha \in (0,1)$, and since $(y_1, \ldots, y_t)$ is $4\gamma \cdot \delta_{C'}(w)$-close to $\widehat{C}^t(x)$, then $(z_1, \ldots, z_t)$ is $\delta_{C'}(w)/2$-far from any $(\hat{z}_1, \ldots, \hat{z}_t) \in \{0,1\}^{(n-|I|)t}$ such that $\left(\widehat{C}^t(x), (\hat{z}_1, \ldots, \hat{z}_t)\right)$ is a codeword of C'. Thus, at least $\delta_{C'}(w)/4$ fraction of the z_i's are $\delta_{C'}(w)/4$-far from their corresponding $\hat{z}_i$'s. Hence, at least $\delta_{C'}(w)/8$ fraction of the (y_i, z_i) pairs satisfy (1) y_i is $\delta(\widehat{C})/2$-close to $\widehat{C}(x)$, and (2) z_i is $\delta_{C'}(w)/4$-far from $\hat{z}_i(x)$. Therefore, if we invoke the verifier of C_i on such (y_i, z_i), it will reject with probability $\Omega(\delta_{C'}(w))$. Therefore, the tester T rejects with probability $\mathsf{poly}\big(\delta_{C'}(w)\big)$, as required. □

5.8.3 Robustness of Tensor Codes

In this section we prove Theorem 5.4, which is implicit in [Vid12]. Specifically, in [Vid12, Theorem A.5] it is shown that for $d \geq 3$, if a codeword w of a d-dimensional tensor code $C^{\otimes d}$ is corrupted, then the corruption in a random hyperplane (i.e., a $d-1$-dimensional subplane) of w is proportional to the corruption in the entire (d-dimensional) tensor w. By applying this theorem recursively we obtain that for *constant* values of $d \geq 3$, the corruption in a random 2-dimensional plane of a corrupted codeword of $C^{\otimes d}$ is proportional to the corruption in the entire codeword.

We start by recalling the definition of robustness. Informally, we say that a tester is robust if for every word that is far from the code, the tester's view is far in expectation from any consistent view. This notion was defined for LTCs following an analogous definition for PCPs [BSGH+06].

Definition 5.15 (Robustness)**.** *Given a tester T for a code $C : \{0,1\}^k \to \{0,1\}^n$, for every word $w \in \{0,1\}^k$ we define*

$$\rho^T(w) = \mathop{\mathbf{E}}_{I}\left[\delta(w|_I, C|_I)\right],$$

where $w|_I$ denotes the local view of the tester after querying on coordinates given by I. We say that the tester T has robustness *ρ_C^T on the code C if for every $w \in \{0,1\}^k$ it holds that $\rho^T(w) \geq \rho_C^T \cdot \delta_C(w)$.*

Next, we consider the "hyperplane tester for tensor codes" of Ben-Sasson and Sudan [BS06]. Towards this end, we first provide a notation for hyperplanes. For every $j \in [d]$, and $b \in [n]$, we say that τ is a (j,b)-hyperplane in $\{0,1\}^{n^d}$ if

$$\tau = \{(i_1, \ldots, i_{j-1}, b, i_{j+1}, \ldots, i_d) \ : \text{ for all } t \in [d] \setminus \{j\} \text{ we have } i_t \in [n]\}.$$

We denote by $\mathsf{Hyperplanes} = \{(j,b)\text{-hyperplane}\}_{\{j\in[d], b\in[n]\}}$ the set of all hyperplanes in $\{0,1\}^{n^d}$, and denote the restriction of a tensor $w \in \{0,1\}^{n^d}$ to a hyperplane $\tau \in \mathsf{Hyperplanes}$ by $w|_\tau \in \{0,1\}^{n^{d-1}}$.

Definition 5.16 (Hyperplane Tester for Tensor Codes)**.** *Let C be a linear code, $d \geq 3$ an integer, and $w \in \{0,1\}^{n^d}$. The hyperplane tester for $C^{\otimes d}$ selects uniformly at random $\tau \in \mathsf{Hyperplanes}$, obtains $w|_\tau$ by querying all points on τ, and accepts if and only if $w|_\tau \in C^{\otimes d-1}$.*

Theorem 5.16 ([Vid12, Theorem A.5])**.** *Let C be a linear code and $d \geq 3$. Let T be the hyperplane tester for $C^{\otimes d}$. Then, $\rho^T_{C^{\otimes d}} \geq \frac{\delta(C)^d}{2d^2}$.*

We show that Theorem 5.4 follows by iterative applications of Theorem 5.16.

Proof of Theorem 5.4. Let C be a linear code and $d \geq 3$ a constant integer. Let $w \in \{0,1\}^{n^d}$ be a tensor. For every $3 \leq t \leq d$, let T_t be the hyperplane tester for $C^{\otimes t}$. Note that for every $3 \leq t \leq d$, the tester T_t queries a hyperplane that is allegedly a codeword of $C^{\otimes t-1}$; hence T_{t-1} can be composed with T_t; that is, we can run T_t on input w, during which T_t generates a local view $w|_I$ to be queried, and so, we can run T_{t-1} on the local view $w|_I$. (Note that the composed tester $T_3 \circ \ldots \circ T_d$ queries the restriction of the input w to a uniformly selected plane $\mathsf{p} \in \mathsf{Planes}$.) The robustness of the composed tester will hence be

$$\rho^{T_3 \circ \ldots \circ T_d}_{C^{\otimes d}} \geq \rho^{T_d}_{C^{\otimes d}} \cdot \rho^{T_{d-1}}_{C^{\otimes d-1}} \cdot \ldots \cdot \rho^{T_3}_{C^{\otimes 3}}.$$

By Theorem 5.16, for every $t \geq 3$ we have $\rho^{T_t}_{C^{\otimes t}} \geq \frac{\delta(C)^t}{2t^2}$. Thus, for constant $d \geq 3$ it holds that $c_{\mathsf{robust}} \triangleq \rho^{T_3 \circ \ldots \circ T_d}_{C^{\otimes d}}$ is a positive constant that depends only on $\delta(C)$ and d. □

5.8.4 Average Smoothness and Error Reduction for Relaxed LDCs

In this appendix, following [BSGH+06, Section 4.2], we show that the modified definition of relaxed-LDCs (see Definition 5.8) implies the standard definition of relaxed-LDCs (see Definition 6.3). Towards this end we need to show the following: (1) The soundness can be increased from $\Omega(1)$ (as in Condition 2 of Definition 5.8) to $2/3$ (as in Condition 2 of Definition 6.3), and (2) the *average smoothness* (i.e., Condition 3 of Definition 5.8) can be replaced with the *success rate* condition (i.e., Condition 3 of Definition 6.3). Both claims were shown in [BSGH+06]; we provide their proofs (adapted to our settings) for completeness.

We start by showing how to perform error-reduction for relaxed-LDC with soundness $\Omega(1)$. Recall that the decoder is required to successfully decode each valid codeword, and in addition, given a somewhat corrupted codeword the decoder is required to either decode successfully or abort with probability $\Omega(1)$. On the face of it, it may seem that standard error reduction cannot be applied (since we start with a large error probability). However, the error reduction can be simply performed by repeating the execution of the decoder, outputting a bit only if all invocations returned this bit, and aborting otherwise. We remark that the above may cause an increase in the number of indices on which the decoder aborts (with probability at least $2/3$). However, in the modified definition (i.e., Definition 5.8) there is no restriction on the success rate.

Proposition 5.17. *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a modified relaxed-*LDC*, according to Definition 5.8. Then, C has a modified relaxed-*LDC* decoder that also satisfies Condition 2 of Definition 6.3.*

Proof. Let C be a modified relaxed-LDC. Denote its decoder by D. There exists a constant $p > 0$ such that for every string w that is sufficiently close to a codeword of C it holds that $\Pr_D[D^w(i) = \{x_i, \perp\}] \geq p$. Consider a decoder D' that operates follows: D' executes the original decoder D (with fresh randomness) for r times, where r is a constant to be determined later. If all of the executions are consistent, i.e., there exists an $a \in \{0,1,\perp\}$ such that in every execution $D^w(i) = a$, then D' output a; otherwise, D' output $\perp$. (We remark that the distribution of queries of D' is identical to that of D, and thus D' also satisfies the *average smoothness* condition.)

Note that the new decoder D' satisfies Condition 1 of Definition 6.3 (the *completeness* condition). Moreover, D' satisfies Condition 2 of Definition 6.3: Indeed, given w that is sufficiently close to $C(x)$, the probability that D' errs is at most $p' = (1-p)^r$. Hence, by fixing $r = 2/p$ we get that $\Pr_{D'}[D'^w(i) = \{x_i, \perp\}] \geq 1 - p' \geq 2/3$, as needed. □

Finally, we show that the *average smoothness* condition (i.e., Condition 3 of Definition 5.8) can be replaced by the *success rate* condition (i.e., Condition 3 of Definition 6.3, which limits the number of indices upon which the decoder aborts (with probability at least 2/3)). The key idea is that a decoder that satisfies the completeness and soundness conditions (i.e., Conditions 1 and 2 of Definition 6.3) only aborts if the local view of the codeword that it queries contains a corrupted point. By the *average smoothness*, on average the decoder will only query a corrupted point with low probability. Thus, by an averaging argument, we can deduce that there is a small number of indices upon which the decoder might abort.

Proposition 5.18. *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a linear code, and let D be a constant-query decoder for C that satisfies Conditions 1 and 2 of Definition 6.3 as well as Condition 3 of Definition 5.8 (i.e., average smoothness). Then, C satisfies all three conditions of Definition 6.3.*

Proof. Let the code C and the decoder D be as in the hypothesis of the proposition. Denote the (constant) query complexity of D by q. According to Condition 1, for any $x \in \{0,1\}^k$ and every $i \in [k]$, it holds that $\Pr\left[D^{C(x)}(i) = x_i\right] = 1$. Considering any w that is δ-close to $C(x)$ (where $\delta \leq \delta_{\mathsf{radius}}$), the probability that given a *uniformly distributed* index $i \in [k]$ the decoder D queries a location on which w and $C(x)$ disagree is at most $q \cdot (2/n) \cdot \delta n = 2q\delta$. This is due to the fact that, for a uniformly distributed i, no position is queried with probability greater than $2/n$.

Let p_i^w denote the probability that on input i the decoder D queries a location on which w and $C(x)$ disagree. We have just established that $(1/k) \cdot \sum_{i=1}^k p_i^w \leq 2q\delta$. By an averaging argument, for $I_w \triangleq \{i \in [k] : p_i^w \leq 1/3\}$, it holds that $|I_w| \geq (1 - 6q\delta) \cdot k$. Observe that for any $i \in I_w$, it holds that $\Pr[D^w(i) = x_i] \geq 1 - 1/3 = 2/3$, as required. □

5.8.5 Proof of Claim 5.11.2

In this section we provide the proof of Claim 5.11.2. The proof is similar to the proof of Claim 5.11.1. However, note that Claim 5.11.1 and Claim 5.11.2 deal with different objects: While Claim 5.11.1 deals with the planes of the tensor code and the *plane* scPCPPs, Claim 5.11.2 deals with the lines of the tensor and the *point-line* scPCPPs. In particular, every plane in the tensor code is coupled with a *unique* plane scPCPP proof, whereas every line in the tensor code is coupled with n *different* point-line scPCPPs, one for each point on the line. We begin by restating Claim 5.11.2. Recall that $\gamma = \delta(C)/(24d)$.

Claim 5.18.1 (restated)**.** *Assuming $\bar{c}$ is $\gamma \cdot \delta_{C'}(w)$-close to being a codeword of C^{t_1}, if $\delta_{\bar{p}^{\mathsf{lines}}} > \delta_{C'}(w)$, then* $\Pr_T[T^w = 0] \geq \mathsf{poly}\big(\delta_{C'}(w)\big)$.

Proof. By the lemma's hypothesis, $\bar{c}$ is $\delta_{\bar{c}}$-close to $C(x)^{t_1}$, where $\delta_{\bar{c}} \leq \gamma \cdot \delta_{C'}(w)$. By an averaging argument, with probability at least $2/3$ the random copy $\mathbf{c}$ is $3\delta_{\bar{c}}$-close to $C(x)$. We say that a point $\bar{\imath} \in [n]^d$ in $\mathbf{c}$ is corrupted if $\mathbf{c}_{\bar{\imath}} \neq C'(x)_{\bar{\imath}}$ and so, there are at most $3\delta_{\bar{c}} n^d$ corrupted points in $\mathbf{c}$. Since there are $d \cdot n^{d-1}$ axis-parallel lines in $\mathbf{c}$, then on average, the number of corrupted points in a random axis-parallel line is at most $\frac{3\delta_{\bar{c}} n^d}{d \cdot n^{d-1}} \leq 3\delta_{\bar{c}} n$. Thus, by an averaging argument, we obtain that at most $\frac{\delta_{\bar{p}}}{4}$ fraction of the axis-parallel lines in $\mathbf{c}$ contain at least $\frac{4}{\delta_{\bar{p}}} \cdot 3\delta_{\bar{c}} n$ corrupted points.

Recall that every axis-parallel line ℓ has n corresponding *point-line* scPCPP proofs (one for each point on ℓ). For every line ℓ we view these n proofs as one concatenated proof for the line ℓ. By an averaging argument, with probability at least $\delta_{\bar{p}} \triangleq \delta_{\bar{p}^{\mathsf{lines}}}/2$ the random copy $\bar{p}$ in $\bar{p}^{\mathsf{lines}}$ is $\delta_{\bar{p}}$-far from its corresponding set of canonical proofs, $\pi_{\mathsf{lines}}(x)$. Assume from now on that $\bar{p}$ is $\delta_{\bar{p}}$-far from $\pi_{\mathsf{lines}}(x)$. By another averaging argument, at least a $\delta_{\bar{p}}/2$ fraction of the concatenated line proofs (i.e., proofs which consists of n point-line scPCPP proofs) are $\delta_{\bar{p}}/2$-far from their corresponding (concatenated) canonical line proofs.

By combining the conclusions of the last two paragraphs, we deduce that $\Omega\big(\delta_{C'}(w)\big)$-fraction of the axis-parallel lines ℓ in $\mathbf{c}$ are both $\delta(C_0)/2$-close to the restriction of the tensor codeword $C(x)$ to ℓ, and their corresponding (concatenated) proofs are $\Omega\big(\delta_{C'}(w)\big)$-corrupted; that is, there is a subset of lines, denoted BAD, which consists of at least $\frac{\delta_{\bar{p}}}{4}$ fraction of all the lines in $\mathbf{c}$ that are $\delta(C_0)/2$-close to $C(x)|_\ell$ (recall that $\delta_{\bar{c}} \leq \gamma \cdot \delta_{C'}(w)$ and $\delta_{\bar{p}} > \delta_{C'}(w)$, therefore $\frac{12 \cdot \delta_{\bar{c}}}{\delta_{\bar{p}}} < \delta(C_0)/2$), and in addition satisfy the following: For every $\ell \in$ BAD, the n (alleged) *point-line* scPCPP proofs that correspond to ℓ are $\delta_{\bar{p}}/2$-far from their (correct) canonical proofs in $\pi_{\mathsf{lines}}(x)$. By an averaging argument, for every $\ell \in$ BAD it holds that $\delta_{\bar{p}}/4$ fraction of the point-line PCPP proofs that correspond to the line ℓ (recall that there are n such proofs) are $\delta_{\bar{p}}/4$-far from their canonical proof in $\pi_{\mathsf{lines}}(x)$.

Recall that the tester chooses a line $\ell = \ell_{j,\bar{\imath}}$ by sampling uniformly at random a point $\bar{\imath} \in [n]^d$ and a direction $j \in [d]$. Notice that for if $\ell \in$ BAD, then with probability $\delta_{\bar{p}}/4$, in order for input $\mathbf{c}|_\ell$ and the proof $p^{\ell_{j,\bar{\imath}}}$ (that refers to the same line as ℓ) to be a valid claim for the input-proof language that $V^{\mathsf{line}}(i_j, \mathbf{c}_{\bar{\imath}})$ verifies, one must make at least one

of the following changes: (1) change a fraction of at least $\frac{\delta_{\bar{p}}}{4}$ of the proof $p^{\ell_{j,\bar{\imath}}}$ such that it matches $\pi_{\mathsf{line}}\big(C(x)|_{\ell_{j,\bar{\imath}}}, i_j\big)$, or (2) change a fraction of at least $\delta(C_0)/2$ of $\mathbf{c}|_\ell$ (since $p^{\ell_{j,\bar{\imath}}}$ might be a valid proof for input $C_0(y) \neq \mathbf{c}|_\ell$). Thus, for every $\ell_{j,\bar{\imath}} \in \text{BAD}$, the probability that $V^{\mathsf{line}}(i_j, \mathbf{c}_{\bar{\imath}})$ rejects input $c|_{\ell_{j,\bar{\imath}}}$ and proof $p^{\ell_{j,\bar{\imath}}}$ is at least polynomial in $\delta_{C'}(w)$.

Putting it all together, with probability $2/3$ we hit a random copy $\mathbf{c}$ of the tensor code that is $3\delta_{\bar{c}}$-close to $C(x)$. Furthermore, with probability at least $\delta_{\bar{p}}$ we hit a random copy $\bar{p}$ that is $\delta_{\bar{p}}$-corrupted, and subsequently, with probability $\delta_{\bar{p}}/2$ we hit a set of n line scPCPP proofs that are $\delta_{\bar{p}}/2$-corrupted. Moreover, with probability at least $\delta_{\bar{p}}/4$ we hit a point-line scPCPP proof that is $\delta_{\bar{p}}/4$ corrupted. Finally, assuming the foregoing, the corresponding scPCPP verifier rejects with probability $\mathsf{poly}\left(\delta_{C'}(w)\right)$. Therefore,

$$\Pr_T[T^w = 0] \geq \frac{2}{3} \cdot \delta_{\bar{p}} \cdot \frac{\delta_{\bar{p}}}{2} \cdot \frac{\delta_{\bar{p}}}{4} \cdot \mathsf{poly}\left(\delta_{C'}(w)\right) \geq \mathsf{poly}\big(\delta_{C'}(w)\big).$$

□

Chapter 6

Universal Locally Testable Codes

6.1 Introduction

Locally testable codes [FS95, RS96, GS06] are error-correcting codes that have local procedures for ascertaining the integrity of purported codewords. More accurately, a code C is a locally testable code (LTC) if there exists a probabilisitic algorithm (tester) that gets a proximity parameter $\varepsilon > 0$, makes a small number of queries to a string w, and with high probability accepts if w is a codeword of C and rejects if w is ε-far from C. The query complexity of the tester is the number of queries that it makes (also referred to as the locality of the LTC).

6.1.1 The Notion of Universal-LTC

In this chapter we initialize a study of a generalization of the notion of LTCs, which we call universal locally testable codes. A universal-LTC is a code that not only admits a local test for membership in the code C but also a local test for membership in a *family of subcodes of* C. More precisely, a binary code $C : \{0,1\}^k \to \{0,1\}^n$ is a q-local universal-LTC for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ if for every $i \in [M]$ the subcode $\Pi_i \{C(x) : f_i(x) = 1\}$ is locally testable with query complexity q. Viewed in an alternative perspective, such codes allow for testing properties of the encoded *message*; that is, testing whether $C(x)$ is an encoding of a message x that satisfies a function $f_i \in \mathcal{F}$.

Universal-LTCs implicit in previous works. We note that the notion of universal-LTCs is implicit in the literature. For instance, the *long code* [BGS98], which maps a message to its evaluations under all Boolean functions, can be thought of as the "ultimate" universal-LTC for all Boolean functions. To see this, recall that the long code is both locally testable and *correctable* (i.e., there exists a local algorithm that can recover any bit of a slightly corrupted codeword). Now, observe that we can test a subcode $\{\mathsf{LC}(x) : f(x) = 1\}$, where $\mathsf{LC} : \{0,1\}^k \to \{0,1\}^{2^{2^k}}$ is the long code and $f : \{0,1\}^k \to \{0,1\}$ is some Boolean function, by first running the codeword test (and

rejecting if it rejects), and then running the local correcting algorithm with respect to the bit in $\mathsf{LC}(x)$ that corresponds to the evaluation of f on x. Note, however, the ability to test all subcodes comes at the cost of great redundancy, since the length of the long code is *doubly exponential* in the length of the message.

By an analogous argument, the *Hadamard code*, which maps a message to its evaluations under all *linear* Boolean functions, can be thought of as a universal-LTC for all *linear* Boolean functions. Note that the length of the Hadamard code is *exponential* in the length of the message. Another example is the inner PCP for satisfiability of quadratic equations [ALM+98], wherein the (exponentially long) PCP oracle is an encoding of an assignment, *independent from the set of quadratic equations it allegedly satisfies*. Hence, this PCP is an "universal" encoding that admits a local test for the satisfiability of *any* function that is given by a set of quadratic equations, and so it can be thought of as a universal-LTC for quadratic equations.

In this chapter, we ask whether universal-LTCs can be constructed for any family of functions $\mathcal{F}$, and what are the optimal parameters (i.e., the code's length, locality, and number of subcodes for which it admits a local test) that can be obtained by universal-LTCs.

Universal (relaxed) Locally Decodable Codes. Before proceeding to present our results, we highlight a close connection between universal-LTCs and a universal generalization of the notion of relaxed local decodability. Recall that a code is said to be a relaxed locally decodable code (relaxed-LDC) [BSGH+06] if for every location i in the message there exists a local algorithm (decoder) that is given query access to an alleged codeword, and satisfies the following: If the codeword is valid, the decoder successfully decodes the i'th symbol, and if the codeword is corrupted, the decoder, with high probability, either decodes correctly or rejects (indicating it detected a corruption in the codeword). It turns out that universal-LTCs immediately imply a generalization of the notion of relaxed-LDCs, which we describe next. (We also note that, under certain conditions, universal-LTCs imply (non-relaxed) local decodability, see Section 6.7.1.)

We define a universal relaxed locally decodable code (in short, universal-LDC) for a family of functions $\mathcal{F}$ (analogously to universal-LTCs) as a relaxed-LDC wherein, instead of local procedures for (relaxed) decoding of bits of the message x, we have local procedures for (relaxed) decoding of the value of $f(x)$ for every $f \in \mathcal{F}$.

Now, let $\mathcal{F}$ be a family of Boolean functions. Observe that a universal-LTC for $\mathcal{F} \cup (1 - \mathcal{F})$ (i.e., a code with a tester $T_{f,b}$ for each subcode $\{C(x) : f(x) = b\}$, where $f \in \mathcal{F}$ and $b \in \{0, 1\}$) implies a universal-LDC for $\mathcal{F}$, which is also locally *testable*, and vice versa. To see this, consider the following local decoding procedure for $f \in \mathcal{F}$: To decode $f(x)$, invoke $T_{f,0}$ and $T_{f,1}$. If one tester accepted and the other rejected, rule according to the accepting tester, and otherwise reject. The reader may verify that this is indeed a (relaxed) local decoding procedure (see Section 6.7.1 for discussion and generalizations). For the other direction, to test the subcode $\{C(x) : f(x) = 1\}$, first run the codeword test, then decode the value of $f(x)$ and accept if and only if it equals 1 (i.e., a decoded value of 0 and a decoding error both cause rejection). We remark that all universal-LTCs

in this chapter can be easily extended to families of the type $\mathcal{F} \cup (1 - \mathcal{F})$, and thus we also obtain analogous results for universal-LDCs.

On "uniformity" with respect to $\mathcal{F}$. For simplicity, we defined universal LTCs and LDCs in a "non-uniform" manner with respect to the family of functions $\mathcal{F}$; that is, we required that for every function $f \in \mathcal{F}$, there exists a testing or decoding procedure. A stronger, "$\mathcal{F}$-uniform", definition would require that there exists a procedure that receives $f \in \mathcal{F}$ as a parameter and tests or decodes with respect to f. We remark that all of our upper bounds can be easily adapted to satisfy the stronger $\mathcal{F}$-uniform condition, while our lower bounds hold even without this condition.

6.1.2 Our Results

To simplify the presentation of our results, throughout the introduction we fix the proximity parameter ε to a small constant, and when we refer to "codes", we shall actually mean error-correcting codes with linear distance. Our first result shows "canonical" universal-LTCs for *any* family of functions.

Theorem 3 (informally stated, see Theorem 6.2)**.** *Let $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i \in [M]}$ be any family of Boolean functions that can each be computed by a size $s = s(k)$ circuit. Then, there exists a (one-sided error)* universal-LTC *$C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(M \cdot s)}$ for $\mathcal{F}$ with query complexity $O(1)$.*

We complement the foregoing"canonical" universal-LTC with a general lower bound on the query complexity of universal-LTCs, as a function of the encoding's length and number of subcodes for which it admits a local test.

Theorem 4 (informally stated, see Theorem 6.5)**.** *Let $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i \in [M]}$ be a family of* distinct *Boolean functions. Then, every* universal-LTC *$C : \{0,1\}^k \to \{0,1\}^n$ for $\mathcal{F}$ must have query complexity $\Omega(\log\log M - \log\log n - \log k)$. Furthermore, if the functions in $\mathcal{F}$ are* pairwise far *(i.e., $\Pr_x[f_i(x) \neq f_j(x)] = \Omega(1)$ for every $i \neq j$), then the query complexity is $\Omega(\log\log M - \log\log n)$.*

Note that $\log\log M - \log\log n = O(1)$ implies a lower bound of $n \geq M^{\Omega(1)}$. In contrast, recall that Theorem 3 shows an upper bound of $n = \tilde{O}(M \cdot s)$, where s bounds the circuit size for computing each $f \in \mathcal{F}$. Thus, for sufficiently large families of pairwise-far functions, Theorem 4 shows that the length of the canonical universal-LTC (in Theorem 3) is optimal, up to a constant power. This raises the question of whether the aforementioned slackness can be removed. We answer this question to the affirmative, albeit for a specific family of functions.

Specifically, we show a universal-LTC $C : \{0,1\}^k \to \{0,1\}^{k^{1.01}}$ for a family of $M = \binom{k}{m} \cdot 2^{2^m}$ functions, namely the family of m-juntas,[1] with query complexity $\tilde{O}(m)$; note that for a large constant m, the number of functions M is an arbitrarily large *polynomial*

[1] That is, all Boolean functions that only depend on m of their k variables.

in the code's length $k^{1.01} < M^{1.01/m}$, whereas for the canonical universal-LTC the length is linear in M.

In addition, note that the lower bound in Theorem 4 allows for a tradeoff between the universal-LTC's length and locality (i.e., query complexity), whereas Theorem 3 only shows universal-LTCs in the constant locality regime. In Section 6.6 we show that for the family of m-juntas, there exists a universal-LTC that allows for a tradeoff between locality and length. (See Proposition 6.8 for a precise statement.)

6.2 Preliminaries

We begin with standard notations:

- We denote the *absolute distance*, over alphabet Σ, between two strings $x \in \Sigma^n$ and $y \in \Sigma^n$ by $\Delta(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|$ and their *relative distance* by $\delta(x,y) \stackrel{\text{def}}{=} \frac{\Delta(x,y)}{n}$. If $\delta(x,y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we denote the *absolute distance* of x from a non-empty set $S \subseteq \Sigma^n$ by $\Delta(x,S) \stackrel{\text{def}}{=} \min_{y \in S} \Delta(x,y)$ and the *relative distance* of x from S by $\delta(x,S) \stackrel{\text{def}}{=} \min_{y \in S} \delta(x,y)$. If $\delta(x,S) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S. We denote the projection of $x \in \Sigma^n$ on $I \subseteq [n]$ by $x|_I$.

- We denote by $A^x(y)$ the output of algorithm A given direct access to input y and oracle access to string x. Given two interactive machines A and B, we denote by $(A^x, B(y))(z)$ the output of A when interacting with B, where A (respectively, B) is given oracle access to x (respectively, direct access to y) and both parties have direct access to z. Throughout this chapter, probabilistic expressions that involve a randomized algorithm A are taken over the inner randomness of A (e.g., when we write $\Pr[A^x(y) = z]$, the probability is taken over the coin-tosses of A).

Integrality. Throughout this chapter, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the closest integer.

Uniformity. To facilitate notation, throughout this chapter we define all algorithms *non-uniformly*; that is, we fix an integer $n \in \mathbb{N}$ and restrict the algorithms to inputs of length n. Despite fixing n, we view it as a generic parameter and allow ourselves to write asymptotic expressions such as $O(n)$. We remark that while our results are proved in terms of non-uniform algorithms, they can be extended to the uniform setting in a straightforward manner.

Circuit Size. We define the size $s(k)$ of a Boolean circuit $C : \{0,1\}^k \to \{0,1\}$ as the number of *gates* C contains. We count the input vertices of C as gates, and so $s(k) \geq k$.

We shall write $f \in \mathsf{SIZE}\big(s(k)\big)$ to state that a Boolean function $f : \{0,1\}^k \to \{0,1\}$ can be computed by a Boolean circuit of size $s(k)$.

6.2.1 Locally Testable and Decodable Codes

Let $k, n \in \mathbb{N}$. A code over alphabet Σ with distance d is a function $C : \Sigma^k \to \Sigma^n$ that maps messages to codewords such that the distance between any two codewords is at least $d = d(n)$. If $d = \Omega(n)$, we say that C has linear distance. If $\Sigma = \{0,1\}$, we say that C is a binary code. If C is a linear map, we say that it is a linear code. The relative distance of C, denoted by $\delta(C)$, is d/n, and its rate is k/n. When it is clear from the context, we shall sometime abuse notation and refer to the code C as the set of all codewords $\{C(x)\}_{x\in\Sigma^k}$. Following the discussion in the introduction, we define locally testable codes and locally decodable codes as follows.

Definition 6.1 (Locally Testable Codes)**.** *A code $C : \Sigma^k \to \Sigma^n$ is a locally testable code (LTC) if there exists a probabilistic algorithm (tester) T that, given oracle access to $w \in \Sigma^n$ and direct access to proximity parameter ε, satisfies:*

1. *Completeness: For any codeword $w = C(x)$, it holds that $\Pr[T^{C(x)}(\varepsilon) = 1] \geq 2/3$.*
2. *Soundness: For any $w \in \{0,1\}^n$ that is ε-far from C, it holds that $\Pr[T^{w}(\varepsilon) = 0] \geq 2/3$.*

The query complexity of a LTC is the number of queries made by its tester (as a function of ε and k). A LTC is said to have one-sided error if its tester satisfy perfect completeness (i.e., accepts valid codewords with probability 1).

Definition 6.2 (Locally Decodable Codes)**.** *A code $C : \Sigma^k \to \Sigma^n$ is a locally decodable code (LDC) if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$ and a probabilistic algorithm (decoder) D that, given oracle access to $w \in \Sigma^n$ and direct access to index $i \in [k]$, satisfies the following condition: For any $i \in [k]$ and $w \in \Sigma^n$ that is δ_{radius}-close to a codeword $C(x)$ it holds that $\Pr[D^w(i) = x_i] \geq 2/3$. The query complexity of a LDC is the number of queries made by its decoder.*

We shall also need the notion of relaxed-LDCs (introduced in [BSGH+06]). Similarly to LDCs, these codes have decoders that make few queries to an input in attempt to decode a given location in the message. However, unlike LDCs, the relaxed decoders are allowed to output a special symbol that indicates that the decoder detected a corruption in the codeword and is unable to decode this location. Note that the decoder must still avoid errors (with high probability).[2]

[2]The full definition of relaxed-LDCs, as defined in [BSGH+06] includes an additional condition on the success rate of the decoder. Namely, for every $w \in \{0,1\}^n$ that is δ_{radius}-close to a codeword $C(x)$, and for at least a ρ fraction of the indices $i \in [k]$, with probability at least $2/3$ the decoder D outputs the i^{th} bit of x. That is, there exists a set $I_w \subseteq [k]$ of size at least ρk such that for every $i \in I_w$ it holds that $\Pr[D^w(i) = x_i] \geq 2/3$. We omit this condition since it is irrelevant to our application, and remark that every relaxed-LDC that satisfies the first two conditions can also be modified to satisfy the third conditions (see [BSGH+06, Lemmas 4.9 and 4.10]).

Definition 6.3 (relaxed-LDC)**.** *A code $C : \Sigma^k \to \Sigma^n$ is a relaxed-*LDC *if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$,*

1. *(Perfect) Completeness: For any $i \in [k]$ and $x \in \Sigma^k$ it holds that $D^{C(x)}(i) = x_i$.*
2. *Relaxed Soundness: For any $i \in [k]$ and any $w \in \Sigma^n$ that is δ_{radius}-close to a (unique) codeword $C(x)$, it holds that*

$$\Pr[D^w(i) \in \{x_i, \perp\}] \geq 2/3.$$

There are a couple of efficient constructions of codes that are both relaxed-LDCs and LTCs (see [BSGH+06, GGK15]). We shall need the construction in [GGK15], which has the best parameters for our setting.[3]

Theorem 6.1 (e.g., [GGK15, Theorem 1.1])**.** *For every $k \in \mathbb{N}$ and $\alpha > 0$ there exists a (linear) code $C : \{0,1\}^k \to \{0,1\}^{k^{1+\alpha}}$ with linear distance, which is both a relaxed-*LDC *and a (one-sided error)* LTC *with query complexity* $\mathsf{poly}(1/\varepsilon)$*.*

6.3 The Definition of Universal Locally Testable Codes

Following the discussion in the introduction, we define universal locally testable codes as follows.

Definition 6.4. *Let $k, M \in \mathbb{N}$, and $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i \in [M]}$ be a family of functions. A universal locally testable code (*universal-LTC*) for $\mathcal{F}$ with query complexity $q = q(k, \varepsilon)$ is a code $C : \{0,1\}^k \to \{0,1\}^n$ such that for every $i \in [M]$ and $\varepsilon > 0$ there exists an ε-tester for the subcode $\Pi_i \{C(x) : f_i(x) = 1\}$ with query complexity q. A* universal-LTC *is said to have one-sided error if all of its testers satisfy perfect completeness.*

Notation (ε-testing). We shall refer to a universal-LTC with respect to a specific proximity parameter $\varepsilon > 0$ as a $\text{universal-LTC}_\varepsilon$.

Organization. We start, in Section 7.4.1, by showing a canonical universal-LTC for every family of functions. This construction relies on a PCP-based machinery for asserting consistency of encodings, which we shall use throughout this chapter. Next, in Section 6.5, we prove general lower bounds on the query complexity of universal-LTCs as a function of the code's length and number of functions it can test. Finally, in Section 6.6, we show a specific family of functions (namely, the family of m-juntas, i.e., Boolean functions that only depend on m of their variables) for which we can obtain a smooth tradeoff between the universal-LTC length and locality.

[3] Specifically, the codes in [GGK15] are meaningful for every value of the proximity parameter, whereas the codes in [BSGH+06] require $\varepsilon > 1/\mathsf{polylog}(k)$.

6.4 The Canonical Universal-LTC

In this subsection we show a methodology for constructing an $O(1)$-local universal-LTC for any family of Boolean functions.

Theorem 6.2. *Let $t(k)$ be a proper complexity function, and let $\mathcal{F} = \{f_i : \{0,1\}^k \to \{0,1\}\}_{i\in[M]}$ be a family of functions such that for every $i \in [M]$, the function f_i can be computed by a size $t(k)$ circuit (i.e., $f_i \in \mathsf{SIZE}(t(k))$). Fix $n = M \cdot \tilde{O}(t(k))$. Then, for every $\varepsilon > 1/\mathsf{polylog}(n)$ there exists a (one-sided error) $\mathsf{universal\text{-}LTC}_\varepsilon$ $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n)}$ for $\mathcal{F}$ with linear distance and query complexity $O(1/\varepsilon)$.*

We remark that, loosely speaking, the "canonical" universal-LTC above tightly matches the lower bound (see Theorem 6.5) in the low query complexity regime, for a reasonable setting of the parameters; see Section 6.5 for a more accurate statement.

The key idea for proving Theorem 6.2 is to design a universal-LTC that includes, for every $f \in \mathcal{F}$, a PCP encoding of the message x, which asserts the value of $f(x)$; this way we obtain a local test for each function in $\mathcal{F}$, simply by running its corresponding PCP verifier. The main problem, however, is that given concatenated PCP oracles we cannot locally verify that all of these PCPs are consistent with the exact same message. To overcome this issue, we shall first show a machinery for "bundling" encodings together in a way that allows for locally testing that all of the encodings are consistent with the same message. The key components for this construction are PCPs of proximity, which we discuss below.

6.4.1 Preliminaries: PCP of proximity

PCPs of proximity (PCPPs) [BSGH+06, DR06] are a variant of PCP proof systems, which can be thought of as the PCP analogue of *property testing*. Recall that a standard PCP is given explicit access to a statement and oracle access to a proof. The PCP verifier is required to probabilistically verify whether the (explicitly given) statement is correct, by making few queries to proof. In contrast, a PCPP is given oracle access to a statement and a proof, and is only allowed to make a small number of queries to both the statement and the proof. Since a PCPP verifier only sees a small part of the statement, it cannot be expected to verify the statement precisely. Instead, it is required to only accept correct statements and reject statements that are far from being correct (i.e., far in Hamming distance from any valid statement). More precisely, PCPs of proximity are defined as follows.

Definition 6.5. *Let V be a probabilistic algorithm (verifier) that is given* explicit *access to a proximity parameter $\varepsilon > 0$,* oracle *access to an* input *$x \in \{0,1\}^k$ and to a* proof *$\bar{p} \in \{0,1\}^n$. We say that V is a* PCPP *verifier for language L if it satisfies the following conditions:*

- *Completeness: If $x \in L$, there exists a proof $\bar{p}$ such that the verifier always accepts the pair $(x, \bar{p})$; i.e., $V^{x,\bar{p}}(\varepsilon) = 1$.*

- *Soundness: If x is ε-far from L, then for every $\bar{p}$ the verifier rejects the pair $(x, \bar{p})$ with high probability; that is,* $\Pr[V^{x,\bar{p}}(\varepsilon) = 0] \geq 2/3$.

The length of the PCPP *is n and the query complexity is the number of queries made by V to* both *x and $\bar{p}$.*

We shall use the following PCPP due to Ben-Sasson and Sudan [BS05] and Dinur [Din07b].

Theorem 6.3 (Short PCPPs for NP)**.** *For every $L \subseteq \{0,1\}^k$ that can be computed by a circuit of size $t(k)$, there exists a* PCPP *with query complexity $q = O(1/\varepsilon)$ and length $t(k) \cdot \mathsf{polylog}(t(k))$.*

6.4.2 Consistency-Testable Bundles

Building on techniques of Ben-Sasson at el. [BSGH+06], we show a way to bundle together (possibly partial) encodings of the same message such that it possible to locally test that all these encodings are indeed consistent. That is, we are given some encodings $E_1, \ldots, E_s : \{0,1\}^k \to \{0,1\}^n$, and we wish to encode a *single* message $x \in \{0,1\}^k$ by all of these encodings (i.e., to bundle $E_1(x), \ldots, E_s(x)$) such that we can test that all of the encodings are valid and consistent with the same message x. We shall need such bundles twice in this chapter: In Section 6.4.3 each E_i will simply correspond to a Boolean function $f_i \in \mathcal{F}$, and in Section 6.6 the E_i's will correspond to encodings of small chunks x.

The main idea is to construct a bundle that consists of three parts: (1) the (explicit) message x, (2) the encodings $E_1(x), \ldots, E_s(x)$, and (3) PCPPs that asset the consistency of the first part (the message) with each purported encoding $E_i(x)$ in the second part. However, such PCPPs can only ascertain that each purported pair of message and encoding, denoted (y, z_i), is *close* to a valid pair $(x, E_i(x))$. Thus, in this way we can only verify that the bundle consists of encodings of pairwise-close messages, rather than being close to encodings of a single message (e.g., the PCPPs may not reject a bundle $(x, E_1(y_1), \ldots, E_s(y_s))$ wherein each y_i is close to x).

To avoid this problem, we also encode the message via an error-correcting code ECC, so the bundle is of the form $\big(\mathsf{ECC}(x), (E_1(x), \ldots, E_s(x)), (\mathsf{PCPP}_1(x), \ldots, \mathsf{PCPP}_s(x))\big)$. Now, each PCPP ascertains that a purported pair (y, z_i) is close to $(\mathsf{ECC}(x), E_i(x))$. Due to the distance of ECC, this allows to verify that the bundle consists of s (close to valid) encodings of the *same* message. Lastly, we repeat $\mathsf{ECC}(x)$ such that it constitutes most of the bundle's length, and so if an alleged bundle is far from valid, its copies of $\mathsf{ECC}(x)$ must be corrupted, and so the bundle itself constitutes an error-correcting code that is locally testable (by verifying at random one of the PCPPs in the bundle).

More precisely, consider the following way of bundling several encodings of the same message.

Construction 6.4 (Consistency-Testable Bundles)**.** *Let $E_1, \ldots, E_s : \{0,1\}^k \to \{0,1\}^n$ be encodings such that for every $i \in [s]$, the problem of (exactly) deciding whether $(x, y) \in$*

$\{0,1\}^{k+n}$ *satisfies* $y = E_i(x)$ *can be computed by a size* $t(k)$ *circuit. The* consistency-testable bundle *of* $\{E_i(x)\}_{i\in[s]}$ *is the code* $B(x) : \{0,1\}^k \to \{0,1\}^\ell$ *that consists of the following ingredients.*

1. *An (arbitrary) code* $\mathsf{ECC} : \{0,1\}^k \to \{0,1\}^{n'}$ *with linear distance, which can be computed by a size* $\tilde{O}(n')$ *circuit, where* $n' = \tilde{O}(k)$.
2. *Encodings* $E_1, \ldots, E_s$ *(given by the application) that we wish to bundle.*
3. PCP *of proximity oracles* $\bar{p}_1, \ldots, \bar{p}_s$ *for the language*

$$L_i = \{(a,b) \ : \ \exists x \in \{0,1\}^k \text{ such that } a = \mathsf{ECC}(x)^{r_a} \text{ and } b = E_i(x)^{r_b}\}.$$

where and r_a, r_b *are set such that* $|a| \approx |b| = O(t(k))$.

Let $\varepsilon \geq 1/\mathsf{polylog}(s \cdot t(k))$. *Consider the bundle*

$$B(x) = \Big(\mathsf{ECC}(x)^r, \big(E_1(x), \ldots, E_s(x)\big), \big(\bar{p}_1(x), \ldots, \bar{p}_s(x)\big)\Big),$$

where the length of each PCPP *oracle* $\bar{p}_i(x)$ *is* $\tilde{O}(t(k))$,[4] *and where* r *is the minimal integer such that the first part of the bundle constitutes* $(1-\varepsilon/2)$ *fraction of the bundle's length (i.e.,* $|\mathsf{ECC}(x)|^r \geq (1-\varepsilon/2)\cdot\ell$*).*

Note that the length of B is $\ell = \tilde{O}(s \cdot t(k))$ and that B has linear distance, because $|\mathsf{ECC}(x)|^r$ dominates B's length.

Notation for (alleged) bundled. For the analysis, when we consider an arbitrary string $w \in \{0,1\}^\ell$ (which we think of as an alleged bundle), we view $w \in \{0,1\}^{\ell_1+\ell_2+\ell_3}$ as a string composed of three parts (analogous to the three parts of Construction 6.4):

1. The anchor, $\widetilde{\mathsf{ECC}}(x) = (\widetilde{\mathsf{ECC}}(x)_1, \ldots, \widetilde{\mathsf{ECC}}(x)_r) \in \{0,1\}^{n'\cdot r}$, which consists of r alleged copies of $\mathsf{ECC}(x)$;
2. The bundled encodings $(\widetilde{E_1}(x), \ldots, \widetilde{E_s}(x)) \in \{0,1\}^{n\cdot s}$, which allegedly equals $(E_1(x), \ldots, E_s(x))$;
3. The PCPPs $(\widetilde{\bar{p}_1}(x), \ldots, \widetilde{\bar{p}_s}(x)) \in \{0,1\}^{\tilde{O}(t(k))\cdot s}$, which allegedly equals $(\bar{p}_1(x), \ldots, \bar{p}_s(x))$.

We show that there exists a local test that can ascertains the validity of the bundle as well as asserts the consistency of any encoding E_i in the bundle with the *anchor* of the bundle. Note that since the bundle's anchor dominates its length, it is possible that the bundle is very close to valid, and yet all of the E_i's are heavily corrupted. Thus, we also need to provide a test for the validity of each E_i and its consistency with the anchor.

[4] Note that $L_i \in \mathsf{SIZE}(m)$ by the hypothesis regarding ECC and E_i. Thus, by Theorem 7.1, such a PCPP exists.

Proposition 6.6. *For every bundle $B(x)$, as in Construction 6.4, there exists a consistency test T that for every $\varepsilon \geq 1/\mathsf{polylog}(\ell)$ makes $O(1/\varepsilon)$ queries to a string $w \in \{0,1\}^\ell$ and satisfies the following conditions.*

1. *If $w = B(x)$, then for every $i \in \{0\} \cup [s]$ it holds that $\Pr[T^w(i) = 1] = 1$.*
2. *If w is ε-far from B, then $\Pr[T^w(0) = 0] \geq 2/3$.*
3. *For every $i \in [s]$, if there exists $x \in \{0,1\}^k$ such that w is ε-close to $B(x)$ and $\widetilde{E_i}(x)$ is ε-far from $E_i(x)$, then $\Pr[T^w(i) = 0] \geq 2/3$.*

Note that $T^w(0)$ is a codeword test for B, whereas for every $i \in [s]$, the test $T^w(i)$ asserts that $\widetilde{E_i}$ is close to an encoding of the anchor. To verify that w is a bundle wherein all encodings refer to the same message (the anchor), we have to invoke $T^w(i)$ for all $i \in \{0\} \cup [s]$, but typically we will be interested only in the consistency of one encoding with the anchor, where this encoding is determined by the application.

Proof of Proposition 7.7. We show that for every bundle $B(x)$, as in Construction 6.4, there exists a consistency test T that, for every $\varepsilon \geq 1/\mathsf{polylog}(\ell)$, makes $O(1/\varepsilon)$ queries to a string $w \in \{0,1\}^\ell$ and satisfies the following conditions.

1. If $w = B(x)$, then for every $i \in \{0\} \cup [s]$ it holds that $\Pr_T[T^w(i) = 1] = 1$.
2. If w is ε-far from B, then $\Pr[T^w(0) = 0] \geq 2/3$.
3. For every $i \in [s]$, if there exists $x \in \{0,1\}^k$ such that w is ε-close to $B(x)$ and $\widetilde{E_i}$ is ε-far from $E_i(x)$, then $\Pr[T^w(i) = 0] \geq 2/3$.

Let $\varepsilon \geq 1/\mathsf{polylog}(\ell)$, and assume without loss of generality that $\varepsilon < \delta(\mathsf{ECC})/2$.[5] For every $i \in [s]$ denote by V_i the PCPP verifier for the language

$$L_i = \{(a,b) \ : \ \exists x \in \{0,1\}^k \text{ such that } a = \mathsf{ECC}(x)^{r_a} \text{ and } b = E_i(x)^{r_b}\},$$

with respect to proximity parameter $\varepsilon/6$ and soundness $9/10$. Consider the ε-tester T that is given $i \in \{0\} \cup [s]$ and oracle access to $w = (\widetilde{\mathsf{ECC}(x)}, (\widetilde{E_i})_{i\in[s]}, (\widetilde{p_i})_{i\in[s]}) \in \{0,1\}^\ell$ and accepts if both of the following tests accept.

1. Repetition Test: Query two random copies from the long-code part of w and check if they agree on a random location. More accurately, select uniformly at random $j, j' \in [r]$ and reject if and only if $\widetilde{\mathsf{ECC}(x)}_j$ and $\widetilde{\mathsf{ECC}(x)}_{j'}$ disagree on a *random* coordinate. Repeat this test $O(1/\varepsilon)$ times.
2. Consistency Test: Choose uniformly $j \in [r]$. If $i = 0$, choose uniformly $i' \in [s]$, otherwise set $i' = i$. Reject if the verifier $V_{i'}$ rejects on input $(\widetilde{\mathsf{ECC}(x)}_j^{r_a}, \widetilde{E_{i'}(x)}^{r_b})$ and proof $\widetilde{p_{i'}}(x)$.

[5] The relative distance of ECC is constant, so if $\varepsilon \geq \delta(\mathsf{ECC})/2$, we can set the proximity parameter to $\delta(\mathsf{ECC})/2$, increasing the complexity by only a constant factor.

The first condition of Proposition 7.7 follows by construction. For the other conditions, first observe that if $\widetilde{\mathsf{ECC}}(x)$ is far from consisting of r identical copies, then the repetition test rejects with high probability. That is, let $\hat{c} \in \{0,1\}^{n'}$ be a string that is closest on average to the copies in $\widetilde{\mathsf{ECC}}(x)$, i.e., a string that minimizes $\Delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) = \sum_{j=1}^{r} \Delta(\widetilde{\mathsf{ECC}}(x)_j, \hat{c})$. Observe that

$$\mathop{\mathbf{E}}_{j,j' \in_R [r]}[\delta(\widetilde{\mathsf{ECC}}(x)_j, \widetilde{\mathsf{ECC}}(x)_{j'})] \geq \mathop{\mathbf{E}}_{j \in_R [r]}[\delta(\widetilde{\mathsf{ECC}}(x)_j, \widetilde{\mathsf{ECC}}(x))] = \delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r).$$

If $\delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) > \varepsilon/60$, then by invoking the codeword repetition test $O(1/\varepsilon)$ times, with probability at least $2/3$ one of the invocations will reject. Otherwise, note that with probability at least $9/10$ the random copy $\widetilde{\mathsf{ECC}}(x)_j$ is $\varepsilon/6$-close to $\hat{c}$; assume hereafter that this is the case.

If w is ε-far from B, then since $\widetilde{\mathsf{ECC}}(x) \geq (1-\varepsilon/2)\ell$, it follows that $\widetilde{\mathsf{ECC}}(x)$ is $\varepsilon/2$-far from ECC^r, and thus

$$\delta_{\mathsf{ECC}^r}(\hat{c}^r) \geq \delta_{\mathsf{ECC}^r}(\widetilde{\mathsf{ECC}}(x)) - \delta(\hat{c}^r, \widetilde{\mathsf{ECC}}(x)) = \varepsilon/2 - \varepsilon/60 > \varepsilon/3.$$

Recall that we assumed that $\delta(\widetilde{\mathsf{ECC}}(x)_j, \hat{c}) \leq \varepsilon/6$, and so $\delta_{\mathsf{ECC}}(\widetilde{\mathsf{ECC}}(x)_j) > \varepsilon/6$. Thus, $\Pr[V_{i'}^w = 0] \geq 9/10 \cdot 9/10$.

Finally, If there exists $x \in \{0,1\}^k$ such that w is ε-close to $B(x)$ and $\widetilde{E_i}(x)$ is ε-far from $E_i(x)$, then since $\delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) \leq \varepsilon/60$, it follows that with probability at least $9/10$ the random copy $\widetilde{\mathsf{ECC}}(x)_j$ is $\varepsilon/6$-close to $\mathsf{ECC}(x)$. Hence, $(\widetilde{\mathsf{ECC}}(x)_j^{r_a}, \widetilde{E_i}(x)^{r_b})$ is at least $5\varepsilon/6$-far from L_i, and so $\Pr[V_i^w = 0] \geq 9/10 \cdot 9/10$. □

6.4.3 Proof of Theorem 6.2

Let $\mathcal{F} = \{f_i : \{0,1\}^k \to \{0,1\}\}_{i \in [M]}$ be a family of functions such that for every $i \in [M]$ it holds that $f_i \in \mathsf{SIZE}(t(k))$. Fix $n = M \cdot \tilde{O}(t(k))$ and $\varepsilon > 1/\mathsf{polylog}(n)$. We set $E_i = f_i$ for every $i \in [M]$, bundle these encodings via Proposition 7.7, and denote the bundle by $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n)}$. Note that by Proposition 7.7, the code C has linear distance.

Fixing $f_i \in \mathcal{F}$, we show an $O(1/\varepsilon)$-local ε-tester T_i for the subcode $\Pi_i \stackrel{\text{def}}{=} \{C(x) : f_i(x) = 1\}$. Given input $w \in \{0,1\}^{\tilde{O}(n)}$, the tester T_i simply invokes the bundle consistency test on w (which makes $O(1/\varepsilon)$ queries to w), with respect to proximity parameter ε and the purported copy of $f_i(x)$ in the bundle, which is a bit, denoted by z_i. The tester accepts if and only if the consistency test accepts and $z_i = 1$.

The perfect completeness of T_i follows by the one-sided error of the bundle consistency test. For the soundness, assume that w is ε-far from Π_i. By Proposition 7.7, we can assume that there exists $y \in \{0,1\}^k$ such that w is ε-close to $C(y)$ (otherwise the consistency test fails with probability $2/3$), and since w is ε-far from Π_i, it holds that $f_i(y) = 0$; furthermore, the value of w at f_i is uncorrupted (i.e., it actually equals 0),[6] and so T_i rejects.

[6] Formally, Proposition 7.7 guaranties that w contains a copy of $f_i(y)$ that is ε-close to z_i, but since $f_i(y)$ is a single bit, this means that $f_i(y)$ is uncorrupted.

6.5 General Lower Bounds

In this section we prove a general lower bound on the query complexity of universal-LTCs for *any* family of functions $\mathcal{F}$, as a function of the universal-LTC's length and the number of functions in $\mathcal{F}$. We also prove a stronger lower bound for the case that the functions in $\mathcal{F}$ are "pairwise far".

Theorem 6.5. *Let $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ be a family of distinct functions. Then, every q-local* universal-LTC$_\varepsilon$ *$C : \{0,1\}^k \to \{0,1\}^n$ for $\mathcal{F}$ with linear distance and $\varepsilon < \delta(C)/2$ must satisfy*

$$q \geq \log\log M - \log\log n - \log(k) - O(1).$$

Furthermore, if there exists $\beta = \Omega(1)$ such that $\Pr_{x\in\{0,1\}^k}[f_i(x) \neq f_j(x)] > \beta$ for every $i \neq j$, then $q = \Omega(\log\log M - \log\log n)$.

Note that in the constant locality regime (i.e., where $q = O(1)$), the lower bound for "pairwise far" functions implies that $n \geq M^c$ for some constant $c > 0$. On the other hand, recall that the canonical universal-LTC in Theorem 6.2 has query complexity $O(1)$ and length $\tilde{O}(M \cdot t(k))$, for any family of functions that can be computed by a circuit of size $t(k)$ each (recall that $t(k) \geq k$, by definition). Thus, for sufficiently large families of "pairwise far" functions, the lower bound above matches the upper bound of the canonical universal-LTC up to a constant power, where by "sufficiently large" we mean that $t(k) = \mathsf{poly}(M)$.

Proof. We prove Theorem 6.5 using two different representations of testers: when proving the main claim we view testers as *randomized* decision trees, whereas in the proof of the furthermore claim we view testers as a distribution over *deterministic* decision trees. We begin with the main claim, for which we use the following lemma, due to Goldreich and Sheffet [GS10b], which shows that the amount of randomness that suffices for testing is roughly doubly logarithmic in the size of the universe of objects it tests.

Lemma 6.7 ([GS10b, Lemma 3.7] restated)**.** *Let $k \in \mathbb{N}$, $U \subseteq \{0,1\}^k$, and let $\Pi \subseteq U$ be a property. Assume that Π has a tester with randomness complexity r, which makes q queries to a string in U. Then, Π has a tester that makes q queries and has randomness complexity $\log\log|U| + O(1)$.*

Let $C : \{0,1\}^k \to \{0,1\}^n$ be a universal-LTC for $\mathcal{F}$, and assume that each tester T_i for the subcode $\Pi_i = \{C(x) : f_i(x) = 1\}$ is given the *promise* that its input is a valid codeword of C; that is, we only consider the behavior of T_i given a codeword $C(x)$ out of the universe $U \stackrel{\text{def}}{=} \{C(x) : x \in \{0,1\}^k\}$, which consists of 2^k codewords. We shall prove a lower bound of the query complexity of the foregoing testers, and this, in particular, implies a lower bound on standard testers (which are *not* given a promise regarding their input).

Here we view a randomized decision tree is a decision tree wherein the vertices are also allowed to be labeled with a special coin-flip symbol $*$ that indicates that during

computation, one of the children of each $*$-labeled vertex is chosen uniformly at random. Note that any tester with query complexity q and randomness complexity r can be represented by a randomized decision tree of depth $q+r$ in which all vertices in the first r layers are $*$-labeled. By Lemma 6.7 we can assume without loss of generality that $r = \log\log|U| + O(1) = \log(k) + O(1)$. Observe that there are at most $(n+3)^{2^{q+\log(k)+O(1)}}$ such randomized decision trees (we bound the number of depth d decision trees over n variables by counting all possible labeling of a depth d binary tree with the names of the variables, the two terminals, and the coin-flip symbol).

Recall that for every $i \neq j$ the functions f_i and f_j are different, hence there exist $x \in \{0,1\}^k$ such that $C(x) \in \Pi_i \triangle \Pi_j$, and so by the distance of C, a tester for Π_i cannot also be a tester for Π_j. Therefore $M \leq (n+3)^{2^{q+\log(k)+O(1)}}$, and so $q \geq \log\log M - \log\log n - \log k - O(1)$.

For the furthermore claim of Theorem 6.5, for every $i \in [M]$, denote by T_i the q-query ε-tester for the subcode $\Pi_i \stackrel{\text{def}}{=} \{C(x) : f_i(x) = 1\}$, and by amplification, assume that T_i makes $q' = O(q)$ queries and obtains completeness and soundness error of at most $\delta_{\mathsf{err}} = \beta/2$. Note that if x satisfies $f_i(x) = 1$, then $C(x) \in \Pi_i$, thus the tester T_i accepts (i.e., outputs 1) with high probability, and if x satisfies $f_i(x) = 0$, then $C(x)$ is ε-far from Π_i, and thus the tester T_i rejects (i.e., outputs 0) with high probability; that is,

$$\forall x \in \{0,1\}^k \quad \Pr[T_i^{C(x)} = f_i(x)] \geq 1 - \delta_{\mathsf{err}}. \tag{6.1}$$

Hence, testing codewords of C for membership in Π_i amounts to *computing* $f_i(x)$.

Let $D_1, \ldots, D_s$ be all (binary, deterministic) depth q' decision trees over n variables, and note that $s \leq (n+2)^{2^{q'}}$. Here we view each T_i is a distribution over $\{D_j\}_{j\in[s]}$; that is, for every $i \in [M]$ there exists a distribution μ_i over $[M]$ such that for every $w \in \{0,1\}^n$, the output of T_i^w is obtained by drawing $j \sim \mu_i$ and outputting D_j^w. By Eq. (6.1), for every x and $i \in [M]$,

$$\sum_{j=1}^{s} \mu_i(j) \cdot \Pr_{x\in\{0,1\}^k}[D_j^{C(x)} = f_i(x)] \geq 1 - \delta_{\mathsf{err}}.$$

In particular, we obtain that for every $i \in [M]$ there exists $j \in [s]$ such that $\Pr_x[D_j^{C(x)} = f_i(x)] \geq 1 - \delta_{\mathsf{err}}$. Observe that if $M > s$ (i.e., there are more f_i's than depth-q' decision trees), then there exists $i_1, i_2 \in [M]$, where $i_1 \neq i_2$ and $j \in [s]$, such that $\Pr_x[f_{i_1}(x) = D_j^{C(x)} = f_{i_2}(x)] \geq 1 - 2\delta_{\mathsf{err}} = 1 - \beta$, in contradiction to the hypothesis. Thus $M \leq s \leq (n+1)^{2^{q'}}$, and since $q' = O(q)$, then $q = \Omega(\log\log M - \log\log n)$. □

On the gap between "pairwise far" and general families of functions. Recall that there is an additive difference of $\Omega(\log k)$ between the lower bound for general families of functions and the stronger lower bound for families of functions that are "pairwise far". We leave open the question of whether the lower bound for general families of functions can be improved to match the stronger lower bound for "pairwise far" functions, or

whether there exists a universal-LTC for a family of functions, which are *not* "pairwise far", that matches the lower bound for general functions. We point out two observations regarding the forgoing question:

1. The argument in the furthermore claim of Theorem 6.5 also shows that any universal-LTC with *deterministic* testers must satisfy $q = \Omega(\log\log M - \log\log n)$, even for families of functions that are not "pairwise far" and when given the proviso that the input is a valid codeword. Therefore, to construct a universal-LTC that matches the general lower bound, the testers must use a randomized strategy, not only for checking the validity of the encoding, but also for computing the function of the message. (We remark that all of the universal-LTCs in this chapter use randomness only for codeword testing.)

2. The proof of the furthermore claim of Theorem 6.5 actually yields a stronger statement regarding "pairwise far" functions. Specifically, it only requires that the functions should be "pairwise far" under *some* distribution (and not necessarily the uniform distribution); that is, it suffices that there exists a distribution $\mathcal{D}$ over $\{0,1\}^k$ such that $\Pr_{x\sim\mathcal{D}}[f_i(x) \neq f_j(x)] = \Omega(1)$ for every $i \neq j$.

6.6 Trading off Length for Locality

The general lower bound in Theorem 6.5 allows for a tradeoff between the universal-LTC's length and locality. We remark that while the canonical universal-LTC in Theorem 6.2 matches this lower bound, it is limited to the extreme end of the tradeoff, wherein the locality is minimized (i.e., the query complexity is constant). In this subsection we show a specific family of functions (namely, the family of m-juntas) for which we can obtain a smooth tradeoff between the universal-LTC's length and locality.

6.6.1 Universal-LTCs of Nearly-Linear Length

Let $m, k \in \mathbb{N}$ such that $m \leq k$, and denote by $\mathsf{Junta}_{m,k}$ the set of all $\binom{k}{m} \cdot 2^{2^m}$ k-variate Boolean functions that only depend on m coordinates. We start by showing that using super-constant query complexity, we can obtain universal-LTCs that are shorter than the canonical universal-LTC. More precisely, we prove that there exists a universal-LTC for $\mathsf{Junta}_{m,k}$ with linear distance, *nearly-linear length*, and query complexity that is quasilinear in m. (We discuss how this matches the lower bound in Theorem 6.5 in Section 6.6.3.)

Observation 6.6. *Let $k, m \in \mathbb{N}$ such $m \leq k$, and let $\alpha > 0$ be a constant. For every $\varepsilon > 0$ there exists a (one-sided error)* universal-LTC$_\varepsilon$ *$C : \{0,1\}^k \to \{0,1\}^{k^{1+\alpha}}$ for $\mathsf{Junta}_{m,k}$ with linear distance and query complexity $\tilde{O}(m) + \mathsf{poly}(1/\varepsilon)$.*

Sketch of proof. The idea is to use a code C that is both locally testable and decodable, and obtain a tester for each subcode $\{C(x) : f(x) = 1\}$ (where $f \in \mathsf{Junta}_{m,k}$) by invoking the tester for membership in C, using the decoder to recover the values of the

m influencing variables of f (for which we shall need to reduce the error probability of the decoder to $1/m$), and ruling accordingly. Recall, however, that there are no known LDCs with constant query complexity and polynomial length (let alone such with nearly-linear length). Instead, we observe that for the foregoing idea it suffices that C is a relaxed-LDC,[7] and so we can use the code in Theorem 6.1, which is both a (one-sided error) LTC and a relaxed-LDC, with nearly-linear length. The implementation of the aforementioned ideas is straightforward, and so, we omit it.

□

Digression: Universal-LTCs with optimal rate. In Observation 6.6, we are concerned with minimizing the locality of the universal-LTCs, while settling for nearly-linear length (and so, we use the code in Theorem 6.1 as the base code). We remark that the argument underlying Observation 6.6 holds for *any* base code that is both locally testable and (possibly relaxed) locally decodable. Thus, different base codes may be used to obtain universal-LTCs in other regimes. For example, allowing large query complexity (which depends on k) and focusing on optimizing the rate and the distance, we can obtain the following corollary by using the recent construction, due to Meir [Mei14, Theorem 1.1, 1.2, and Remark 1.5], of codes that are both locally testable and decodable with constant rate and optimal distance, and query complexity that is an arbitrary small power of the input length.

Corollary 6.7. *For every $0 < r < 1$, $\alpha, \beta > 0$ there exists a finite field H of characteristic 2 such that for every $m \leq k$, there exists a* universal-LTC $C : \mathbb{F}_2^k \to H^n$ *for* $\mathsf{Junta}_{m,k}$ *with rate at least r, relative distance at least $1 - r - \alpha$, and query complexity $O(k^\beta m \log m + k^\beta/\varepsilon)$.*[8]

6.6.2 The Actual Tradeoff

Next, we show a universal-LTC for $\mathsf{Junta}_{m,k}$ with a smooth tradeoff between length and query complexity.

Proposition 6.8. *Let $k, m \in \mathbb{N}$ such that $m \leq k$. For every $\tau < m$ and $\varepsilon \geq 1/\mathsf{polylog}(n)$, where $n \leq \frac{k^{m+1}}{k^t} \cdot (2^{2^m})^{1/2^t}$, there exists exists a (one-sided error)* universal-LTC$_\varepsilon$ $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n)}$ *for* $\mathsf{Junta}_{m,k}$ *with linear distance and query complexity $\tilde{O}(\tau) + O(1/\varepsilon)$.*

We remark that in the $\tilde{O}(m)$-locality regime (the query-heavy extreme of the tradeoff), Proposition 6.8 only yields a universal-LTC of quadratic length, whereas Observation 6.6 achieves nearly-linear length.[9]

[7] Recall that relaxed-LDCs are a relaxation of locally decodable codes that requires local recovery of individual information-bits, yet allow for recovery-failure, but not error, on the rest (see Definition 6.3).

[8] Recall that the query complexity measures the number of queries made, rather than the number of bits that were read, but since p is a constant, the difference is immaterial.

[9] It is possible to optimize Proposition 6.8 such that in the query-heavy extreme of the tradeoff it will yield universal-LTCs of linear length, by adapting techniques from [BSGH+06, Section 4] to our setting. However, this methodology is far more involved than simply using Observation 6.6 in the $\tilde{O}(m)$-locality regime.

Sketch of proof. The basic idea is to map $x \in \{0,1\}^k$ to the long code encoding of the *projection* of x to each m-subset of $[k]$; that is, $x \to \big(\mathsf{LC}(x|_{S_1}), \ldots, \mathsf{LC}(x|_{S_N})\big)$, where $S_1, \ldots, S_N$ are all $N = \binom{m}{k}$ distinct m-subsets of $[k]$ and $\mathsf{LC} : \{0,1\}^m \to \{0,1\}^{2^{2^m}}$ is the corresponding long code.

Next, to ascertain that all the long code encodings are consistent with restrictions of a single x, we bundle these encodings with PCPs according to the consistency-testable bundling mechanism presented in Section 7.4.1 (where the encodings $\{E_i\}$ correspond to $\{\mathsf{LC}(x|_{S_i})\}$). This yields a universal-LTC *for m-juntas* with query complexity $O(1)$ and length $\binom{k}{m} \cdot \tilde{O}(2^{2^m} + k)$: To test that x satisfies the junta $f(x) = f'(x|_S)$, where $S \subseteq [k]$ such that $|S| = m$, we first use Proposition 7.7 to ensure the consistency of the bundle (i.e., the consistency of f with the anchor), then we extract the value of $f(x)$ by locally correcting the point that corresponds to f' in the purported copy of $\mathsf{LC}(x|_S)$.

Finally, to obtain a smooth tradeoff, we modify the foregoing construction such that x is mapped to the long code encoding of the projection of x to each $(m-\tau)$-subset of $[k]$ (instead of m-subset), for the given parameter $\tau \in [m]$. The idea is that now, to test that x satisfies $f'(x|_S) = 1$, we first arbitrarily choose t bits of $x|_S$ and decode them one-by-one (as in Observation 6.6); this induces a function f'' on the remaining $m - \tau$ bits, which we compute by self-correcting the single bit that corresponds to f'' in the long code encoding of x projected to these $m - \tau$ bits. The implementation of the foregoing ideas is straightforward and is presented in Section 6.7.2. □

6.6.3 Lower Bounds for Universal-LTCs for Juntas

We conclude this subsection by proving a lower bound on the query complexity of universal-LTCs for $\mathsf{Junta}_{m,k}$. Observe that the family of all m-juntas do *not* satisfy the "pairwise far" condition, and thus Theorem 6.5 only gives us a lower bound of $q \geq m - \log\log n - O(\log k)$. However, we show that while the family $\mathsf{Junta}_{m,k}$ is not "pairwise far", it contains a dense subset of functions that are "pairwise far", and so we can strengthen the foregoing lower bound as follows.

Proposition 6.9. *Let $k, m \in \mathbb{N}$ such $m \leq k$. There exists a universal constant $c > 0$ such that every* universal-LTC$_\varepsilon$ $C : \{0,1\}^k \to \{0,1\}^n$ *for* $\mathsf{Junta}_{m,k}$ *with linear distance and $\varepsilon < \delta(C)/2$ must have query complexity $\Omega(m - \log\log(n) + c)$.*

We remark that that for $m = \big(1 + \Omega(1)\big) \cdot \log\log(n)$, the lower bound simplifies to $\Omega(m)$ and matches Observation 6.6 up to a constant power. Furthermore, it is possible to improve Proposition 6.9 such that it gives a non-trivial lower bound when $m < \log\log(k)$ (see discussion at the end of the section).

Proof of Proposition 6.9. We show that $\mathsf{Junta}_{m,k}$ contains a dense subset that is "pairwise far". Specifically, fix $S \subseteq [k]$ such that $|S| = m$, and let $\mathsf{Junta}^S_{m,k} \subseteq \mathsf{Junta}_{m,k}$ denote all m-juntas that depend only on coordinates in S. We prove that there exists a family $\mathcal{F} \subseteq \mathsf{Junta}^S_{m,k}$ of $M = 2^{\Omega(2^m)}$ distinct functions such that every distinct f and g in $\mathcal{F}$ satisfies $\Pr_{x \in \{0,1\}^k}[f(x) \neq g(x)] = \Omega(1)$.

Note that the set of truth tables, restricted to inputs supported on S, of all $f \in \mathsf{Junta}^S_{m,k}$ is isomorphic to $\{0,1\}^{2^m}$, and thus we can choose a subset of it that constitutes a good code. That is, for every $f \in \mathsf{Junta}^S_{m,k}$, note that $f(x) = f'(x|_S)$ for some $f' : \{0,1\}^m \to \{0,1\}$, and denote the truth table of f' by $\langle f' \rangle$. Let C_0 be a code with linear distance, constant rate, and codewords of length 2^m, and observe that by the rate and distance of the code C_0, the set $\mathcal{F} = \{f \in \mathsf{Junta}^S_{m,k} : \langle f' \rangle \in C_0\}$ is a collection of $2^{\Omega(2^m)}$ functions such that every distinct $f, g \in \mathcal{F}$ satisfy

$$\Pr_{x \in \{0,1\}^k}[f(x) \neq g(x)] = \Pr_{\substack{x \in \{0,1\}^k \\ x|_{[k]\setminus S} = \mathbf{0}}}[f(x) \neq g(x)] = \Omega(1).$$

The proof of Proposition 6.9 is concluded by applying Theorem 6.5 to $\mathcal{F}$. □

Improving the lower bound. We point out a slackness in the proof of Proposition 6.9. Specifically, we apply Theorem 6.5 to a subset $\mathcal{F}$ of m-juntas that depend on a *single* set $S \subset [k]$ of cardinality m, and so we lose all dependency in k (the dimension of the code). We sketch below how to tighten this slackness and obtain a slightly stronger lower bound of $\Omega(\max\{m, \Omega(\log(m)) + \log\log(k)\} - \log\log(n))$, which gives a non-trivial lower bound also when $m < \log\log(k)$ and $n < k^m$ (while noting that Proposition 6.9 trivializes for this range of parameters).

As a first attempt, we can consider a *partition* of $[k]$ to sets $S_1, \ldots, S_{k/m}$ of cardinality m, and (similarly to Proposition 6.9) include in $\mathcal{F}$ a subset of functions from each $\mathsf{Junta}^{S_i}_{m,k}$ whose truth-tables form a good code. Inspection shows that as long as the foregoing good code is *balanced*,[10] juntas in such $\mathcal{F}$ are pairwise far, and so we can apply Theorem 6.5. The problem is, however, that such argument only strengthens the lower bound by a constant factor; that is, it yields $q = \Omega\left(\log\log\left(\frac{k}{m} \cdot 2^{2^m}\right) - \log\log n\right)$, which is not asymptotically better than $q = \Omega(\log\log(2^{2^m}) - \log\log n)$, established in Proposition 6.9.

To obtain an asymptotical strengthening, we can choose $k^{\Omega(m)}$ distinct subsets of $[k]$ with small (say, $m/100$) pairwise intersection (using the Nisan-Wigderson combinatorial designs [NW94]), and for each such subset S, include in $\mathcal{F}$ juntas from $\mathsf{Junta}^S_{m,k}$ whose truth-tables form a *random* code. On inspection, it turns out that juntas in such $\mathcal{F}$ are pairwise far, and thus we can apply Theorem 6.5 to obtain $q = \Omega(\log\log(k^m \cdot 2^{2^m}) - \log\log n)$, which yields the aforementioned bound.

[10] That is, a code wherein each codeword consists of an equal number of 0's and 1's.

6.7 Appendices for Chapter 6

6.7.1 On Obtaining Locally Decodable Codes from Universal-LTCs

In this appendix we show that universal-LTCs for the family of linear functions (and more generally, for self-correctable families of functions) imply local decodability in the strong (non relaxed) sense. More accurately, denote the set of all k-variate linear functions over $\mathsf{GF}(2)$ by Linear_k. The following theorem shows that any universal-LTC for Linear_k implies a LDC with roughly the same parameters.

Theorem 6.8. *If there exists an* universal-LTC C *for* Linear_k *with linear distance, rate* r*, and query complexity* $q = q(\varepsilon)$*, then there exists a binary* LDC *with linear distance, rate* $\Omega(r)$*, and query complexity* $O(1)$*.*

Proof. Fix $\varepsilon = \delta(C)/3$. For every linear function $f \in \mathsf{Linear}_k$ and $b \in \{0,1\}$, let $T_{f,b}$ be the ε-tester for the subcode $\Pi_{f,b} \stackrel{\text{def}}{=} \{C(x) : f(x) = b\}$ guarantied by the universal-LTC $C : \{0,1\}^k \to \{0,1\}^n$. These testers admit a natural candidate for a local decoding procedure: to decode x_i, simply invoke $T_{f,0}$ and $T_{f,1}$ for the linear function $f(x) = x_i$, and rule according to the tester that accepted.

The problem is that given a slightly corrupted copy of $C(x)$, the testers $T_{f,0}$ and $T_{f,1}$ may both reject, since they are not necessarily *tolerant*;[11] in this case we cannot decode. (Indeed, if the aforementioned testers are tolerant, then the foregoing procedure is a local decoder.[12]) Nevertheless, since the foregoing case only happens when the input is *not* a valid codeword, we obtain a procedure that either decodes correctly or detects a corruption in the encoding and aborts (similarly to relaxed-LDCs, see Definition 6.3). Then, by slightly modifying the code, we can bound the number of linear functions on which we are forced to abort and use the linear functions that we are able to compute to recover *any* linear function, including $f(x) = x_i$. Details follow.

Assume without loss of generality that the testers of the universal-LTC have soundness error of at most $1/10$. Consider the algorithm $\mathcal{A}$ that, given $f \in \mathsf{Linear}_k$ and oracle access to $w \in \{0,1\}^n$, invokes $T_{f,0}$ and $T_{f,1}$ on w; if one tester accepted and the other rejected, $\mathcal{A}$ rules according to the accepting tester, and otherwise it outputs $\perp$. Hence, $\mathcal{A}$ has query complexity $O(q(\varepsilon)) = O(1)$. The following claim shows that indeed $\mathcal{A}$ succeeds in locally computing $f(x)$ in the following sense (which is analogous to that of relaxed-LDCs).

Claim 6.9.1. *For every* $f \in \mathsf{Linear}_k$*, the algorithm* $\mathcal{A}$ *satisfies the following two conditions.*

1. *If* $w = C(x)$ *for some* $x \in \{0,1\}^k$*, then* $\Pr\left[\mathcal{A}^{C(x)}(f) = f(x)\right] \geq 2/3$.

[11] Recall that tolerant testers accept strings that are (say) $\delta(C)/3$-close to being valid and reject strings that are (say) $\delta(C)/2$-far from being valid (with high probability).

[12] In fact, the argument above shows that a tolerant universal-LTC for *any* family of functions $\mathcal{F}$ that contain the dictator functions, i.e., such that $\{f(x) = x_i\}_{i \in [k]} \subseteq \mathcal{F}$, implies a LDC with roughly the same parameters.

2. If w is $\delta(C)/3$-close to a codeword $C(x)$, then $\Pr\left[\mathcal{A}^w(f) \in \{f(x), \perp\}\right] \geq 2/3$.

Proof. Let $w = C(x)$ for $x \in \{0,1\}^k$ such that $f(x) = 1$ (the case in which $f(x) = 0$ is symmetrical). Since $T_{f,1}$ is a tester for $\Pi_{f,1} \stackrel{\text{def}}{=} \{C(x) : f(x) = 1\}$, then $\Pr[T^w_{f,1} = 1] \geq 9/10$, and since $T_{f,0}$ is a $\delta(C)/3$-tester for $\Pi_{f,0} \stackrel{\text{def}}{=} \{C(x) : f(x) = 0\}$ and w is $\delta(C)$-far from $\Pi_{f,0}$, then $\Pr[T^w_{f,0} = 0] \geq 9/10$. Thus, by the definition of $\mathcal{A}$ it holds that $\Pr\left[\mathcal{A}^w(f) = f(x)\right] \geq (9/10)^2$. Next, assume that w is $\delta(C)/3$-close to a codeword $C(x)$ such that $f(x) = 1$ (again, the case in which $f(x) = 0$ is symmetrical). Then, $\Pr[T^w_{f,0} = 1] < 1/10$ and $\Pr\left[\mathcal{A}^w(f) \in \{f(x), \perp\}\right] \geq 9/10$ follows. □

The second condition of Claim 6.9.1 does not bound the number of linear functions on which the algorithm $\mathcal{A}$ is allowed to abort (and so, given a corrupted codeword, $\mathcal{A}$ can potentially output $\perp$ on all inputs). However, by adapting of the techniques of Ben-Sasson et al. [BSGH+06, Lemmas 4.9 and 4.10] to the setting of universal-LTCs, we obtain the following claim, which shows that C and $\mathcal{A}$ can be modified to allow for such bound.

Claim 6.9.2. *If there exists a code $C : \{0,1\}^k \to \{0,1\}^n$ with distance d and rate r, and an algorithm $\mathcal{A}$ with query complexity q, which satisfies the conditions of Claim 6.9.1, then there exists a constant $\delta_{\mathsf{radius}} > 0$, a code $C' : \{0,1\}^k \to \{0,1\}^{n'}$ with distance $\Theta(d)$ and rate $\Theta(r)$, and an algorithm $\mathcal{B}$ that for every (explicitly given) $f \in \mathsf{Linear}_k$ makes $O(q)$ queries to a string $w \in \{0,1\}^{n'}$ and satisfies the following condition: If w is δ_{radius}-close to a codeword $C'(x)$, then there exists a family $\mathcal{F}$ of at least $(9/10) \cdot 2^k$ functions in Linear_k such that for every $f' \in \mathcal{F}$ it holds that $\Pr\left[\mathcal{B}^w(f') = f'(x)\right] \geq 9/10$.*

We omit the proof of Claim 6.9.2, since it follows by a trivial adaptation of [BSGH+06, Lemmas 4.9 and 4.10] to our setting. We mention that the main idea is that by repeating heavily probed locations in the code, we can modify $\mathcal{A}$ such that on an average f it make queries that are nearly uniformly, and then use this "average smoothness" to bound the fraction of functions on which we are forced to abort.

The proof of Theorem 6.8 follows by noting that given a slightly corrupted copy of $C'(x)$, for every $f \in \mathsf{Linear}_k$ we can use the algorithm $\mathcal{B}$ of Claim 6.9.2 to extract the value of $f(x)$ using the self correctability of linear functions. In more detail, let $w \in \{0,1\}^{n'}$ such that $\delta(w, C'(x)) \leq \delta_{\mathsf{radius}}$ for some $x \in \{0,1\}^k$, and let $i \in [k]$. To decode x_i, we uniformly choose $g \in \mathsf{Linear}_k$, invoke $\mathcal{B}^w(g)$ and $\mathcal{B}^w(g+x_i)$, and output $\mathcal{B}^w(g)+\mathcal{B}^w(g+x_i)$. By the union bound, with probability at least $1 - 2/10$ both g and $g + x_i$ are functions on which $\mathcal{B}$ succeeds with probability at least $9/10$. Thus, with probability at least $(8/10) \cdot (9/10)$, both $\mathcal{B}^w(g) = g(x)$ and $\mathcal{B}^w(g + x_i) = g(x) + x_i$, and so their summation (over $\mathsf{GF}(2)$) is x_i. □

Generalizing to Self-Correctable Families of Functions. We remark that the only place in which the proof of Theorem 6.8 relies on $\mathcal{F}$ being the family of all linear functions is that the latter family admits self correction. Therefore, the same proof holds for any family of functions $\mathcal{F} = \left\{f_i + b : \{0,1\}^k \to \{0,1\}\right\}_{i \in [M], b \in \{0,1\}}$ that is self correctable.

6.7.2 Proof of Proposition 6.8

Let $k, m \in \mathbb{N}$ such that $m \leq k$. We show that for every $\tau < m$ and $\varepsilon \geq 1/\mathsf{polylog}(n)$, where $n = \binom{k}{m-\tau} \cdot \max\{2^{2^{m-\tau}}, k\}$, there exists exists a (one-sided error) $\mathsf{universal\text{-}LTC}_\varepsilon$ $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n)}$ for $\mathsf{Junta}_{m,k}$ with linear distance and query complexity $\tilde{O}(\tau)+O(1/\varepsilon)$.

Let $\tau < m$ and $\varepsilon \geq 1/\mathsf{polylog}(n)$. We bundle the long code encoding of each projection of x to $(m-\tau)$ coordinates; that is, denote the $(m-\tau)$-dimensional long code by $\mathsf{LC} : \{0,1\}^{m-\tau} \to \{0,1\}^{2^{2^{m-\tau}}}$, denote the set of all subsets of $[k]$ of cardinality $m-\tau$ by $\mathcal{S}^{(m-\tau)} = \{S' \subseteq [k] \ : \ |S'| = m-\tau\}$. We bundle the encodings $\{\mathsf{LC}(x|_{S'})\}_{S' \in \mathcal{S}^{(m-\tau)}}$ according to Construction 6.4.

Recall that in Construction 6.4 we bundle encodings $E_i, \ldots, E_s$ with an (arbitrary) error-correcting code ECC (which can be encoded by a circuit of quasilinear size in k and has linear distance) and with a PCPP for every E_i, which ascertains that a pair (a,b) satisfies $a = \mathsf{ECC}(y)$ and $b = E_i(y)$ for some y. Here, the encodings will correspond to the long code encodings of x projected to $(m-\tau)$-subsets in $\mathcal{S}^{(m-\tau)}$, i.e., $\{\mathsf{LC}(x|_{S'})\}_{S' \in \mathcal{S}^{(m-\tau)}}$. Note that each $\mathsf{LC}(x|_{S'})$ can be computed by a circuit of size $O(2^{2^m} \cdot m) = \tilde{O}(n)$. Hence, by Theorem 7.1, for every $S' \in \mathcal{S}^{(m-\tau)}$ there exist a PCPP oracle $\bar{p}_{S'}$, as required in Construction 6.4, of length $\tilde{O}(n)$. We obtain the code $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n)}$ given by

$$C(x) = \Big(\mathsf{ECC}(x)^r, \big(\mathsf{LC}(x|_{S'})\big)_{S' \in \mathcal{S}^{(m-\tau)}}, \big(\bar{p}_{S'}(x)\big)_{S' \in \mathcal{S}^{(m-\tau)}}\Big). \tag{6.2}$$

We show that C is a $\mathsf{universal\text{-}LTC}_\varepsilon$ for $\mathsf{Junta}_{m,k}$ with query complexity $\tilde{O}(\tau) + O(1/\varepsilon)$.

Fix $\varepsilon > 0$, $f \in \mathsf{Junta}_{m,k}$, and write $f(x) = f'(x|_S)$, where S denotes the m influencing coordinates of f. Denote by T the first τ coordinates in S. For every $i \in T$, let $S'_i \in \mathcal{S}^{(m-\tau)}$ be a $(m-\tau)$-subset that contains i. Denote by D the $O(1)$-query corrector of the long code. Using amplification, assume that the corrector D and the bundle consistency-test (see Proposition 7.7) make at most $O(\log(\tau))$ and $O(\log(\tau)/\varepsilon)$ queries (respectively) and obtain soundness error that is (strictly) less than $1/(10(\tau+1))$.

Consider the ε-tester T_f for the subcode $\Pi_f = \{x \in \{0,1\}^k \ : \ f(x) = 1\}$, which has oracle access to a purported bundle $w \in \{0,1\}^{\tilde{O}(n)}$ that is supposed to equal Eq. (6.2); that is, w allegedly consists of three parts: (1) the purported anchor $\widetilde{\mathsf{ECC}}(x)$, (2) the purported long code encodings $\big(\widetilde{\mathsf{LC}}(x|_{S'})\big)_{S' \in \mathcal{S}^{(m-\tau)}}$, and (3) the purported PCPs of proximity $\big(\widetilde{\bar{p}}_{S'}(x)\big)_{S' \in \mathcal{S}^{(m-\tau)}}$. Note that we use $\widetilde{z}$ to denote a string that is allegedly equal to z. The tester T_f operates as follows:

1. $\mathsf{Consistency\ Test}$: Invoke the bundle consistency test on w, with respect to proximity parameter ε and the purported encoding $\widetilde{\mathsf{LC}}(x|_{S \setminus T})$, as well as $\widetilde{\mathsf{LC}}(x|_{T_i})$, for every $i \in T$. Reject if any of the tests fail. (The query complexity of this step is $O(\tau \cdot \log(\tau)/\varepsilon)$.)

2. $\mathsf{Direct\ recovery\ of}$ t $\mathsf{variables}$: Decode $x|_T$ using the self correction of the long code; that is, for every $i \in T$ decode x_i from $\widetilde{\mathsf{LC}}(x|_{S'_i})$ (recall that S'_i is a $(m-\tau)$-subset that contains i), using the corrector D. Denote the string of recovered values by z. (The query complexity of this step is $O(\tau \cdot \log(\tau))$.)

3. Computing the induced $(m-\tau)$-junta: Choose $f'' : \{0,1\}^{m-\tau} \to \{0,1\}$ such that $f''(y) = f'(z \circ y)$, decode f'' from the purported long code encoding $\widetilde{\mathsf{LC}}(x|_{S\setminus T})$ using the corrector D, and accept if and only if it returns 1. (The query complexity of this step is $O(\log(\tau))$.)

The perfect completeness of T_f follows by the one-sided error of the bundle consistency test and the long code corrector D. For the soundness, assume that w is ε-far from Π_f. By Proposition 7.7, we can assume that there exists $y \in \{0,1\}^k$ such that w is ε-close to $C(y)$, and since w is ε-far from Π_f, it holds that $f(y) = 0$; furthermore, $\widetilde{\mathsf{LC}}(y|_{S\setminus T})$ is ε-close to $\mathsf{LC}(y|_{S\setminus T})$, and each $\widetilde{\mathsf{LC}}(y|_{S'_i})$ is ε-close to $\mathsf{LC}(y|_{S'_i})$, otherwise the bundle consistency test rejects with probability at most $(\tau+1)/(10(\tau+1))$. Thus, in Step 2, the corrector D successfully recovers $y|_T$ with probability $(1/10)\cdot\tau/(10(\tau+1))$, and so, with probability at least $2/3$, in Step 3 the tester T_f correctly computes $f''(y|_{S\setminus T}) = f'(y|_T \circ y|_{S\setminus T}) = f(y) = 0$ and rejects. This concludes the proof of Proposition 6.8.

Chapter 7

Universal Locally Verifiable Codes and 3-Round Interactive Proofs of Proximity for CSP

7.1 Introduction

Locally testable codes [FS95, RS96, GS06] are codes admitting local procedures for checking the validity of alleged codewords. A code C is a locally testable code (LTC) if there exists a randomized testing algorithm that receives a proximity parameter $\varepsilon > 0$, makes a small number of queries to a string w, and with high probability accepts if w is a codeword of C and rejects if w is ε-far from C. The query complexity (or locality) of the tester is the number of queries that it makes.

In our companion work [GG16a], we initiated a study of a generalization of the notion of LTCs, called universal locally testable codes. A universal-LTC is a code that not only admits a local test for membership in the code C but also a local test for membership in a *family of subcodes of* C. More specifically, a binary code $C : \{0,1\}^k \to \{0,1\}^\eta$ is a q-local universal-LTC for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ if for every $i \in [M]$ the subcode $\Pi_i \stackrel{\text{def}}{=} \{C(x) \ : \ f_i(x) = 1\}$ is locally testable with query complexity q. Viewed in an alternative perspective, such codes allow for testing properties of the encoded *message*; that is, testing whether $C(x)$ is an encoding of a message x that satisfies a function $f_i \in \mathcal{F}$.

7.1.1 The Notion of Universal Locally Verifiable Codes

In this chapter, we consider the NP proof system analogue of universal-LTCs, in which the testing procedures are replaced with *verification* procedures that are given free access to a short (sublinear length) proof. We call such codes "universal locally *verifiable* codes" (universal-LVCs). One may hope that *verification* of membership in subcodes can be done more efficiently than *testing*, and indeed we will show that universal-LVCs can be much

more powerful than universal-LTCs.

To define the notion of universal-LVC, we recall the notion of non-interactive proofs of proximity [GR13b]. A property Π is said to have an MA *proof of proximity* (MAP) if there exists a probabilisitic algorithm (verifier) V that gets a proximity parameter $\varepsilon > 0$ and a short (sublinear)[1] proof π as well as oracle access to a string w. The verifier satisfies, with high probability, the following conditions: If $w \in \Pi$, there exists proof π such that $V^w(\pi, \varepsilon)$ accepts, and if w is ε-far from Π, then for every alleged proof π, the verifier $V^w(\pi, \varepsilon)$ rejects.

We say that a code $C : \{0,1\}^k \to \{0,1\}^\eta$ is a *universal locally verifiable code* (universal-LVC), with proof length p and query complexity q, for a family of functions $\mathcal{F} = \{f_i : \{0,1\}^k \to \{0,1\}\}_{i \in [M]}$ if for every $i \in [M]$ the subcode $\Pi_i \stackrel{\text{def}}{=} \{C(x) : f_i(x) = 1\}$ has an MAP with proof length p and query complexity q.

7.1.2 Our Results

To simplify the presentation of our results, throughout the introduction we fix the proximity parameter ε to a small constant, and when we refer to "codes", we shall actually mean error-correcting codes with linear distance.

We show quadratic length universal-LVCs of sublinear proof and query complexity for a large and natural complexity class, for which every polynomial length universal-LTC must have almost linear query complexity. Specifically, let $n \geq k$, and denote by $\mathsf{CSP}_{n,k}$ the set of all instances of *constraint satisfaction problems* with n constraints of constant arity over k variables.

Theorem 5 (informally stated, see Theorem 7.2)**.** *For all $k \leq n$, there exists a* universal-LVC *$C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(n^2)}$ for $\mathsf{CSP}_{n,k}$ with proof and query complexity $\tilde{O}(n^{2/3})$. More generally, for every $\alpha > 0$ it is possible to obtain proof length $\tilde{O}(n^{2\alpha})$ and query complexity $\tilde{O}(n^{1-\alpha})$.*

In contrast, as stated above, every polynomial length universal-LTC for $\mathsf{CSP}_{n,k}$ has query complexity that is roughly linear in k. Actually, we provide a lower bound on the tradeoff between the two complexity measures of universal-LVCs for $\mathsf{CSP}_{n,k}$.

Theorem 6 (informally stated, see Corollary 7.5)**.** *For all $k \leq n$ and every polynomial (in k) length* universal-LVC *for $\mathsf{CSP}_{n,k}$ with proof complexity $p \geq 1$ and query complexity q it holds that $p \cdot q = \widetilde{\Omega}(k)$. For $p = 0$ (i.e., a* universal-LTC*), the query complexity is $\widetilde{\Omega}(k)$.*

Note that for $n = \tilde{\Theta}(k)$, Theorem 5 gives a universal-LVC of length $\tilde{O}(k^2)$, with proof and query complexity $\tilde{O}(k^{2/3})$ each, whereas Theorem 6 shows that such a universal-LVC (of length $\mathsf{poly}(k)$) must have either query or proof complexity $\widetilde{\Omega}(\sqrt{k})$.

[1] We remark that if we do not restrict the length of the proof, then *every* property Π can be verified trivially using only a constant amount of queries, by considering an MAP proof that contains a full description of the input and testing identity between the proof and the input.

7.1.3 An Application for Interactive Proofs of Proximity

An *interactive proof of proximity* (IPP), as defined in [RVW13], can be thought of as a generalization of the notion of MAP in which the verifier is allowed to interact with an omniscient prover (instead of a "static" proof). Hence, an IPP is an interactive proof system wherein an all powerful (yet untrusted) prover interacts with a verifier that only has oracle access to an input x. The prover tries to convince the verifier that x has a particular property Π. Here, the guarantee is that for inputs in Π, there exists a prover strategy that will make the verifier accept with high probability, whereas for inputs that are far from Π the verifier will reject with high probability no matter what prover strategy is employed.[2]

Rothblum et al. [RVW13] showed that, loosely speaking, every language in NC has an IPP with query and communication complexities $\tilde{O}(\sqrt{n})$, albeit this IPP requires a large ($\mathsf{polylog}(n)$) number of rounds of interaction. For IPPs that use a small number of rounds of interaction (in particular, MAPs) only results for much lower complexity classes are known (e.g., for context-free languages and languages that are accepted by small read-once branching programs [GGR15]).

We show that the universal-LVC in Theorem 5 can be, in a sense, "emulated" using a small (constant) amount of interaction rounds. This yields the following IPP.

Theorem 7 (informally stated, see Theorem 7.7)**.** *Let $n \geq k$. For every $\varphi \in \mathsf{CSP}_{n,k}$ there exists a 3-round* IPP *for the property $\Pi_\varphi \stackrel{\text{def}}{=} \{x \in \{0,1\}^k : \varphi(x) = 1\}$ with communication and query complexity $n^{6/7+o(1)}$. More generally, there exists an $O(1)$-round* IPP *for Π_φ with communication and query complexity $n^{0.501}$.*

We mention that, for some φ's, testing the property Π_φ requires a linear number of queries to test [BHR05]. We stress that our IPPs are for the set of satisfying assignments of fixed CSP instances, whereas the IPPs in [RVW13, RRR16] are for sets that are in a (uniform) complexity class.[3]

Related Work. Independently of this work, Reingold, Rothblum, and Rothblum [RRR16] showed that for every sufficiently small constant $\sigma \in (0,1)$, there exists an $2^{\tilde{O}(1/\sigma)}$-round IPP, with query and communication complexity $n^{0.5+O(\sigma)}$, for any language that is computable in $\mathsf{poly}(n)$-time and $O(n^\sigma)$-space.

The notion of universal-LVCs is closely related to that of (1-message) holographic interactive proofs (HIP) (which naturally adapts the definition of holographic proofs [BFLS91] to the setting of interactive proofs). An HIP is an interactive proof in which, instead of getting its input x explicitly, the verifier is given *oracle* access to $C(x)$, an error-corrected encoding of the input x, for a bounded number of queries. Hence, HIPs may

[2] Indeed, MAPs can be thought of as a restricted case of IPPs, in which the interaction is limited to a single message sent from the prover to the verifier.

[3] That is, our IPPs are for massively parameterized properties (as surveyed in [New10]): We consider a family of properties $\{\Pi_\varphi\}_{\varphi \in \mathsf{CSP}_{n,k}}$ that are parameterized by CSP formulas of size that is similar to the input's size. Likewise, the IPPs for read-only branching programs in [GGR15] are massively parameterized, but the IPPs for context-free languages are not.

be thought of as interactive proofs for promise problems of the form $(\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ with $\Pi_{\mathsf{YES}} = \{C(x) : x \in \mathcal{L}\}$ and $\Pi_{\mathsf{NO}} = \{C(x) : x \notin \mathcal{L}\}$. We stress that an HIP is given the *promise* that its input is properly encoded, whereas for universal-LVC, the verifier is required to *test* that its input is properly encoded.

7.1.4 Our Techniques

In this section we provide a high-level overview of the key ideas underlying Theorem 5, which shows a universal-LVC for CSP. For an overview of our application to proofs of proximity (Theorem 7), we refer the reader to Section 7.6.1. The lower bound in Theorem 6 follows by a simple application of the "communication complexity method" of Blais et. al. [BBM11], as extended to MAPs in [GR13b] (see Section 7.5 for details).

In the following, we assume basic familiarity with algebraic PCP systems. Our general approach follows the arithmetization paradigm, commonly used in many probabilistic proof systems. However, for reasons detailed next, we cannot use the standard arithmetizations used in the PCP literature. We focus on the first step of arithmetization, which is over the integers, and assume for simplicity that only one type of t-ary constraint, denoted c, is used.

The most common arithmetization, which can be traced back to [FGL+91], represents the t-ary instance φ as a *generic* function $\phi : [k]^t \to \{0,1\}$ such that $\phi(i_1, \ldots i_t) = 1$ if and only if the i'th constraint of φ involves the variables $x_{i_1} \ldots, x_{i_t}$. The satisfiability of φ at x is then given by

$$\sum_{i_1,\ldots i_t \in [k]} \phi(i_1, \ldots i_t) \cdot \mathsf{c}(x_{i_1}, \ldots x_{i_t}) = n. \tag{7.1}$$

This leads to a PCP oracle of length at least k^t, and at best we can hope to implement it by a universal-LVC that has proof length p and query complexity q such that $p \cdot q \geq k^t$. Our goal is, however, to get both p and q to the sublinear (in k) level.

The large PCP length of Eq. (7.1) lead [BFLS91] to suggest a different representation. Using a *universal* circuit ϕ of size $n' = \tilde{O}(n)$, the satisfiability of φ at x is represented by

$$\exists y \in \{0,1\}^{n'} \quad \sum_{i \in [k+n']} \phi(i) \cdot \mathsf{c}'\big((xy)|_{S_i}\big), \tag{7.2}$$

where c' is a fixed condition (which depends on c) and each $S_i \subseteq [k+n']$ is a subset of constant cardinality. The problem with Eq. (7.2) is that y is a sequence of auxiliary variables and its assignment in Eq. (7.2) depends on the instance φ (and not only on the assignment x).

Our alternative arithmetization composes the assignment $x \in \{0,1\}^k$ viewed as a function $x : [k] \to \{0,1\}$ with functions $\varphi_1, \ldots, \varphi_t : [n] \to [k]$ that represent the instance φ. Specifically, $\varphi_j(i) = i'$ if $x_{i'} = x(i')$ is the j'th variable of the i'th constraint of φ. Hence, φ is satisfiable if and only if

$$\sum_{i \in [n]} \mathsf{c}\big(x \circ \varphi_1(i), \ldots, x \circ \varphi_t(i)\big) = n. \tag{7.3}$$

Next, we consider the algebraic representation of Eq. (7.3) over a sufficiently large finite field $\mathbb{F}$ (discussed below). For simplicity, we assume throughout the rest of this overview that $n = k$, $m = O(1)$ and $t = O(1)$. We identify $[n]$ (the number of constraints) with some set H^m, where $H \subset \mathbb{F}$. Throughout this chapter, we shall denote the low-degree extension of a function f by $\widehat{f}$. Let $\widehat{\varphi}_j : \mathbb{F}^m \to \mathbb{F}^m$ and $\widehat{X} : \mathbb{F}^m \to \mathbb{F}$ be the individual degree $n^{1/m}$ extensions of $\varphi_j : H^m \to H^m$ and the assignment $X : H^m \to \{0,1\}$ (respectively), and let $\widehat{\mathsf{c}} : \mathbb{F}^t \to \mathbb{F}$ be the degree t *multilinear* extension of the constraint $\mathsf{c} : \{0,1\}^t \to \{0,1\}$. Note that $\varphi(x) = 1$ if and only if

$$\sum_{z_1,\ldots,z_m \in H} \widehat{\mathsf{c}}\big(\widehat{X} \circ \widehat{\varphi}_1(z_1,\ldots,z_m),\ldots,\widehat{X} \circ \widehat{\varphi}_t(z_1,\ldots,z_m)\big) = n.$$

The straightforward way to implement an MAP for such arithmetization is as follows. Let $\ell \in [m/2]$ be a parameter that will be used to control a tradeoff between proof and query complexity. The purported proof for the MAP is the polynomial

$$\pi(z_1,\ldots,z_\ell) = \sum_{z_{\ell+1},\ldots,z_m \in H} \widehat{\mathsf{c}}\big(\widehat{X} \circ \widehat{\varphi}_1(z_1,\ldots,z_m),\ldots,\widehat{X} \circ \widehat{\varphi}_t(z_1,\ldots,z_m)\big), \tag{7.4}$$

specified by its coefficients. Observe that the total degree of both $\widehat{X}$ and $\widehat{\varphi}_j$ is $m \cdot |H| = m \cdot n^{1/m}$ and that the composition of $\widehat{X}$ with φ_j increases the total degree to $m^2 \cdot |H|^2$. Note this is in contrast to standard arithmetizations, wherein typically the degree of the proof polynomial is $\tilde{O}(|H|)$. In addition, note that $\widehat{\mathsf{c}}$ only contributes a factor of t to the degree of π, since the constraint is Boolean, and so we can take its *multilinear* extension (saving an $\mathsf{exp}(t)$ factor that would have arisen had we constructed a $\mathsf{universal\text{-}LVC}$ for CNF formulas and use reductions to handle general t-ary CSPs.) Observe that the proof length of such MAP is bounded by $\deg(\pi)^\ell \cdot \log|\mathbb{F}| = t^\ell \cdot (m \cdot |H|)^{2\ell} \cdot \log|\mathbb{F}|$ (where $\deg(\pi)$ is the total degree of π, which equals $tm^2 \cdot |H|^2$).

Given the foregoing alleged proof π, the verifier can check that $\sum_{z_1,\ldots z_\ell \in H} \pi(z_1,\ldots,z_\ell) = n$. Thus, ascertaining the validity of the proof reduces to computing π at a random point $r \in \mathbb{F}^\ell$ and comparing it to the right hand side of Eq. (7.4). Recall that the formula φ is hardcoded in the verifier, and so it remains for the verifier to query $\widehat{X} \circ \widehat{\varphi}_j(r, z')$ at all $z' \in H^{m-\ell}$ (which is actually done via self-correction, preceded by a low-degree test). Therefore, it suffices to set the $\mathsf{universal\text{-}LVC}$ to $\widehat{X}$, the low-degree extension of the assignment (which does not depend on the formula). Observe that the query complexity of such MAP is $t \cdot n^{1-\frac{\ell}{m}} \cdot \log|\mathbb{F}|$ (which is primarily determined by the number summands in π).

Unfortunately, a straightforward application of the MAP above requires the order of the field $\mathbb{F}$ (to which we extend) to be greater than the sum we are checking (i.e., n, the number of constraints), because we cannot afford taking a (pseudo) random linear combination of the constraints, as often done in the PCP literature (since this would increase the length of the proof π and prevent us from obtaining sublinear complexity). This causes the length of the $\mathsf{universal\text{-}LVC}$ (i.e., the Reed-Muller encoding of the assignment to $\mathbb{F}$) to be roughly n^m.

We overcome this issue by arithmetizing over several (distinct) prime fields $\{\mathbb{F}_q\}_{q\in Q}$ such that: (1) for every $q \in Q$, the order of $\mathbb{F}_q$ is larger (by a constant multiplicative factor) than the total degree of the proof polynomial, which is $tm^2 \cdot |H|^2 = O(n^{2/m})$,[4] and (2) it holds that $\prod_{q\in Q} q > n$ (and so we shall set $|Q| \approx m$). We then invoke, in parallel, the foregoing MAP for each $\mathbb{F}_q$. This gives us the number of satisfied clauses modulo q, and since $\prod_{q\in Q} q > n$, we can use the Chinese remainder theorem to extract the number of satisfied clauses. Note that each $\mathbb{F}_q$ is of size $O(n^{2/m})$, and so the length of a universal-LVC that consists of the Reed-Muller encodings of the assignment to each field in $\{\mathbb{F}_q\}_{q\in Q}$ is $\tilde{O}(n^2)$.

Finally, recall that we wish the verifier to have access to the low-degree extension of an assignment over several finite fields, and so the verifier needs to be able to verify that its input actually consists of several polynomials that are consistent with the low-degree extension of a single assignment. Towards this end we bundle the foregoing polynomials using the PCP-based consistency mechanism discussed in Section 7.4.1 (which also allows us to ascertain that the assignment is binary).

7.2 Preliminaries

We begin with standard notations:

- We denote the *absolute distance*, over alphabet Σ, between two strings $x \in \Sigma^n$ and $y \in \Sigma^n$ by $\Delta(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|$ and their *relative distance* by $\delta(x,y) \stackrel{\text{def}}{=} \frac{\Delta(x,y)}{n}$. If $\delta(x,y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we denote the *absolute distance* of x from a non-empty set $S \subseteq \Sigma^n$ by $\Delta(x,S) \stackrel{\text{def}}{=} \min_{y\in S} \Delta(x,y)$ and the *relative distance* of x from S by $\delta(x,S) \stackrel{\text{def}}{=} \min_{y\in S} \delta(x,y)$. If $\delta(x,S) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S. We denote the projection of $x \in \Sigma^n$ on $I \subseteq [n]$ by $x|_I$.

- We denote by $A^x(y)$ the output of algorithm A given direct access to input y and oracle access to string x. Given two interactive machines A and B, we denote by $(A^x, B(y))(z)$ the output of A when interacting with B, where A (respectively, B) is given oracle access to x (respectively, direct access to y) and both parties have direct access to z. Throughout this chapter, probabilistic expressions that involve a randomized algorithm A are taken over the inner randomness of A (e.g., when we write $\Pr[A^x(y) = z]$, the probability is taken over the coin-tosses of A).

Integrality. Throughout this chapter, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the closest integer.

[4] This condition is required for the soundness of the MAP.

Uniformity. To facilitate notation, throughout this chapter we define all algorithms *non-uniformly*; that is, we fix an integer $n \in \mathbb{N}$ and restrict the algorithms to inputs of length n. Despite fixing n, we view it as a generic parameter and allow ourselves to write asymptotic expressions such as $O(n)$. We remark that while our results are proved in terms of non-uniform algorithms, they can be extended to the uniform setting in a straightforward manner.

Circuit Size. We define the size $s(k)$ of a Boolean circuit $C : \{0,1\}^k \to \{0,1\}$ as the number of *gates* C contains. We count the input vertices of C as gates, and so $s(k) \geq k$. We shall write $f \in \mathsf{SIZE}\big(s(k)\big)$ to state that a Boolean function $f : \{0,1\}^k \to \{0,1\}$ can be computed by a Boolean circuit of size $s(k)$.

7.2.1 Property Testing and Proofs of Proximity

In this section we review the definitions of testers, MAPs and IPPs. We begin with the definition of IPPs and obtain the definitions of testers and MAPs as special cases of IPPs.

Definition 7.1. *Let $n \in \mathbb{N}$. An* interactive proof of proximity *(*IPP*) for property $\Pi \subseteq \Sigma^n$ is an interactive protocol with two parties: a prover $\mathcal{P}$ that has* free *access to input $x \in \Sigma^n$, and a probabilistic verifier $\mathcal{V}$ that has oracle access to x. The parties exchange messages, and at the end of the communication the following two conditions are satisfied:*

1. Completeness: *For every proximity parameter $\varepsilon > 0$ and input $x \in \Pi$ it holds that*
$$\Pr\left[(\mathcal{V}^x, \mathcal{P}(x))(\varepsilon) = 1\right] \geq 2/3.$$
2. Soundness: *For every $\varepsilon > 0$, $x \in \Sigma^n$ that is ε-far from Π, and (cheating) prover $\mathcal{P}^*$ it holds that*
$$\Pr\left[(\mathcal{V}^x, \mathcal{P}^*)(\varepsilon) = 0\right] \geq 2/3.$$

If the completeness condition holds with probability 1, we say that the IPP *has a* one-sided error*, and otherwise we say that the* IPP *has a* two-sided error*.*

An IPP for property Π has query complexity (or locality) $q = q(n, \varepsilon)$ if for every $\varepsilon > 0$ and $x \in \Sigma^n$ the verifier $\mathcal{V}$ makes at most q queries to x, and communication complexity $c = c(n, \varepsilon)$ if for every $\varepsilon > 0$ and $x \in \Sigma^n$ the parties $\mathcal{V}$ and $\mathcal{P}$ exchange at most c bits. A round of communication consists of a single message sent from $\mathcal{V}$ to $\mathcal{P}$ followed by a single message sent from $\mathcal{P}$ to $\mathcal{V}$. An r-round IPP, where $r = (n, \varepsilon)$, is an IPP in which for every $\varepsilon > 0$ and $x \in \Sigma^n$ the number of rounds in the interaction between $\mathcal{V}$ and $\mathcal{P}$ on input x is at most r.

The definition of a tester can be derived from Definition 7.1 by allowing no communication (which effectively eliminates the prover). Similarly, the definition of an MAP can be derived by restricting the communication to a single message from $\mathcal{P}$ to $\mathcal{V}$ (see [GR13b] for further details on MAPs). We shall sometimes refer to a tester with respect to proximity parameter ε as an ε-tester, and similarly, we refer to an IPP (or MAP) with respect to proximity parameter ε as an IPP_ε (or MAP_ε).

7.2.2 Locally Testable Codes

Let $k, \eta \in \mathbb{N}$. A code over alphabet Σ with distance d is a function $C : \Sigma^k \to \Sigma^\eta$ that maps messages to codewords such that the distance between any two codewords is at least $d = d(\eta)$. If $d = \Omega(\eta)$, we say that C has linear distance. If $\Sigma = \{0, 1\}$, we say that C is a binary code. If C is a linear map, we say that it is a linear code. The relative distance of C, denoted by $\delta(C)$, is d/η, and its rate is k/η. When it is clear from the context, we shall sometime abuse notation and refer to the code C as the set of all codewords $\{C(x)\}_{x\in\Sigma^k}$. Following the discussion in the introduction, we define locally testable codes and locally decodable codes as follows.

Definition 7.2 (Locally Testable Codes)**.** *A code $C : \Sigma^k \to \Sigma^\eta$ is a locally testable code (LTC) if there exists a probabilistic algorithm (tester) T that, given oracle access to $w \in \Sigma^\eta$ and direct access to proximity parameter ε, satisfies:*

1. *Completeness: For any codeword $w = C(x)$, it holds that $\Pr[T^{C(x)}(\varepsilon) = 1] \geq 2/3$.*
2. *Soundness: For any $w \in \{0, 1\}^\eta$ that is ε-far from C, it holds that $\Pr[T^w(\varepsilon) = 0] \geq 2/3$.*

The query complexity of a LTC is the number of queries made by its tester (as a function of ε and k). A LTC is said to have one-sided error if its tester satisfy perfect completeness (i.e., accepts valid codewords with probability 1).

Definition 7.3 (Locally Decodable Codes)**.** *A code $C : \Sigma^k \to \Sigma^\eta$ is a locally decodable code (LDC) if there exists a constant $\delta_{\mathsf{radius}} \in (0, \delta(C)/2)$ and a probabilistic algorithm (decoder) D that, given oracle access to $w \in \Sigma^\eta$ and direct access to index $i \in [k]$, satisfies the following condition: For any $i \in [k]$ and $w \in \Sigma^\eta$ that is δ_{radius}-close to a codeword $C(x)$ it holds that $\Pr[D^w(i) = x_i] \geq 2/3$. The query complexity of a LDC is the number of queries made by its decoder.*

7.2.3 PCP of Proximity

PCPs of proximity (PCPPs) [BSGH+06, DR06] are a variant of PCP proof systems, which can be thought of as the PCP analogue of *property testing*. Recall that a standard PCP is given explicit access to a statement and oracle access to a proof. The PCP verifier is required to probabilistically verify whether the (explicitly given) statement is correct, by making few queries to proof. In contrast, a PCPP is given oracle access to a statement and a proof, and is only allowed to make a small number of queries to both the statement and the proof. Since a PCPP verifier only sees a small part of the statement, it cannot be expected to verify the statement precisely. Instead, it is required to only accept correct statements and reject statements that are far from being correct (i.e., far in Hamming distance from any valid statement). More precisely, PCPs of proximity are defined as follows.

Definition 7.4. *Let V be a probabilistic algorithm (verifier) that is given* explicit *access to a proximity parameter $\varepsilon > 0$,* oracle *access to an* input *$x \in \{0,1\}^k$ and to a* proof *$\bar{p} \in \{0,1\}^n$. We say that V is a* PCPP *verifier for language L if it satisfies the following conditions:*

- *Completeness: If $x \in L$, there exists a proof $\bar{p}$ such that the verifier always accepts the pair $(x, \bar{p})$; i.e., $V^{x,\bar{p}}(\varepsilon) = 1$.*
- *Soundness: If x is ε-far from L, then for every $\bar{p}$ the verifier rejects the pair $(x, \bar{p})$ with high probability; that is, $\Pr[V^{x,\bar{p}}(\varepsilon) = 0] \geq 2/3$.*

The length *of the* PCPP *is n and the* query complexity *is the number of queries made by V to* both *x and $\bar{p}$.*

We shall use the following PCPP due to Ben-Sasson and Sudan [BS05] and Dinur [Din07b].

Theorem 7.1 (Short PCPPs for NP)**.** *For every $L \subseteq \{0,1\}^k$ that can be computed by a circuit of size $t(k)$, there exists a* PCPP *with query complexity $q = O(1/\varepsilon)$ and length $t(k) \cdot \mathsf{polylog}(t(k))$.*

7.3 The Definition of Universal Locally Verifiable Codes

Following the discussion in the introduction, we define the MA analogue of universal-LTCs, i.e., universal-LTCs with MAPs instead of testers. We refer to such codes as "universal locally *verifiable* codes".

Definition 7.5. *Let $k, M \in \mathbb{N}$, and $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i \in [M]}$ be a family of functions. A* universal locally verifiable code *(*universal-LVC*) for $\mathcal{F}$ with query complexity $q = q(k, \varepsilon)$ and proof complexity $p = p(k, \varepsilon)$ is a code $C : \{0,1\}^k \to \{0,1\}^\eta$ such that for every $i \in [M]$ and $\varepsilon > 0$, there exists an* MAP*, with respect to proximity parameter ε, for the subcode $\Pi_i \stackrel{\text{def}}{=} \{C(x) : f_i(x) = 1\}$ with query complexity q and proof complexity p. A* universal-LVC *is said to have* one-sided error *if all of its* MAP*s satisfy perfect completeness.*

Notation. We shall refer to a universal-LVC with respect to a specific proximity parameter $\varepsilon > 0$ as a universal-LVC$_\varepsilon$.

Organization. In the first subsection (Section 7.4) we show an efficient universal-LVC for constraint satisfaction problems (CSPs). As discussed in the introduction, this universal-LVC can be viewed as a concise representation (or encoding) of assignments that allows for efficient MAPs for *every* CSP instance. We remark that the bundle consistency test (see Section 7.4.1) is used in the foregoing construction. Next, in Section 7.5 we show a lower bound on the complexity of universal-LVCs for conjugations (and in particular for CSPs). Finally, in Section 7.6 we show that using *interactive* verification procedures we can, in a

sense, emulate the universal-LVC in Section 7.4 and obtain an *interactive proof of proximity* (IPP) for any CSP. Note that this result refers to the standard model of IPPs, where the verifier is given access to a plain assignment (rather than to its encoding).

7.4 A Universal Locally Verifiable Code for CSP

Throughout this section, let $k, n, t \in \mathbb{N}$ such that $t \leq k$ (the reader is encouraged to think of t as being relatively small with respect to k). A constraint of arity t on k variables is a predicate $\mathsf{c} : \{0,1\}^k \to \{0,1\}$ that only depends on t coordinates (i.e., a t-junta). We denote the set of all such constraints by $\mathsf{Constraint}_{t,k}$.

Definition 7.6. *A function $\varphi : \{0,1\}^k \to \{0,1\}$ is an instance of a constraint satisfaction problem with n constraints of arity t, denoted $\varphi \in \mathsf{CSP}_{n,t,k}$ (or $\varphi \in \mathsf{CSP}_n$, if t and k are clear from the context), if $\varphi(x) = \bigwedge_{i=1}^{n} \mathsf{c}_i(x_1, \ldots, x_k) = 1$, where $\mathsf{c}_1, \ldots, \mathsf{c}_n \in \mathsf{Constraint}_{t,k}$.*

For example, in our formulation, a k-variate, n-clause 3SAT instance $\varphi : \{0,1\}^k \to \{0,1\}$ can be expressed as a $\mathsf{CSP}_{n,3,k}$ by writing $\varphi(x) = \bigwedge_{i=1}^{n} \mathsf{c}_i(x_1, \ldots, x_k)$, where each c_i is a disjunction of 3 literals from $\{ x_1, \ldots, x_k \} \cup \{ 1 - x_1, \ldots, 1 - x_k \}$. We stress that in Definition 7.6 we allow the constraints to be *arbitrary and different* predicates of the same arity.

The following theorem shows an efficient universal-LVC for constraint satisfaction problems. For simplicity, we assume without loss of generality that $n \geq k$ (otherwise, we add $k - n$ empty clauses).

Theorem 7.2. *Let $n, k, t, m \in \mathbb{N}$ such that $t < k \leq n$ and $\varepsilon > 1/\mathsf{polylog}(n)$.*[5] *There exists a (one-sided error) $\mathsf{universal\text{-}LVC}_\varepsilon$ $C : \{0,1\}^k \to \{0,1\}^{\tilde{O}(m^{2m+1} t^m \cdot n^2)}$ for $\mathsf{CSP}_{n,t,k}$ with linear distance such that for every $\ell \in [m/2]$, the universal-LVC has proof complexity $\tilde{O}(m^{2\ell} \cdot n^{2\ell/m} \cdot t^\ell)$ and query complexity $\tilde{O}(m \cdot t n^{1-\ell/m}/\varepsilon)$.*

Note that for constant t, m, and ε we obtain code length $\tilde{O}(n^2)$,[6] proof length $\tilde{O}(n^{2\ell/m})$, and query complexity $\tilde{O}(n^{1-\ell/m})$. In particular, for $\ell = m/3$ (e.g., for $m = 3$), Theorem 7.2 yields a (nearly) quadratic length universal-LVC with both proof and query complexity $\tilde{O}(n^{2/3})$. We remark that the proof complexity of our MAP has a factor of $m^{2\ell} \cdot t^\ell$ (and ℓ may be as large as $m/2$), and so we shall want to choose $m = O(1)$ and work with individual degree $d = n^{1/m}$ polynomials, rather than the usual setting of $m = \log(n)/\log\log(n)$ and $d = \log(n)$.

7.4.1 Preliminaries: Consistency-Testable Bundles

We shall need the following bundling mechanism from [GG16a], which in turn builds on techniques of Ben-Sasson at el. [BSGH+06] to show a way to bundle together (possibly

[5] We believe that the limitation on the proximity parameter can be eliminated, by adapting the techniques in [GGK15] to our setting. We leave the verification of this idea as an open problem.

[6] We remark that the *quadratic* length of our universal-LVC is inherent in our techniques, and it is an open question whether it is possible to obtain sub-quadratic length.

partial) encodings of the same message such that it possible to locally test that all these encodings are indeed consistent. That is, we are given some encodings $E_1, \ldots, E_s : \{0,1\}^k \to \{0,1\}^n$, and we wish to encode a *single* message $x \in \{0,1\}^k$ by all of these encodings (i.e., to bundle $E_1(x), \ldots, E_s(x)$) such that we can test that all of the encodings are valid and consistent with the same message x. In this chapter, the E_i's will correspond to the encodings of x by different error-correcting codes (i.e., Reed-Muller codes over different finite fields).

The main idea is to construct a bundle that consists of three parts: (1) the (explicit) message x, (2) the encodings $E_1(x), \ldots, E_s(x)$, and (3) PCPPs that assert the consistency of the first part (the message) with each purported encoding $E_i(x)$ in the second part. However, such PCPPs can only ascertain that each purported pair of message and encoding, denoted (y, z_i), is *close* to a valid pair $(x, E_i(x))$. Thus, in this way we can only verify that the bundle consists of encodings of pairwise-close messages, rather than being close to encodings of a single message (e.g., the PCPPs may not reject a bundle $(x, E_1(y_1), \ldots, E_s(y_s))$ wherein each y_i is close to x).

To avoid this problem, we also encode the message via an error-correcting code ECC, so the bundle is of the form $\big(\mathsf{ECC}(x), (E_1(x), \ldots, E_s(x)), (\mathsf{PCPP}_1(x), \ldots, \mathsf{PCPP}_s(x))\big)$. Now, each PCPP ascertains that a purported pair (y, z_i) is close to $(\mathsf{ECC}(x), E_i(x))$. Due to the distance of ECC, this allows to verify that the bundle consists of s (close to valid) encodings of the *same* message. Lastly, we repeat $\mathsf{ECC}(x)$ such that it constitutes most of the bundle's length, and so if an alleged bundle is far from valid, its copies of $\mathsf{ECC}(x)$ must be corrupted, and so the bundle itself constitutes an error-correcting code that is locally testable (by verifying at random one of the PCPPs in the bundle).

More precisely, consider the following way of bundling several encodings of the same message.

Construction 7.3 (Consistency-Testable Bundles)**.** *Let $E_1, \ldots, E_s : \{0,1\}^k \to \{0,1\}^n$ be encodings such that for every $i \in [s]$, the problem of (exactly) deciding whether $(x, y) \in \{0,1\}^{k+n}$ satisfies $y = E_i(x)$ can be computed by a size $t(k)$ circuit. The consistency-testable bundle of $\{E_i(x)\}_{i\in[s]}$ is the code $B(x) : \{0,1\}^k \to \{0,1\}^\ell$ that consists of the following ingredients.*

1. *An (arbitrary) code $\mathsf{ECC} : \{0,1\}^k \to \{0,1\}^{n'}$ with linear distance, which can be computed by a size $\tilde{O}(n')$ circuit, where $n' = \tilde{O}(k)$.*
2. *Encodings $E_1, \ldots, E_s$ (given by the application) that we wish to bundle.*
3. PCP *of proximity oracles $\bar{p}_1, \ldots, \bar{p}_s$ for the language*

$$L_i = \{(a,b) \ : \ \exists x \in \{0,1\}^k \text{ such that } a = \mathsf{ECC}(x)^{r_a} \text{ and } b = E_i(x)^{r_b}\}.$$

where and r_a, r_b are set such that $|a| \approx |b| = O(t(k))$.

Let $\varepsilon \geq 1/\mathsf{polylog}(s \cdot t(k))$. Consider the bundle

$$B(x) = \Big(\mathsf{ECC}(x)^r, \big(E_1(x), \ldots, E_s(x)\big), \big(\bar{p}_1(x), \ldots, \bar{p}_s(x)\big)\Big),$$

where the length of each PCPP *oracle* $\bar{p}_i(x)$ *is* $\tilde{O}(t(k))$,[7] *and where* r *is the minimal integer such that the first part of the bundle constitutes* $(1-\varepsilon/2)$ *fraction of the bundle's length (i.e.,* $|\mathsf{ECC}(x)|^r \geq (1-\varepsilon/2)\cdot\ell$*).*

Note that the length of B is $\ell = \tilde{O}(s \cdot t(k))$ and that B has linear distance, because $|\mathsf{ECC}(x)|^r$ dominates B's length.

In [GG16a], it is shown that there exists a local test that can ascertains the validity of the bundle as well as asserts the consistency of any encoding E_i in the bundle with the *anchor* of the bundle. Note that since the bundle's anchor dominates its length, it is possible that the bundle is very close to valid, and yet all of the E_i's are heavily corrupted. Thus, we also need to provide a test for the validity of each E_i and its consistency with the anchor.

Proposition 7.7. *For every bundle* $B(x)$*, as in Construction 7.3, there exists a consistency test* T *that for every* $\varepsilon \geq 1/\mathsf{polylog}(\ell)$ *makes* $O(1/\varepsilon)$ *queries to a string* $w \in \{0,1\}^\ell$ *and satisfies the following conditions.*

1. *If* $w = B(x)$*, then for every* $i \in \{0\} \cup [s]$ *it holds that* $\Pr[T^w(i) = 1] = 1$.
2. *If* w *is* ε*-far from* B*, then* $\Pr[T^w(0) = 0] \geq 2/3$.
3. *For every* $i \in [s]$*, if there exists* $x \in \{0,1\}^k$ *such that* w *is* ε*-close to* $B(x)$ *and* $\widetilde{E_i}(x)$ *is* ε*-far from* $E_i(x)$*, then* $\Pr[T^w(i) = 0] \geq 2/3$.

Note that $T^w(0)$ is a codeword test for B, whereas for every $i \in [s]$, the test $T^w(i)$ asserts that $\widetilde{E_i}$ is close to an encoding of the anchor. To verify that w is a bundle wherein all encodings refer to the same message (the anchor), we have to invoke $T^w(i)$ for all $i \in \{0\} \cup [s]$, but typically we will be interested only in the consistency of one encoding with the anchor, where this encoding is determined by the application. For completeness, we include the proof of Proposition 7.7 in Section 7.7.1.1.

7.4.2 Proof of Theorem 7.2

Following the overview presented in Section 7.1.4, we construct a universal-LVC that maps each assignment $x \in \{0,1\}^k$ to its low-degree extensions over m distinct finite fields, each of cardinality roughly $n^{1/m}$, bundled (via Construction 7.3) in a way that allows for locally verifying that all codewords encode the same assignment. More precisely, fix $d = n^{1/m} - 1$, and let Q be the set of the first $m/2$ primes greater than $10(m^2d^2t + d) = O(m^2t \cdot n^{2/m})$; note that each $q \in Q$ satisfies $q = O(m^2t \cdot n^{2/m})$ and that $\prod_{q\in Q} q > n$. For every $q \in Q$, denote by $\mathbb{F}_q$ the finite field with q elements.

[7] Note that $L_i \in \mathsf{SIZE}(m)$ by the hypothesis regarding ECC and E_i. Thus, by Theorem 7.1, such a PCPP exists.

The universal-LVC. Let $H = [d]$, and note that $H \subset \mathbb{F}_q$ for every $q \in Q$. We fix a bijection $H^m \leftrightarrow [n]$ and use these domains interchangeably. We denote by $X : H^m \to \{0,1\}$ the embedding of an assignment $x \in \{0,1\}^k$ in H^m, given by

$$X(z) = \begin{cases} x_z & \text{if } z \in [k] \\ 0 & \text{otherwise} \end{cases}.$$

For every $q \in Q$, let $\widehat{X}'_q : \mathbb{F}_q^m \to \mathbb{F}_q$ be the unique individual degree d extension of X to $\mathbb{F}_q$.

To reduce the alphabet to binary, let $C_0 : \mathbb{F}_q \to \{0,1\}^{100 \log |\mathbb{F}_q|}$ be a good linear code, and consider the concatenation of $\widehat{X}'_q$ with C_0 as the inner code, which we denote by $\widehat{X}_q : \mathbb{F}_q^m \to \{0,1\}^{100 \log |\mathbb{F}_q|}$. For convenience, we shall treat $\widehat{X}_q$ as if it maps to $\mathbb{F}_q$, and so whenever we query $\widehat{X}_q$ at a point $z \in \mathbb{F}_q^m$, we actually query the $100 \log |\mathbb{F}_q|$ bits of the codeword $C_0(\widehat{X}_q(z))$ and decode (the $\mathbb{F}_q$ element) $\widehat{X}_q(z)$.

Next, we bundle the Reed-Muller encodings $\{\langle \widehat{X}_q \rangle\}_{q \in Q}$ (where $\langle \widehat{X}_q \rangle$ denotes the evaluation of the function $\widehat{X}_q$ over its entire domain) according to Construction 7.3, so that we can locally test that all of these encodings are consistent with the same message (assignment). Recall that in Construction 7.3 we bundle encodings $E_i, \ldots, E_s$ with an (arbitrary) error-correcting code ECC (which can be computed by a circuit of quasilinear size and has linear distance) and with a PCPP for every E_i, which ascertains that a pair (a, b) satisfies $a = \mathsf{ECC}(y)$ and $b = E_i(y)$ for some y. Here, the encodings will correspond to the Reed-Muller encodings $\{\langle \widehat{X}_q \rangle\}_{q \in Q}$ of the assignment X. Note that (exact) verification of m-dimensional Reed-Muller codes over $\mathbb{F}_q$ can be done using circuits of size $m \cdot |\mathbb{F}_q|^m \cdot \mathsf{polylog}|\mathbb{F}_q| = \tilde{O}(m^{2m+1} t^m \cdot n^2)$, since $|\mathbb{F}_q| = O(m^2 t \cdot n^{2/m})$.[8] Hence, by Theorem 7.1, for every $q \in Q$ there exist a PCPP oracle $\bar{p}_q$, as required in Construction 7.3, of length $n' = \tilde{O}(m^{2m+1} t^m \cdot n^2)$. We obtain the code $C : \{0,1\}^k \to \{0,1\}^{m \cdot n'}$ given by

$$C(x) = \left(\mathsf{ECC}(x)^r, \left(\langle \widehat{X}_q \rangle \right)_{q \in Q}, \left(\bar{p}_q(x) \right)_{q \in Q} \right). \tag{7.5}$$

We show that C is a $\mathsf{universal\text{-}LVC}$ for CSP_n. This calls for describing a short (MAP) proof for each $\varphi \in \mathsf{CSP}_n$ and describing how it is verified.

Let $\varphi \in \mathsf{CSP}_n$, and write $\varphi(x) = \bigwedge_{i=1}^n \mathsf{c}'_i(x_1, \ldots, x_k) = 1$, where $\mathsf{c}'_1, \ldots, \mathsf{c}'_n \in \mathsf{Constraint}_{t,k,\{0,1\}}$. Recall that each c'_i is a t-junta, denote its influencing variables by I_i, and note that there exists $\mathsf{c}_i : \{0,1\}^t \to \{0,1\}$ such that $\mathsf{c}'_i(x) = \mathsf{c}_i(x|_{I_i})$. We stress that unlike the overview in Section 7.1.4, each constraint c_i may be a *different* predicate; this will make our arithmetization slightly more involved. Note that each c_i takes *binary* inputs, and so, for every $q \in Q$, we denote by $\widehat{\mathsf{c}}_{i,q} : \mathbb{F}_q^t \to \mathbb{F}_q$ the degree t *multilinear* extension of c_i to $\mathbb{F}_q$. We show an MAP for the subcode $\Pi_\varphi \stackrel{\text{def}}{=} \{C(x) : \varphi(x) = 1\}$. We shall first describe the MAP proof and then describe how it is verified.

[8] This can be done by checking that each one of the $m \cdot |\mathbb{F}_q|^{m-1}$ axis-parallel lines is a degree d univariate polynomial, and each such check can be done by a circuit of size $|\mathbb{F}_q| \cdot \mathsf{polylog}|\mathbb{F}_q|$.

The MAP proof (for $C(x)$ being in Π_φ). For every $q \in Q$, consider the following functions.

- Constraint Indicator: For every $i \in [n]$, let $\chi_i : H^m \to \{0,1\}$ be the indicator of the i'th constraint, i.e., for every $z \in H^m = [n]$ it holds that $\chi_i(z) = 1$ if and only if $z = i$. Denote by $\widehat{\chi}_{i,q} : \mathbb{F}_q^m \to \mathbb{F}_q$ the unique, individual degree d, extension of χ_i to $\mathbb{F}_q$. (This component is necessary now since each constraint may be a different predicate.)

- Variable Indicator: For every $j \in [t]$, let $\varphi_j : H^m \to H^m$ be the function that maps a *constraint* index $z \in H^m$ to the j'th *variable* index that appears in the z'th constraint (e.g., if $\mathsf{c}_z = (x_5 \vee x_7 \vee x_{11})$, then $\varphi_1(z) = 5$, $\varphi_2(z) = 7$, and $\varphi_3(z) = 11$). Denote by $\widehat{\varphi}_{j,q} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ the unique, individual degree d, extension of φ_j to $\mathbb{F}_q$. (The variable indicator is the same as in the overview.)

- Constraint-Satisfication Indicator: Let $\psi_q : \mathbb{F}_q^m \to \mathbb{F}_q$ be the total degree $m^2d^2t + md$ polynomial given by

$$\psi_q(z_1,\ldots,z_m) = \sum_{i=1}^{n} \widehat{\chi}_{i,q}(z_1,\ldots,z_m)\cdot\widehat{\mathsf{c}}_{i,q}\big(\widehat{X}_q\circ\widehat{\varphi}_{1,q}(z_1,\ldots,z_m),\ldots,\widehat{X}_q\circ\widehat{\varphi}_{t,q}(z_1,\ldots,z_m)\big), \tag{7.6}$$

  where the summation is over $\mathbb{F}_q$. Note that for every $z \in H^m$, the value of $\psi_q(z)$ indicates whether the z'th constraint of φ is satisfied by the assignment encoded in $\widehat{X}_q$. Note that the factor of $(md)^2$ in the degree of ψ_q is due to the composition of $\widehat{X}_q$ with $\widehat{\varphi}_{j,q}$.

The prescribed MAP proof for $C(x)$ being in Π_φ) is $\pi_\varphi = \{\pi_{\varphi,q}\}_{q\in Q}$, where $\pi_{\varphi,q} : \mathbb{F}_q^\ell \to \mathbb{F}_q$ is given by

$$\pi_{\varphi,q}(z_1,\ldots,z_\ell) = \sum_{z_{\ell+1},\ldots,z_m\in H} \psi_q(z_1,\ldots,z_\ell,z_{\ell+1},\ldots,z_m), \tag{7.7}$$

where the summation is over $\mathbb{F}_q$. Note that the length of the MAP proof is bounded by $\sum_{q\in Q}(m^2d^2t+md)^\ell\cdot 100\log|\mathbb{F}_q| = \tilde{O}(m^{2\ell}\cdot n^{2\ell/m}\cdot t^\ell)$, and observe that $\sum_{z_1,\ldots,z_\ell\in H}\pi_{\varphi,q}(z_1,\ldots,z_\ell)$ counts the number of φ's constraints that are satisfied by the assignment encoded in $\widehat{X}_q$ *modulo* q (due to the field's characteristic).

The MAP verifier (for φ). Hereafter, *we shall use $\widetilde{z}$ to denote a string that is allegedly equal to z.* Consider the MAP_ε verifier V_φ for the subcode $\{C(x) : \varphi(x) = 1\}$, which has free access to a purported proof $\widetilde{\pi}_\varphi = \{\widetilde{\pi}_{\varphi,q}\}_{q\in Q}$, which is supposed to equal $\pi_\varphi = \{\pi_{\varphi,q}\}_{q\in Q}$ (as defined above), and oracle access to a purported bundle $w \in \{0,1\}^{m\cdot n'}$ that is supposed to equal Eq. (7.5); that is, w allegedly consists of three parts: (1) the purported anchor $\widetilde{\mathsf{ECC}}(x)$, (2) the purported Reed-Muller encodings $(\langle\widetilde{X}_q\rangle)_{q\in Q}$, and (3) the purported PCPs of proximity $(\widetilde{p}_q(x))_{q\in Q}$. Let T be the bundle consistency test in Proposition 7.7. Recall that T is given a proximity parameter ε, an encoding-index parameter $q \in Q$, and oracle access to a purported bundle w. The test T accepts, with

high probability, if and only if w is ε-close to $C(x)$, and $\langle \widetilde{X}_q \rangle$ is ε-close to $\langle \widehat{X}_q \rangle$ (i.e., the low-degree extension of a *binary* assignment x).

The verifier V_φ performs the following checks for every $q \in Q$, in parallel, and accepts if none of the checks failed.

1. The MAP proof $\widetilde{\pi}_\varphi$ is consistent with a satisfying assignment: Check that
$$\sum_{z_1,\ldots,z_\ell \in H} \widetilde{\pi}_{\varphi,q}(z_1,\ldots,z_\ell) \equiv n \pmod{q}.$$

2. The universal-LTC itself is a bundle of Reed-Muller encodings of a binary assignment: Invoke the bundle consistency test T with respect to proximity parameter ε, encoding-index parameter q, and purported bundle w. (Hence, we may assume that $\langle \widetilde{X}_q \rangle$ is ε-close to $\langle \widehat{X}_q \rangle$, which is consistent with x; that is, all $\langle \widehat{X}_q \rangle$'s are pairwise consistent with the same *binary* assignment x.)

3. The MAP proof $\widetilde{\pi}_{\varphi,q}$ is consistent with the universal-LTC w: Compare the evaluation of $\widetilde{\pi}_{\varphi,q}$ and $\pi_{\varphi,q}$ at a random point. That is, recall that the verifier V_φ has the formula φ hard-coded, and so it can evaluate $\pi_{\varphi,q}$ (without help from the prover) by self-correcting $\widetilde{X}_q$, as follows. Select uniformly at random $r_1,\ldots,r_\ell \in_R \mathbb{F}_q$, and for every $z_{\ell+1},\ldots,z_m \in H$ and $j \in [t]$, decode $\widetilde{X}_q \circ \widehat{\varphi}_{j,q}(r_1,\ldots,r_\ell,z_{\ell+1},\ldots,z_m)$ using the Reed-Muller self-corrector, repeated $O((m-\ell)\cdot t \cdot \log(|H|))$ times so that the error probability in the self-correction is $1/(10 \cdot t \cdot |H|^{m-\ell})$ for each point. Denoting the value read by $v_{j,q}(r_1,\ldots,r_\ell,z_{\ell+1},\ldots,z_m)$, check that
$$\widetilde{\pi}_{\varphi,q}(r_1,\ldots,r_\ell) = \sum_{z_{\ell+1},\ldots,z_m \in H} \sum_{i=1}^{n} \widehat{\chi}_{i,q}(r_1,\ldots,r_\ell,z_{\ell+1},\ldots,z_m) \cdot \widehat{\mathsf{c}}_{i,q}\big(v_{1,q}(r_1,\ldots,r_\ell,z_{\ell+1},\ldots,z_m),\ldots,v_{t,q}(r_1,\ldots,r_\ell,z_{\ell+1},\ldots,z_m)\big). \tag{7.8}$$
(Note that, assuming Test 2 passes (with high probability) and all invocations of the self-corrector were successful,[9] the right-hand side of Eq. (7.8) equals $\pi_{\varphi,q}(r_1,\ldots,r_\ell)$.)

Recall that for each $q \in Q$, the purported proof $\widetilde{\pi}_{\varphi,q}$ is a low-degree polynomial (like $\pi_{\varphi,q}$). Hence, if $\widetilde{\pi}_{\varphi,q}$ and $\pi_{\varphi,q}$ agree (with high probability) on a random point, as checked in Test 3, then $\widetilde{\pi}_{\varphi,q} = \pi_{\varphi,q}$. Note that $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell)$ counts the number of constraints of φ that the binary assignment x satisfies modulo q (where Test 2 asserts that all

[9] Note that $\pi_{\varphi,q}$ is well defined if the purported bundle w is close to a codeword $C(x)$, which Test 2 asserts. In this case,
$$\pi_{\varphi,q}(z_1,\ldots,z_\ell) = \sum_{z_{\ell+1},\ldots,z_m \in H} \sum_{i=1}^{n} \widehat{\chi}_{i,q}(z_1,\ldots,z_m) \cdot \widehat{\mathsf{c}}_{i,q}\big(\widehat{X}_q \circ \widehat{\varphi}_{1,q}(z_1,\ldots,z_m),\ldots,\widehat{X}_q \circ \widehat{\varphi}_{t,q}(z_1,\ldots,z_m)\big),$$
where $\widehat{X}_q$ is the low-degree extension of x to $\mathbb{F}_q$. Hence, the verifier V_φ, which has the formula φ hard-coded, can evaluate $\pi_{\varphi,q}$ by self-correcting $\widetilde{X}_q$.

$\pi_{\varphi,q}$'s refer to the same assignment x). By Test 1, it follows that $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell)$ is congruent to n modulo q. Since this holds for all $q \in Q$, then by the Chinese remainder theorem, $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell) \equiv n \pmod{\prod_{q\in Q} q}$, and since $\prod_{q\in Q} q \geq n$, the assignment x satisfies the formula φ.

Note that for each of the $O(m)$ primes in Q, the verifier V_φ makes $O(1/\varepsilon)$ queries during the bundle consistency test and then queries $t \cdot |H|^{m-\ell} = t \cdot n^{1-(\ell/m)}$ points in $\widehat{X}_q$ via (amplified) self-correction of $\widetilde{X}_q$. Thus, the total query complexity is

$$\sum_{q\in Q} \left(O\left(\frac{1}{\varepsilon}\right) + tn^{1-\frac{\ell}{m}} \cdot O(m \log(|H|)) \right) \cdot \log(|\mathbb{F}_q|) = \tilde{O}\left(mt \cdot n^{1-\frac{\ell}{m}} \cdot \frac{1}{\varepsilon} \right).$$

Perfect completeness follows from the one-sided error of the bundle test and the self-correction procedure. The following claim establishes the soundness of V_φ.

Claim 7.7.1. *If w is ε-far from the subcode $\{C(x) : \varphi(x) = 1\}$, then for every alleged* MAP *proof $\widetilde{\pi}_\varphi$, it holds that $Pr[V_\varphi^w(\widetilde{\pi}_\varphi) = 0] \geq 2/3$.*

The proof of Claim 7.7.1 is a straightforward analysis of the construction, and so we defer its proof to Section 7.7.1.2. This concludes the proof of Theorem 7.2.

7.5 Lower Bounds on Verifying Conjugation Properties

Denote by Conjugation the set of all conjugations (of *at most* k variables); that is, Conjugation $= \{ f_S(x_1,\ldots,x_k) = \wedge_{i\in S} x_i \}_{S\subseteq[k]}$. The following theorem shows a lower bound on the universal-LVC complexity of Conjugation, which in particular, yields a lower bound on the universal-LVC complexity of CSP.

Theorem 7.4. *Suppose $C : \{0,1\}^k \to \{0,1\}^\eta$ is a code of constant relative distance $\delta(C)$, and fix $\varepsilon < \delta(C)$. If C is a* universal-LVC$_\varepsilon$ *for* Conjugation *with proof complexity p and query complexity q, then $p \cdot q = \Omega(k/\log \eta)$.*

Note that the foregoing lower bound trivializes for $\eta = 2^k$, and indeed there exists a universal-LTC for Conjugation of roughly such length (see [GG16a]). As an immediate consequence of Theorem 7.4, we obtain the following corollary.

Corollary 7.5. *Suppose $C : \{0,1\}^k \to \{0,1\}^\eta$ is a code of constant relative distance $\delta(C)$, and fix $\varepsilon < \delta(C)$. If C is a* universal-LVC$_\varepsilon$ *for* CSP$_{n,k}$ *with proof complexity p and query complexity q, then $p \cdot q = \Omega(k/\log \eta)$.*

We prove Theorem 7.4 by a reduction from MA communication complexity protocols, which we briefly recall next.

7.5.1 Preliminaries: MA Communication Complexity

In MA communication protocols we have a function $f : X \times Y \to \{0,1\}$, for some finite sets X and Y, and three computationally unbounded parties: Merlin, Alice, and Bob. The function f is known to all parties. Alice gets an input $x \in X$, and Bob gets an input $y \in Y$. Merlin sees both A, B, but Alice and Bob share a random string r that Merlin does not see. The protocol starts with a message $\pi = \pi(x, y)$ sent from Merlin to both Alice and Bob, which is supposed to be a proof that $f(x, y) = 1$. Then, the two players exchange messages to verify that indeed $f(x, y) = 1$.

Definition 7.8. *Let $f : X \times Y \to \{0,1\}$. An MA communication protocol for f, with proof complexity p and communication complexity c is a probabilistic protocol between two parties who share a random string r, and also receive a p-bit string $\pi = \pi(x, y)$, which is a functions of x and y, but independent of r. The parties communicate c bits and output $\langle A(x), B(y)\rangle(r, \pi)$ such that:*

1. *Completeness: for every Yes-input $(x, y) \in f^{-1}(1)$, there exists a proof $\pi \in \{0,1\}^p$ such that*
$$\Pr_r \left[\langle A(x), B(y)\rangle(r, \pi) = 1\right] \geq 2/3.$$
2. *Soundness: for every No-input $(x, y) \in f^{-1}(0)$ and for any alleged proof $\pi \in \{0,1\}^p$,*
$$\Pr_r \left[\langle A(x), B(y)\rangle(r, \pi) = 0\right] \geq 2/3.$$

We shall use the following (tight) lower bound on the MA communication complexity of the set-disjointness problem, in which Alice has input $S \subseteq [k]$, Bob has input $T \subseteq [k]$, and the parties need to decide whether their sets are disjoint; that is, compute the predicate

$$\mathsf{DISJ}_k(S, T) = \begin{cases} 1 & \text{if } |S \cap T| = 0 \\ 0 & \text{if } |S \cap T| \geq 1 \end{cases}.$$

It is well-known (see [KS92]) that the randomized communication complexity of the set-disjointness problem is linear in the length of the inputs. Moreover, Klauck [Kla03] showed the following (tight) lower bound on the MA communication complexity of set-disjointness.

Theorem 7.6 ([Kla03])**.** *Every MA communication complexity protocol for DISJ_k with proof complexity p and communication complexity c satisfies $p \cdot c = \Omega(k)$.*

7.5.2 Proof of Theorem 7.4

Consider the communication complexity problem, in which Alice has input $A \subseteq [k]$, Bob has input $B \subseteq [k]$, and the parties need to decide whether Alice's set is a subset of Bob's set; that is, compute the predicate $\mathsf{SUBSET}_k(A, B) = \begin{cases} 1 & \text{if } A \subseteq B \\ 0 & \text{otherwise} \end{cases}$.

Claim 7.8.1. *Every* MA *communication complexity protocol for* SUBSET_k *with proof complexity* p *and communication complexity* c *satisfies* $p \cdot c = \Omega(k)$.

Proof. We reduce from DISJ_k. Let $\mathsf{Prot}_{\mathsf{SUBSET}}$ be an MA protocol for SUBSET_k with proof complexity p and communication complexity c, and let $S, T \subseteq [k]$ be the inputs of Alice and Bob to the DISJ_k problem. The parties emulate $\mathsf{Prot}_{\mathsf{SUBSET}}$ on inputs $A \stackrel{\text{def}}{=} S$ and $B \stackrel{\text{def}}{=} [k] \setminus T$. Note that if $S \cap T = \emptyset$, then $A = S \subseteq [k] \setminus T = B$. Otherwise, there exists $i \in S \cap T$ such that $i \notin [k] \setminus T = B$, and $A \not\subseteq B$ follows. We stress that the reduction maps 1-instances to 1-instances, and so it preserves membership in the class MA. □

We prove the following claim by adapting the methodology in [BBM11], in which property testing lower bounds are obtained via reductions from communication complexity, to the setting of universal-LTCs.

Claim 7.8.2. *If the* universal-LVC C *has proof complexity* p *and query complexity* q*, then there exists an* MA *communication complexity protocol for* SUBSET_k *with proof complexity* p *and communication complexity* $q \cdot (1 + \log \eta)$.

Proof. Let $A, B \subseteq [k]$ be the inputs of Alice and Bob (respectively) to the SUBSET_k problem. Bob computes the codeword $C(B)$, where B is viewed as a k-bit string.[10] Then, Alice invokes the MAP verifier for the subcode $C_A \stackrel{\text{def}}{=} \{C(x) : \wedge_{i \in A} x_i = 1\}$, and answers each of its q queries by communicating with Bob as follows. On query $i \in [\eta]$, Alice sends i (communicating $\log \eta$ bits) to Bob, who responds with (a single bit) $C(B)_i$, which Alice provides as answer to the MAP verifier for C_A, denoted V_A. If $A \subseteq B$, then $\wedge_{i \in A} B_i = 1$, and so $C(B) \in C_A$; thus there exists a proof $\pi \in \{0,1\}^p$ such that $\Pr[V_A^{C(B)} = 1] \geq 2/3$. Otherwise (i.e., $A \not\subseteq B$), there exists $i \in A$ such that $i \notin B$, hence $\wedge_{i \in A} B_i = 0$, and so $C(B)$ is $\delta(C)$-far from C_A, and for every $\pi \in \{0,1\}^p$ it holds that $\Pr[V_A^{C(B)} = 0] \geq 2/3$. □

Combining Claim 7.8.1 and Claim 7.8.2 concludes the proof of the Theorem 7.4.

7.6 Constant-Round IPPs for CSP

Recall that an interactive proof of proximity (hereafter, IPP) is an interactive proof system in which the verifier only queries a sublinear number of input bits and soundness only means that, with high probability, the input is close to an accepting input (see Definition 7.1). In this section, we show that using $O(1)$ rounds of interaction, an IPP protocol wherein the verifier has oracle access to an assignment $x \in \{0,1\}^k$ can, in a sense, emulate the universal-LVC for CSP of Theorem 7.2; thus, we obtain an efficient IPP for satisfiability of fixed CSPs. We shall make an effort to keep the round complexity of such IPP to a minimum. We warn that Section 7.4 is a prerequisite for this section.

[10] Via the standard mapping in which the i'th bit of the string is 1 if $i \in B$ and 0 otherwise.

Let $k \in \mathbb{N}$. We consider $\mathsf{CSP}_n = \mathsf{CSP}_{n,t,k}$, where for simplicity of presentation, in this subsection we fix $n = k$ and $t = O(1)$ (generalizing to general values of n, k, t can handled similarly as in Section 7.4). Recall that each *round* of an IPP consists of two messages, one from the prover and one from the verifier (see Section 7.2.1). We prove the following.

Theorem 7.7. *For every $\varepsilon \geq 1/n^{6/7}$ and $\varphi \in \mathsf{CSP}_n$ there exists a 3-round (one-sided error) IPP for the property $\Pi_\varphi = \{ x \in \{0,1\}^k : \varphi(x) = 1 \}$ with communication and query complexity $O(n^{6/7+o(1)})$.*

We remark that by allowing additional $O(1)$ rounds of interaction, it is possible to obtain both query and communication complexity n^α for any constant $\alpha > 1/2$, see Section 7.6.3.

7.6.1 High-Level Overview

We start with a brief overview of the main ideas behind the proof of Theorem 7.7. Fixing any $\varphi \in \mathsf{CSP}_n$, let $C(x)$ be the universal-LVC encoding of an assignment $x \in \{0,1\}^k$, as used in Theorem 7.2. Recall that $C(x)$ consists of a bundle of Reed-Muller encodings of x over several prime fields $\{\mathbb{F}_q\}_{q \in Q}$,[11] and let V_φ be the MAP verifier for $\Pi_i = \{C(x) : \varphi(x) = 1\}$.

Let $C(x)$ be a *valid* codeword (where $\varphi(x) \in \{0,1\}$). Then, by Theorem 7.2: (1) if $\varphi(x) = 1$, then there exists a proof π such that $\Pr[V_\varphi^{C(x)}(\pi) = 1] = 1$, and (2) if $\varphi(x) = 0$, then for every alleged proof π it holds that $\Pr[V_\varphi^{C(x)}(\pi) = 1] < 1/3$. A closer inspection of the proof of Theorem 7.2 shows that, for every $q \in Q$, the verifier $V_\varphi(\pi)$ generates, as a function of the alleged proof π and its own randomness, a subset of indices $J_q \subseteq [|C(x)|]$ and a vector of values $\vec{v}_q \in \{0,1\}^{|J_q|}$ such that: (1) if $\varphi(x) = 1$, then for every $q \in Q$ there exists a proof π such that $\Pr_{(J_q,\vec{v}_q) \leftarrow V_\varphi(\pi)}[C(x)|_{J_q} = \vec{v}_q] = 1$, and (2) if $\varphi(x) = 0$, then for every alleged proof π there exists $q \in Q$ such that $\Pr_{(J_q,\vec{v}_q) \leftarrow V_\varphi(\pi)}[C(x)|_{J_q} = \vec{v}_q] < 1/3$.[12] Hence, we view V_φ as a *reduction* of verifying that x satisfies φ to verifying that $C(x)|_{J_q} = \vec{v}_q$ for every $q \in Q$. Hereafter, we fix $q \in Q$ and omit it from subscripts.

Recall, however, that in the setting of Theorem 7.7 the verifier does *not* have access to the encoding $C(x)$, but rather only oracle access to the plain assignment x itself. Aiming at sublinear query complexity, the verifier cannot read all of x. Instead the verifier sends the set of locations J to the prover and asks it to prove to it that $C(x)|_J = \vec{v}$. To this end, we use techniques from [RVW13] that allow us to verify claims regarding $C(x)$ by only making a small number of queries to x. This is performed in two steps, which we describe next.

[11] Actually, C consists of the foregoing Reed-Muller encodings, bundled with PCPPs that ascertain the consistency of the encodings (see Construction 7.3). However, in the context of Theorem 7.7, we shall not need these PCPPs, and we view C as consisting solely of the low-degree extensions.

[12] This is because (1) the verifier is *non-adaptive*, and (2) assuming $C(x)$ is valid, the verifier only needs to make queries to the Reed-Muller encodings (and do *not* need to query the PCPP oracles that are used for consistency testing).

The first step is to strengthen the soundness condition of V_φ such that, with high probability, if x is ε-far from $\Pi_i \stackrel{\text{def}}{=} \{z \in \{0,1\}^k \,:\, \varphi(z) = 1\}$, not only $C(x)|_J \neq \vec{v}$, but also for every x' that is ε-close to x (simultaneously) it holds that $C(x')|_J \neq \vec{v}$. That is, if x is ε-far from Π_i, then it is ε-far from $\{z \in \{0,1\}^k \,:\, C(z)|_J = \vec{v}\}$. The second step is to invoke an IPP (due to [RVW13]) for verifying membership in $\{z \in \{0,1\}^k \,:\, C(z)|_J = \vec{v}\}$, where C consists of Reed-Muller encodings. Details follow.

Denote the query complexity of the verifier V_φ by ℓ. We start by reducing the soundness error of V_φ, via S parallel repetitions (at the cost of increasing the the query complexity to $S \cdot \ell$). Note that the amplified verifier V'_φ generates a pair $(J, \vec{v})$ of $O(S \cdot \ell)$ locations and values, such that if $\varphi(x) = 0$, then $\Pr_{(J,\vec{v})}[C(x)|_J = \vec{v}] = \mathsf{exp}(-S)$. Observe that if x is ε-far from satisfying φ (and in particular $\varphi(x') = 0$), then the probability there exists x' that is ε-close to x such that $C(x')|_J = \vec{v}$ is at most $\binom{n}{\varepsilon n} \cdot \mathsf{exp}(-S)$.

Therefore, by setting $S = \Theta(\varepsilon \cdot n \log n)$ we obtain that with high probability no x' that is ε-close to x satisfies $C(x')|_J = \vec{v}$. Thus, if x is ε-far from $\{x \in \{0,1\}^k \,:\, \varphi(x) = 1\}$, then with high probability (over the pair $(J, \vec{v})$, chosen by V'_φ) the assignment x is ε-far from the affine subspace $A_{J,\vec{v}} \stackrel{\text{def}}{=} \{x \in \{0,1\}^k \,:\, C(x)|_J = \vec{v}\}$.

Therefore, the foregoing constitutes a 2-message "reduction": The prover sends the MAP proof (constructed as in Theorem 7.2) that x satisfies φ, and the verifier sends back a set of random locations J, asking the prover to provide a vector $\vec{v}$ and prove that it is equal to $C(x)|_J$. Hence, we performed a randomized reduction of verifying that x satisfies φ to verifying membership in the affine subspace $A_{J,\vec{v}}$. Fortunately, 3-message IPPs with sublinear communication and query complexity are known for testing membership in affine subspaces that are induced by Reed-Muller codes. Furthermore, these IPPs also have sublinear communication and query complexity for sub-constant values of ε. This is crucial since we perform $S = \Theta(\varepsilon \cdot n \log n)$ parallel repetitions of V_φ, which adds a factor of $\Theta(\varepsilon \cdot n \log n)$ to the communication complexity, and since we aim for sublinear communication complexity, the proximity parameter must be sub-constant. Finally, we compose the aforementioned reduction protocol with an IPP for membership in $A_{J,\vec{v}}$, and hence obtain an IPP for $\left\{ x \in \{0,1\}^k \,:\, \varphi(x) = 1 \right\}$.

To present the actual proof of Theorem 7.7, we shall need to define the following property of membership in the affine subspace that corresponds to the Reed-Muller code.

Definition 7.9 (PVAL)**.** *Let $\mathbb{F}$ be a finite field, $J \subseteq \mathbb{F}^m$, and $\vec{v} \in \mathbb{F}^{|J|}$. The property $\mathsf{PVAL}^{\mathbb{F},d,m}_{J,\vec{v}}$ (or just $\mathsf{PVAL}^{\mathbb{F}}_{J,\vec{v}}$, when d and m are clear from the context) consists of all strings $x \in \{0,1\}^{d^m}$ such that their (individual) degree d extension to $\mathbb{F}$, denoted $\widehat{X} : \mathbb{F}^m \to \mathbb{F}$, takes the values $\vec{v}$ on the coordinates J; that is,*

$$\mathsf{PVAL}^{\mathbb{F}}_{J,\vec{v}} = \{x \in \{0,1\}^{d^m} \,:\, \widehat{X}(J) = \vec{v}\}.$$

The following theorem, due to Rothblum et al. [RVW13], shows that PVAL has efficient IPPs.

Theorem 7.8 ([RVW13, Theorem 3.12])**.** *Let $d, m \in N$, and let $\mathbb{F}$ be a finite field. Fix parameters r and q such that $r \leq \min(d, |F|/10)$ and $q > \max\{(d^r)^{1+o(1)}, |\mathbb{F}|\}$.*

Then, for every $J \subseteq \mathbb{F}^m$, $\vec{v} \in \mathbb{F}^{|J|}$, and any $\varepsilon \geq 1/q^{1-o(1)}$ there exists a one-sided error, $(2r+1)$-message (where the first message is sent by the prover) IPP_ε *for* $\mathsf{PVAL}_{J,\vec{v}}^{\mathbb{F},d,m}$ *with communication complexity $(d^{m-r} + |J| \cdot d) \cdot q^{o(1)}$ and query complexity q.*

We remark that the product of the proof and query complexities in Theorem 7.8 can be made almost linear in some cases; specifically, for $r = \frac{\log q}{\log d}$ we obtain communication complexity $\frac{d^m}{q^{1-o(1)}} + |J| \cdot d \cdot q^{o(1)}$ and query complexity q. We shall, however, use $r = O(1)$.

7.6.2 Proof of Theorem 7.7

Let $\varphi \in \mathsf{CSP}_n$, and write $\varphi(x) = \bigwedge_{i=1}^{n} \mathsf{c}'_i(x_1, \ldots, x_k)$, where $\mathsf{c}'_1, \ldots, \mathsf{c}'_n \in \mathsf{Constraint}_{t,k,\{0,1\}}$. Recall that each c'_i is a t-junta, denote its influencing variables by I_i, and note that there exists $\mathsf{c}_i : \{0,1\}^t \to \{0,1\}$ such that $\mathsf{c}'_i(x) = \mathsf{c}_i(x|_{I_i})$.

We show an IPP for the property $\Pi_\varphi \stackrel{\text{def}}{=} \{x \in \{0,1\}^k \ : \ \varphi(x) = 1\}$. As discussed in the overview, we begin by using a similar construction to that of Theorem 7.2, to the end of performing a randomized reduction of verifying that the assignment x satisfies φ to verifying membership in the affine subspace induced by Reed-Muller encodings of x. More accurately, we shall use a "bare-bones" version of the foregoing universal-LTC, which only consists of Reed-Muller encodings of x over several prime fields (note that we omit both the alphabet reduction, and the PCP-based consistency testing mechanism), and whose MAP verifiers do not query the universal-LTC, but rather send to the prover the queries they wish to make. We stress that this construction do *not* include the anchor and PCPPs in Construction 7.3.

For the convenience of the reader, we briefly review the following definitions from Section 7.4, which are needed to describe the foregoing "bare-bones" version of the universal-LTC in Theorem 7.2.

Review of the arithmetization in Theorem 7.2. Let $m = O(1)$, to be determined later, and fix $d = n^{1/m} - 1$. Let Q be the set of the first $m/2$ primes that are greater than $2(m^2d^2t + md) = O(n^{2/m})$. Note that $\prod_{q \in Q} q > n$. Let $q \in Q$. Denote by $\mathbb{F}_q$ the finite field with q elements. Denote by $\widehat{\mathsf{c}}_{i,q} : \mathbb{F}_q^t \to \mathbb{F}_q$ the multilinear extension of c_i to $\mathbb{F}_q$. Let $H = [d]$ (note that $H \subset \mathbb{F}_q$); we fix a bijection $H^m \leftrightarrow [n]$ and use these domains interchangeably. For every $x \in \{0,1\}^k$ consider $X_q : H^m \to \{0,1\}$ given by $X_q(z) = x_z$. Let $\widehat{X}_q : \mathbb{F}_q^m \to \mathbb{F}_q$ be the unique individual degree d extension of X_q to $\mathbb{F}_q$.

For every $i \in [n]$, let $\chi_i : H^m \to \{0,1\}$ be the indicator of the i'th constraint, i.e., for every $z \in H^m = [n]$ it holds that $\chi_i(z) = 1$ if and only if $z = i$. Denote by $\widehat{\chi}_{i,q} : \mathbb{F}_q^m \to \mathbb{F}_q$ the unique, individual degree d, extension of χ_i to $\mathbb{F}_q$. For every $j \in [t]$, let $\varphi_j : H^m \to H^m$ be the function that maps a *constraint* index $z \in H^m$ to the j'th *variable* index that appears in the z'th constraint. Denote by $\widehat{\varphi}_{j,q} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ the unique, individual degree d, extension of φ_j to $\mathbb{F}_q$. For every $i \in [n]$ and $j \in [t]$, denote by $\widehat{\chi}_{i,q}$ and $\widehat{\varphi}_{j,q}$ the low-degree extension of χ_i and φ_j to $\mathbb{F}_q$. Finally, let $\psi_q(z_1, \ldots, z_m) : \mathbb{F}_q^m \to \mathbb{F}_q$,

given by

$$\psi_q(z_1,\ldots,z_m)=\sum_{i=1}^{n}\widehat{\chi}_{i,q}(z_1,\ldots,z_m)\cdot\widehat{\mathsf{c}}_{i,q}\big(\widehat{X}_q\circ\widehat{\varphi}_{1,q}(z_1,\ldots,z_m),\ldots,\widehat{X}_q\circ\widehat{\varphi}_{t,q}(z_1,\ldots,z_m)\big).$$

Having reviewed the foregoing definitions, we are ready to proceed with the proof of Theorem 7.7.

The 3-round IPP. Let $\varepsilon>0$, $\ell\in[m/2]$, and $S\in\mathbb{N}$, to be determined later. Consider the following 3-round IPP_ε for the property $\Pi_\varphi\stackrel{\text{def}}{=}\{x\in\{0,1\}^k\ :\ \varphi(x)=1\}$. The protocol starts by emulating a "bare-bones" version of the MAP verifier of Theorem 7.2, which differs in the following aspects: (1) the consistency test and alphabet reduction are omitted, (2) the soundness of the verifier is amplified via $S=O(\varepsilon n\log n)$ parallel repetitions, and (3) the verifier does *not* make queries to its input, but rather communicates to the prover the queries it wishes to make and asks the prover to assert the values of these queries. Details follow.

Hereafter, we shall denote by $\widetilde{f}$ a function, sent by the prover, which allegedly equals f. For every $q\in Q$, the prover sends a polynomial $\widetilde{\pi}_q:\mathbb{F}_q^\ell\to\mathbb{F}_q$, which allegedly equals $\pi_q(z_1,\ldots,z_\ell)\stackrel{\text{def}}{=}\sum_{z_{\ell+1},\ldots,z_m\in H}\psi_q(z_1,\ldots,z_\ell,z_{\ell+1},\ldots,z_m)$, where the summation is over $\mathbb{F}_q$. The verifier first checks that all π_q's are consistent with a satisfying assignment (i.e., checks that $\sum_{z_1,\ldots,z_\ell\in H}\widetilde{\pi}_q(z_1,\ldots,z_\ell)\equiv n\pmod q$, for all $q\in Q$). Then, the verifier wishes to evaluate each π_q on S randomly chosen points and compare it to the value of $\widetilde{\pi}_q$ on these points,[13] which amounts to evaluating the low-degree extensions $\{\widehat{X}_q\}_{q\in Q}$ of the assignment x at $S\cdot|H|^{m-\ell}$ points; denote these points by J_q.

Recall, however, that the verifier only has access to the plain assignment x, and not to its encodings $\{\widehat{X}_q\}_{q\in Q}$ (note that evaluating $\widehat{X}_q$ at any point, without assistance from the prover, may require reading the assignment x entirely). Instead the verifier asks the prover to assert the values of $\{\widehat{X}_q\}_{q\in Q}$ at the points it wishes to probe. To that end, the verifier selects uniformly at random $r_q^{(s)}\stackrel{\text{def}}{=}(r_1^{(s)},\ldots,r_\ell^{(s)})\in\mathbb{F}_q^\ell$, for every $s\in[S]$ and sends it to the prover, which in turns sends a vector $\vec{v}_q$ of the evaluations of $\widehat{X}_q$ at J_q, for every $q\in Q$. Finally the parties invoke the IPP in Theorem 7.8 with respect to $(J_q,\vec{v}_q)$, for every $q\in Q$, and accept if and only if all of the invocations accepted. More accurately, the IPP is described as follows. For every $q\in Q$, in parallel, perform the following steps:

1. The prover sends a (total) degree m^2d^2t+md polynomial $\widetilde{\pi}_q:\mathbb{F}_q^\ell\to\mathbb{F}_q$ (by specifying its coefficients), which allegedly equals:

$$\pi_q(z_1,\ldots,z_\ell)=\sum_{z_{\ell+1},\ldots,z_m\in H}\psi_q(z_1,\ldots,z_\ell,z_{\ell+1}\ldots,z_m).$$

[13] Note that by the proof of Theorem 7.2, evaluating each π_q on a *single* randomly chosen point yields constant soundness, and so, in the setting of Theorem 7.7, as discussed in the overview, we obtain soundness $\mathsf{exp}(-S)$ by evaluating each π_q on S randomly chosen points.

2. The verifier checks that $\sum_{z_1,\ldots,z_\ell \in H} \widetilde{\pi}_q(z_1,\ldots,z_\ell) \equiv n \pmod q$.

3. The verifier selects uniformly at random and sends $r_q^{(s)} \stackrel{\text{def}}{=} (r_1^{(s)},\ldots,r_\ell^{(s)}) \in \mathbb{F}_q^\ell$, for every $s \in [S]$.

4. The prover sends $\vec{v}_q \in \mathbb{F}_q^{S\cdot|H|^{m-\ell}\cdot t}$ such that allegedly $\vec{v}_q[s,\vec{z},i] = \widehat{X}_q \circ \widehat{\varphi}_{i,q}\big(r_1^{(s)},\ldots,r_\ell^{(s)},\vec{z}\big)$, for every $s \in [S]$, $\vec{z} \in H^{m-\ell}$, and $i \in [t]$.

5. The verifier checks that, for every $s \in [S]$,

$$\sum_{\vec{z}\in H^{m-\ell}} \sum_{i=1}^{n} \widehat{\chi}_{i,q}\big(r_1^{(s)},\ldots,r_\ell^{(s)},z\big) \cdot \widehat{\mathsf{c}}_{i,q}\big(\vec{v}_q[s,z,1],\ldots,\vec{v}_q[s,z,t]\big) \equiv n \pmod q.$$

6. Fix $J_q = \big(\widehat{\varphi}_{i,q}(r_1^{(s)},\ldots,r_\ell^{(s)},\vec{z})\big)_{s\in[S],\vec{z}\in H^{\ell-m},i\in[t]}$, and invoke the IPP for PVAL (Theorem 7.7) on input x (the assignment), field $\mathbb{F}_q$, location set J_q, and evaluation vector $\vec{v}_q$.

Note that in Step 1 the prover communicates $\sum_{q\in Q}(m^2d^2t+md)^\ell \cdot \log|\mathbb{F}_q|$ bits, in Step 3 the verifier sends $\sum_{q\in Q} S\cdot\ell\cdot\log|\mathbb{F}_q|$ bits, and in Step 4, the prover sends $\sum_{q\in Q} S\cdot|H|^{m-\ell}\cdot t\cdot\log|\mathbb{F}_q|$ bits. Hence, prior to the final step (i.e., Step 6), $\tilde{O}\big(n^{2\ell/m}+S\cdot n^{m-\ell/m}\big)$ bits are being communicated and *no* queries are being made to the assignment x by the verifier.

Finally, the parties invoke the 3-message (starting with the prover) PVAL IPP (in Step 6), whose communication complexity is

$$\left(d^{m-1}+\sum_{q\in Q}|J_q|\cdot d\right)\cdot q^{o(1)} = \left(n^{\frac{m-1}{m}}+S\cdot n^{\frac{m-\ell+1}{m}}\right)\cdot q^{o(1)}$$

and query complexity is q. (Note that only the PVAL protocol actually makes queries to the input x). Fixing $\varepsilon = 1/n^{6/7}$, $q = n^{6/7+o(1)}$, $S = O(\varepsilon n\log n)$, $m = 7$, and $\ell = 3$ yields the claimed complexity. Perfect completeness follows by construction. To show soundness, we shall first need the following claim

Claim 7.9.1. *If $x \notin \Pi_\varphi$, then there exists $q \in Q$ such that $\Pr_{(J_q,\vec{v}_q)}[\widehat{X}_q(J_q) = \vec{v}_q] < (1/10)^S$.*

The proof of Claim 7.9.1 is by a straightforward analysis of the construction, and thus we defer its proof to Section 7.7.1.3. Next, assume that x is ε-far from Π_φ, and observe that by Claim 7.9.1 there exists $q \in Q$ such that

$$\begin{aligned}
\Pr_{(J_q,\vec{v}_q)}[\forall x' \in N_\varepsilon(x)\ \widehat{X'}_q|(J_q) \neq \vec{v}_q] &\geq 1 - \binom{n}{\varepsilon n}\cdot \max_{x'\notin\Pi_\varphi}\left\{\Pr_{(J_q,\vec{v}_q)}[\widehat{X'}_q(J_q)=\vec{v}_q]\right\} \\
&\geq 1-\binom{n}{\varepsilon n}\cdot(1/10)^S && \text{(Claim 7.9.1)} \\
&\geq 9/10. && (S = O(\varepsilon n\log n))
\end{aligned}$$

(where $N_\varepsilon(x)$ consists of all strings that are ε-close to x). Thus, there exists $q \in Q$ such that with probability $9/10$ over the verifier's randomness, the assignment x is ε-far from $\mathsf{PVAL}^{\mathbb{F}_q}_{J_q,\vec{v}_q}$, and so, by Theorem 7.7, x is rejected with probability at least $9/10 \cdot 9/10$ in the last step of the IPP (the invocation of the PVAL protocol). This concludes the proof of Theorem 7.7.

7.6.3 Round Complexity versus Communication and Query Complexity Tradeoff

The proof of Theorem 7.7 naturally extends to IPPs with a higher round complexity, admitting $O(1)$-round IPPs with proof and query complexity n^α for any constant $\alpha > 1/2$. We sketch below how such extension is performed.

The idea is to replace the emulation of the "bare-bones" MAP verifier V_φ (Steps 1-3 of the IPP in Theorem 7.7) with a sumcheck protocol [LFKN92], in which the summation is striped down in iterations, coordinate-by-coordinate. That is, the protocols starts with m rounds (recall that we arithmetize over m-variate polynomials), where in the j'th round, for every $q \in Q$ and $s \in [S]$, the prover sends a degree $m^2d^2t + md$ *univariate* polynomial $\widetilde{\pi}^{(s)}_{j,q} : \mathbb{F}_q \to \mathbb{F}_q$ that allegedly equals:

$$\pi^{(s)}_{j,q}(z) = \sum_{z_{j+1},\ldots,z_m \in H} \psi_q(r^{(s)}_1, \ldots, r^{(s)}_{j-1}, z, z_{j+1} \ldots, z_m).$$

The verifier then checks the consistency of each $\widetilde{\pi}^{(s)}_{j,q}$ with $\widetilde{\pi}^{(s)}_{j-1,q}$; i.e., verifies that

$$\widetilde{\pi}^{(s)}_{j-1,q}\left(r^{(s)}_{j-1}\right) = \sum_{z \in H} \pi^{(s)}_{j,q}(z),$$

and the j'th round is concluded by letting the verifier select uniformly at random $r^{(s)}_j \in \mathbb{F}_q$ and send it to the prover.

Standard analysis of the sumcheck protocol shows that the larger m is (which in turn dictates the round complexity), the smaller the communication and query complexity of such protocols; in particular for $O(1)$-rounds, we can obtain both query and proof complexity n^β, where $\beta = \beta(m)$ is an arbitrarily small constant. The bottleneck in both query and proof complexity is, however, due to the final step of our IPP, which is an invocation of IPP in Theorem 7.8, wherein both query and proof complexity are inherently $\omega(\sqrt{n})$.

7.7 Appendices for Chapter 7

7.7.1 Deferred Details of Proofs

7.7.1.1 Proof of Proposition 7.7

For the analysis, when we consider an arbitrary string $w \in \{0,1\}^\ell$ (which we think of as an alleged bundle), we view $w \in \{0,1\}^{\ell_1+\ell_2+\ell_3}$ as a string composed of three parts (analogous to the three parts of Construction 7.3):

1. The anchor, $\widetilde{\mathsf{ECC}}(x) = (\widetilde{\mathsf{ECC}}(x)_1, \ldots, \widetilde{\mathsf{ECC}}(x)_r) \in \{0,1\}^{n' \cdot r}$, which consists of r alleged copies of $\mathsf{ECC}(x)$;
2. The bundled encodings $(\widetilde{E_1}(x), \ldots, \widetilde{E_s}(x)) \in \{0,1\}^{n \cdot s}$, which allegedly equals $(E_1(x), \ldots, E_s(x))$;
3. The PCPPs $(\widetilde{\bar{p}_1}(x), \ldots, \widetilde{\bar{p}_s}(x)) \in \{0,1\}^{\tilde{O}(t(k)) \cdot s}$, which allegedly equals $(\bar{p}_1(x), \ldots, \bar{p}_s(x))$.

We show that for every bundle $B(x)$, as in Construction 7.3, there exists a consistency test T that, for every $\varepsilon \geq 1/\mathsf{polylog}(\ell)$, makes $O(1/\varepsilon)$ queries to a string $w \in \{0,1\}^\ell$ and satisfies the following conditions.

1. If $w = B(x)$, then for every $i \in \{0\} \cup [s]$ it holds that $\Pr_T[T^w(i) = 1] = 1$.
2. If w is ε-far from B, then $\Pr[T^w(0) = 0] \geq 2/3$.
3. For every $i \in [s]$, if there exists $x \in \{0,1\}^k$ such that w is ε-close to $B(x)$ and $\widetilde{E_i}$ is ε-far from $E_i(x)$, then $\Pr[T^w(i) = 0] \geq 2/3$.

Let $\varepsilon \geq 1/\mathsf{polylog}(\ell)$, and assume without loss of generality that $\varepsilon < \delta(\mathsf{ECC})/2$.[14] For every $i \in [s]$ denote by V_i the PCPP verifier for the language

$$L_i = \{(a,b) \ : \ \exists x \in \{0,1\}^k \text{ such that } a = \mathsf{ECC}(x)^{r_a} \text{ and } b = E_i(x)^{r_b}\},$$

with respect to proximity parameter $\varepsilon/6$ and soundness $9/10$. Consider the ε-tester T that is given $i \in \{0\} \cup [s]$ and oracle access to $w = (\widetilde{\mathsf{ECC}(x)}, (\widetilde{E_i})_{i \in [s]}, (\widetilde{\bar{p}_i})_{i \in [s]}) \in \{0,1\}^\ell$ and accepts if both of the following tests accept.

1. Repetition Test: Query two random copies from the long-code part of w and check if they agree on a random location. More accurately, select uniformly at random $j, j' \in [r]$ and reject if and only if $\widetilde{\mathsf{ECC}}(x)_j$ and $\widetilde{\mathsf{ECC}}(x)_{j'}$ disagree on a *random* coordinate. Repeat this test $O(1/\varepsilon)$ times.
2. Consistency Test: Choose uniformly $j \in [r]$. If $i = 0$, choose uniformly $i' \in [s]$, otherwise set $i' = i$. Reject if the verifier $V_{i'}$ rejects on input $(\widetilde{\mathsf{ECC}}(x)_j^{r_a}, \widetilde{E_{i'}}(x)^{r_b})$ and proof $\widetilde{\bar{p}_{i'}}(x)$.

[14] The relative distance of ECC is constant, so if $\varepsilon \geq \delta(\mathsf{ECC})/2$, we can set the proximity parameter to $\delta(\mathsf{ECC})/2$, increasing the complexity by only a constant factor.

The first condition of Proposition 7.7 follows by construction. For the other conditions, first observe that if $\widetilde{\mathsf{ECC}}(x)$ is far from consisting of r identical copies, then the repetition test rejects with high probability. That is, let $\hat{c} \in \{0,1\}^{n'}$ be a string that is closest on average to the copies in $\widetilde{\mathsf{ECC}}(x)$, i.e., a string that minimizes $\Delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) = \sum_{j=1}^{r} \Delta(\widetilde{\mathsf{ECC}}(x)_j, \hat{c})$. Observe that

$$\mathop{\mathbf{E}}_{j,j' \in_R [r]}[\delta(\widetilde{\mathsf{ECC}}(x)_j, \widetilde{\mathsf{ECC}}(x)_{j'})] \geq \mathop{\mathbf{E}}_{j \in_R [r]}[\delta(\widetilde{\mathsf{ECC}}(x)_j, \widetilde{\mathsf{ECC}}(x))] = \delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r).$$

If $\delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) > \varepsilon/60$, then by invoking the codeword repetition test $O(1/\varepsilon)$ times, with probability at least $2/3$ one of the invocations will reject. Otherwise, note that with probability at least $9/10$ the random copy $\widetilde{\mathsf{ECC}}(x)_j$ is $\varepsilon/6$-close to $\hat{c}$; assume hereafter that this is the case.

If w is ε-far from B, then since $\widetilde{\mathsf{ECC}}(x) \geq (1-\varepsilon/2)\ell$, it follows that $\widetilde{\mathsf{ECC}}(x)$ is $\varepsilon/2$-far from ECC^r, and thus

$$\delta_{\mathsf{ECC}^r}(\hat{c}^r) \geq \delta_{\mathsf{ECC}^r}(\widetilde{\mathsf{ECC}}(x)) - \delta(\hat{c}^r, \widetilde{\mathsf{ECC}}(x)) = \varepsilon/2 - \varepsilon/60 > \varepsilon/3.$$

Recall that we assumed that $\delta(\widetilde{\mathsf{ECC}}(x)_j, \hat{c}) \leq \varepsilon/6$, and so $\delta_{\mathsf{ECC}}(\widetilde{\mathsf{ECC}}(x)_j) > \varepsilon/6$. Thus, $\Pr[V_{i'}^w = 0] \geq 9/10 \cdot 9/10$.

Finally, If there exists $x \in \{0,1\}^k$ such that w is ε-close to $B(x)$ and $\widetilde{E_i}(x)$ is ε-far from $E_i(x)$, then since $\delta(\widetilde{\mathsf{ECC}}(x), \hat{c}^r) \leq \varepsilon/60$, it follows that with probability at least $9/10$ the random copy $\widetilde{\mathsf{ECC}}(x)_j$ is $\varepsilon/6$-close to $\mathsf{ECC}(x)$. Hence, $(\widetilde{\mathsf{ECC}}(x)_j^{r_a}, \widetilde{E_i}(x)^{r_b})$ is at least $5\varepsilon/6$-far from L_i, and so $\Pr[V_i^w = 0] \geq 9/10 \cdot 9/10$.

7.7.1.2 Proof of Claim 7.7.1

We show that if w is ε-far from the subcode $\{C(x) \ : \ \varphi(x) = 1\}$, then for every alleged MAP proof $\widetilde{\pi}_\varphi$, it holds that $Pr[V_\varphi^w(\widetilde{\pi}_\varphi) = 0] \geq 2/3$. Assume, without loss of generality, that $\varepsilon < 1/3$. By Proposition 7.7, the consistency test (Step 2 of V_φ) rejects with probability $2/3$ unless there exists $x \in \{0,1\}^k$ such that: (1) the input w is ε-close to the codeword $C(x)$, and (2) for every $q \in Q$, the purported function $\widetilde{X}_q$ in w is ε-close to $\widehat{X}_q$, the low-degree extension of x to $\mathbb{F}_q$. Note that, in particular, the polynomial $\widehat{X}_q$ takes *binary* values over H^m (i.e., encodes a binary assignment). Since w is ε-far from the subcode $\{C(x) \ : \ \varphi(x) = 1\}$, this implies that the assignment x does not satisfy φ. In addition, we may also assume that for every $q \in Q$ the purported proof $\widetilde{\pi}_{\varphi,q}$ satisfies

$$\sum_{z_1,\ldots,z_\ell \in H} \widetilde{\pi}_{\varphi,q}(z_1,\ldots,z_\ell) \equiv n \pmod{q},$$

since otherwise the verifier rejects in Step 1.

On the other hand, observe that there exists $q^* \in Q$ such that

$$\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q^*}(z_1,\ldots,z_\ell) \not\equiv n \pmod{q^*}.$$

To see this, first recall that $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell)$ counts the number of clauses that the assignment satisfies, modulo q. Note that since the assignment is binary, then $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell) \leq n$, where the summation is over the integers, and that $\prod_{q\in Q} q > n$. Thus, if $\sum_{z_1,\ldots,z_\ell \in H} \pi_{\varphi,q}(z_1,\ldots,z_\ell)$ is congruent to n for all $q \in Q$, then by the Chinese remainder theorem, the assignment satisfies all n constraints, in contraction to our assumption.

Therefore, the total degree $m^2d^2t + md$ polynomials $\pi_{\varphi,q}$ and $\widetilde{\pi}_{\varphi,q}$ are not identical, and so, by the Schwartz-Zippel Lemma, they disagree on a randomly chosen point with probability at least $1 - \frac{(m^2d^2t+md)}{\mathbb{F}_q} \geq 9/10$.

To complete the argument, note that the (amplified) self-correctability of low-degree polynomials guarantees that every location in $\widehat{X}_q$ can be reconstructed from $\widetilde{X}_q$ with probability $1 - 1/10|H|^{m-\ell}$. Therefore, all points are read correctly with probability at least $9/10$, and thus, with probability $9/10 \cdot 9/10$, the verifier rejects (in Step 3) when checking whether $\pi_{\varphi,q}(r_1,\ldots,r_\ell)$ equals $\widetilde{\pi}_{\varphi,q}(r_1,\ldots,r_\ell)$.

7.7.1.3 Proof of Claim 7.9.1

We show that if $x \notin \Pi_\varphi$, then there exists $q \in Q$ such that $\Pr_{(J_q,\vec{v}_q)}[\widehat{X}_q(J_q) = \vec{v}_q] \leq (1/2)^S$. Fix $s \in S$. For every $q \in Q$, denote $J_{q,s} = \{\widehat{\varphi}_i(r_1^{(s)},\ldots,r_\ell^{(s)},\vec{z})\}_{\vec{z}\in H^{m-\ell}, i\in[t]}$ and $\vec{v}_{q,s} \in \mathbb{F}_q^{|H|^{m-\ell}\cdot t}$ such that $\vec{v}_{q,s}[\vec{z},i] = \vec{v}_q[s,\vec{z},i]$ for all $\vec{z} \in H^{m-\ell}$ and $i \in [t]$ (recall that $\vec{v}_q[s,z,i]$ allegedly equals $\widehat{X}_q \circ \widehat{\varphi}_i(r_1^{(s)},\ldots,r_\ell^{(s)},\vec{z})$). We first show that there exists $q \in Q$ such that $\Pr_{(J_{q,s},\vec{v}_{q,s})}[\widehat{X}_q(J_{q,s}) = \vec{v}_{q,s}] \leq 1/2$.

Similarly to the case in Theorem 7.2, observe that there exists $q \in Q$ such that $\sum_{z\in H^\ell} \pi_q(z) \not\equiv n \pmod q$, since otherwise, by the Chinese remainder theorem,

$$\sum_{z\in H^m} \sum_{i=1}^{n} \chi_i(z) \cdot \mathsf{c}_i\big(X\circ\varphi_1(z),\ldots,X\circ\varphi_t(z)\big) \equiv n \pmod{\prod_{q\in Q} q},$$

in contradiction to the assumption that $\varphi(x) = 0$; fix such $q \in Q$. Therefore the total degree $m^2d^2t + md$ polynomials π_q and π_q differ, and so, by the Schwartz-Zippel Lemma,

$$\Pr_{r_1^{(s)},\ldots,r_\ell^{(s)}\in\mathbb{F}_q}[\pi_q(r_1^{(s)},\ldots,r_\ell^{(s)}) \neq \pi_q(r_1^{(s)},\ldots,r_\ell^{(s)})] \geq 1 - \frac{m^2d^2t+md}{\mathbb{F}_q} \geq 9/10.$$

In other words, it holds that $\Pr_{(J_{q,s},\vec{v}_{q,s})}[\widehat{X}_q(J_{q,s}) = \vec{v}_{q,s}] < 1/10$. Finally, since $\{(J_{q,s},\vec{v}_{q,s})\}_{s\in[S]}$ are independently selected, it holds that

$$\Pr_{(J_q,\vec{v}_q)}[\widehat{X}_q(J_q) = \vec{v}_q] = \left(\Pr_{(J_{q,s},\vec{v}_{q,s})}[\widehat{X}_q(J_{q,s}) = \vec{v}_{q,s}]\right)^S \leq \left(\frac{1}{10}\right)^S.$$

This concludes the proof of Claim 7.9.1.

Chapter 8

Appendix: Brief Descriptions of Works not included in this Thesis

In this section we provide a high-level description of results obtained during our doctoral studies, which were not included above. See the links provided below for the full versions.

8.1 Relaxed Locally Correctable Codes

In a joint work with Govind Ramnarayan and Ron Rothblum [GRR17], we studied the extension of the notion of relaxed decodability to *locally correctable codes* (LCC). Recall that locally decodable codes (LDCs) and locally correctable codes (LCCs) are error-correcting codes in which individual bits of the message and codeword, respectively, can be recovered by reading only few bits from a noisy codeword. These codes have found numerous applications both in theory and in practice.

A natural relaxation of LDCs, introduced by Ben-Sasson *et al.* (SICOMP, 2006), allows the decoder to reject (i.e., refuse to answer) in case it detects that the codeword is corrupt. They call such a decoder a *relaxed decoder* and construct a constant-query relaxed LDC with rate that is sub-exponentially better than what is known for (full-fledged) LDCs in this regime.

We considered an analogous relaxation for local *correction*. Thus, a *relaxed local corrector* reads only few bits from a (possibly) corrupt codeword and either recovers the desired bit of the codeword, or rejects in case it detects a corruption.

We gave two constructions of relaxed LCCs in two regimes, where the first optimizes the query complexity and the second optimizes the rate:

1. **Constant Query Complexity:** A relaxed LCC with polynomial blocklength whose corrector only reads a constant number of bits of the codeword. This is a sub-exponential improvement over the best *constant query* (full-fledged) LCCs that are known.

2. **Constant Rate:** A relaxed LCC with *constant rate* (i.e., linear blocklength) with quasi-polylogarithmic query complexity (i.e., $(\log n)^{O(\log \log n)}$). This is a nearly

sub-exponential improvement over the query complexity of a recent (full-fledged) constant-rate LCC of Kopparty *et al.* (STOC, 2016).

To this prove these results, we constructed self-correctable, robust *strong canonical* PCPs of proximity, which we believe may also be of independent interest.

8.2 An Adaptivity Hierarchy Theorem for Interactive Proofs of Proximity

In a joint work with Clément Canonne [CG17], we studied the role of adaptivity in property testing. Adaptivity is known to play a crucial role in property testing. In particular, there exist properties for which there is an exponential gap between the power of *adaptive* testing algorithms, wherein each query may be determined by the answers received to prior queries, and their *non-adaptive* counterparts, in which all queries are independent of answers obtained from previous queries.

In this work, we investigated the role of adaptivity in property testing at a finer level. We first quantified the degree of adaptivity of a testing algorithm by considering the number of "rounds of adaptivity" it uses. More accurately, we say that a tester is k-(round) adaptive if it makes queries in $k+1$ rounds, where the queries in the $(i+1)$'st round may depend on the answers obtained in the previous i rounds. Then, we asked the following question:

Does the power of testing algorithms smoothly grow with the number of rounds of adaptivity?

We provided a positive answer to the foregoing question by proving an adaptivity hierarchy theorem for property testing. Specifically, our main result shows that for every $n \in \mathbb{N}$ and $0 \leq k \leq n^{0.99}$ there exists a property $\Pi_{n,k}$ of functions for which (1) there exists a k-adaptive tester for $\Pi_{n,k}$ with query complexity $\tilde{O}k$, yet (2) any $(k-1)$-adaptive tester for $\Pi_{n,k}$ must make $\Omega(n)$ queries. In addition, we showed that such a qualitative adaptivity hierarchy can be witnessed for testing natural properties of graphs.

8.3 Distribution Testing Lower Bounds via Reductions from Communication Complexity

In a joint work with Eric Blais and Clément Canonne [BCG16], we presented a new methodology for proving distribution testing lower bounds, establishing a connection between distribution testing and the simultaneous message passing (SMP) communication model. Extending the framework of Blais, Brody, and Matulef (CCC 2011), we showed a simple way to reduce (private-coin) SMP problems to distribution testing problems. This method allowed us to prove several new distribution testing lower bounds, as well as to provide simple proofs of known lower bounds.

Our main result is concerned with testing identity to a specific distribution p, given as a parameter. In an influential work, Valiant and Valiant (FOCS 2014) showed that the sample complexity of the aforementioned problem is closely related to the $\ell_{2/3}$-quasinorm of p. We obtained alternative bounds on the complexity of this problem in terms of an arguably more intuitive measure and using simpler proofs. More specifically, we proved that the sample complexity is essentially determined by a fundamental operator in the theory of interpolation of Banach spaces, known as Peetre's *K-functional*. We showed that this quantity is closely related to the size of the effective support of p (loosely speaking, the number of supported elements that constitute the vast majority of the mass of p). This result, in turn, stems from an unexpected connection to functional analysis and refined concentration of measure inequalities, which arise naturally in our reduction.

8.4 Testing Booleanity and the Uncertainty Principle

In a joint work with Omer Tamuz [GT13], we showed a structural result regarding Boolean functions, which admits efficient testers for Booleanity, in certain settings of parameters.

More specifically, a real function on the hypercube $f : \{0,1\}^n \to \mathbb{R}$ is said to be Boolean if its image is in $\{0,1\}$. We showed that every function on the hypercube with a sparse Fourier expansion must either be Boolean or far from Boolean. In particular, we showed that a multilinear polynomial with at most k terms must either be Boolean, or output values different than 0 or 1 for a fraction of at least $2/(k+2)^2$ of its domain.

It follows that given oracle access to f, together with the guarantee that its representation as a multilinear polynomial has at most k terms, one can test Booleanity using $O(k^2)$ queries. We also showed an $\Omega(k)$ queries lower bound for this problem.

Our proof crucially uses Hirschman's entropic version of Heisenberg's uncertainty principle.

Bibliography

[AFNS06] Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. In *STOC*, pages 251–260, 2006.

[AGH90] William Aiello, Shafi Goldwasser, and Johan Håstad. On the power of interaction. *Combinatorica*, 10(1):3–25, 1990.

[AKNS00] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An O(n log n) sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9, 1983.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1:2:1–2:54, February 2009.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429. ACM, 1985.

[BBM11] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. In *IEEE Conference on Computational Complexity*, pages 210–220, 2011.

[BCG16] Eric Blais, Clément Canonne, and Tom Gur. Alice and bob show distribution testing lower bounds. 2016. ECCC.

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[BFS86] Laszlo Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 337–347, Washington, DC, USA, 1986. IEEE Computer Society.

[BGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 113–131. ACM, 1988.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the P=?NP question. *SIAM Journal on computing*, 4(4):431–442, 1975.

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[BHR05] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3CNF properties are hard to test. *SIAM J. Comput.*, 35(1):1–21, 2005.

[Bla10] Eric Blais. Testing juntas: A brief survey. In Goldreich [Gol10b], pages 32–40.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[Bol05] Beate Bollig. Property testing and the branching program size of boolean functions. In *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, pages 258–269, 2005.

[BS05] Eli Ben-Sasson and Madhu Sudan. Simple PCPs with poly-log rate and query complexity. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 266–275. ACM, 2005.

[BS06] Eli Ben-Sasson and Madhu Sudan. Robust locally testable codes and products of codes. *Random Structures & Algorithms*, 28(4):387–402, 2006.

[BSGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BT04] Andrej Bogdanov and Luca Trevisan. Lower bounds for testing bipartiteness in dense graphs. In *IEEE Conference on Computational Complexity*, pages 75–81, 2004.

[BV12] Eli Ben-Sasson and Michael Viderman. Towards lower bounds on locally testable codes via density arguments. *Computational Complexity*, 21(2):267–309, 2012.

[BY17] Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing*. Forthcoming, 2017.

[BYKS01] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *STOC*, pages 266–275, 2001.

[Can15] Clément L. Canonne. A Survey on Distribution Testing: your data is Big. But is it Blue? 22:63, 2015.

[CCG+94] Richard Chang, Benny Chor, Oded Goldreich, Juris Hartmanis, Johan Håstad, Desh Ranjan, and Pankaj Rohatgi. The random oracle hypothesis is false. *Journal of Computer and System Sciences*, 49(1):24–39, 1994.

[CCGT13] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. *arXiv preprint arXiv:1304.3816*, 2013.

[CCGT14] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–706. SIAM, 2014.

[CCM09] Amit Chakrabarti, Graham Cormode, and Andrew Mcgregor. Annotations in data streams. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 222–234, Berlin, Heidelberg, 2009. Springer-Verlag.

[CCM+15] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and Arthur–Merlin communication. In *30th Conference on Computational Complexity (CCC 2015)*, volume 33, pages 217–243. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[CDI+13] Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 185–202, 2013.

[CEG95] Ran Canetti, Guy Even, and Oded Goldreich. Lower bounds for sampling algorithms for estimating the average. *Inf. Process. Lett.*, 53(1):17–25, 1995.

[CG17] Clément Canonne and Tom Gur. An adaptivity hierarchy theorem for property testing. 2017. Submitted.

[CGI+16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270, 2016.

[CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[CGR+12] Artur Czumaj, Oded Goldreich, Dana Ron, C Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Structures & Algorithms*, 2012.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.

[CMT13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

[CR11] Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 51–60, New York, NY, USA, 2011. ACM.

[Din07a] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[Din07b] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM (JACM)*, 54(3):12, 2007.

[DK11] Irit Dinur and Tali Kaufman. Dense locally testable codes cannot have constant rate and distance. In *APPROX-RANDOM*, pages 507–518, 2011.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[DTV15] Samira Daruki, Justin Thaler, and Suresh Venkatasubramanian. Streaming verification in data analysis. *arXiv preprint arXiv:1509.05514*, 2015.

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM Journal on Computing*, 41(6):1694–1703, 2012.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004.

[FGL+91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete. 1991.

[FGL14] Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 483–500, 2014.

[FLM+12] Eldar Fischer, Oded Lachish, Arie Matsliah, Ilan Newman, and Orly Yahalom. On the query complexity of testing orientations for being eulerian. *ACM Transactions on Algorithms*, 8(2):15, 2012.

[FLV15] Eldar Fischer, Oded Lachish, and Yadu Vasudev. Trading query complexity for sample-based testing and multi-testing scalability. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1163–1182. IEEE, 2015.

[FS88] Lance Fortnow and Michael Sipser. Are there interactive protocols for CO-NP languages? *Inf. Process. Lett.*, 28(5):249–251, 1988.

[FS95] Katalin Friedl and Madhu Sudan. Some improvements to total degree tests. In *Theory of Computing and Systems, 1995. Proceedings., Third Israel Symposium on the*, pages 190–198. IEEE, 1995.

[Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004.

[GG16a] Oded Goldreich and Tom Gur. Universal locally testable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:42, 2016.

[GG16b] Oded Goldreich and Tom Gur. Universal locally verifiable codes and 3-round interactive proofs of proximity for CSP. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:192, 2016.

[GGK14] Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:25, 2014.

[GGK15] Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 1–41, 2015.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[GGR15] Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:24, 2015.

[GK92] Oded Goldreich and Hugo Krawczyk. Sparse pseudorandom distributions. *Random Struct. Algorithms*, 3(2):163–174, 1992.

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

[GKS13] Alan Guo, Swastik Kopparty, and Madhu Sudan. New affine-invariant codes from lifting. In *ITCS*, pages 529–540. ACM, 2013.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMS87] Oded Goldreich, Yishay Mansour, and Michael Sipser. Interactive proof systems: Provers that never fail and random selection (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 449–461, 1987.

[Gol99] Oded Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1999.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[Gol10a] Oded Goldreich. On testing computability by small width obdds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 574–587. Springer, 2010.

[Gol10b] Oded Goldreich, editor. *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*. Springer, 2010.

[Gol10c] Oded Goldreich. Short locally testable codes and proofs: A survey in two parts. In *Property Testing* [Gol10b], pages 65–104.

[Gol11a] Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 470–506. Springer, 2011.

[Gol11b] Oded Goldreich. Valiant's polynomial-size monotone formula for majority. http://www.wisdom.weizmann.ac.il/~oded/PDF/mono-maj.pdf, 2011.

[Gol14] Oded Goldreich. On multiple input problems in property testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 704–720, 2014.

[Gol16] Oded Goldreich. *Introduction to Property Testing*. forthcoming (http://www.wisdom.weizmann.ac.il/~oded/pt-intro.html), 2016.

[Gol17] Oded Goldreich. *Introduction to Property Testing*. Forthcoming, 2017.

[GPW15a] Mika Göös, Toniann Pitassi, and Thomas Watson. The landscape of communication complexity classes. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:49, 2015.

[GPW15b] Mika Göös, Toniann Pitassi, and Thomas Watson. Zero-information protocols and unambiguity in Arthur-Merlin communication. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 113–122, 2015.

[GR62] Seymour Ginsburg and H Gordon Rice. Two families of languages related to ALGOL. *Journal of the ACM (JACM)*, 9(3):350–371, 1962.

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GR05] Venkatesan Guruswami and Atri Rudra. Tolerant locally testable codes. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 306–317. Springer, 2005.

[GR09] Oded Goldreich and Dana Ron. On proximity oblivious testing. In *STOC*, pages 141–150, 2009.

[GR11] Oded Goldreich and Dana Ron. On proximity-oblivious testing. *SIAM Journal on Computing*, 40(2):534–566, 2011.

[GR13a] Oded Goldreich and Dana Ron. On sample-based testers. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:109, 2013.

[GR13b] Tom Gur and Ran Raz. Arthur-Merlin streaming complexity. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, 2013.

[GR13c] Tom Gur and Ron Rothblum. Non-interactive proofs of proximity. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:78, 2013.

[GR14] Oded Goldreich and Dana Ron. On learning and testing dynamic environments. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 336–343, 2014.

[GR15a] Oded Goldreich and Dana Ron. On sample-based testers. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 337–345, 2015.

[GR15b] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 133–142. ACM, 2015.

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. 2017. The 8th Innovations in Theoretical Computer Science (ITCS 2017) conference (to appear).

[GRR17] Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. 2017. In Submission.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, pages 59–68, 1986.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

[GS10a] Dmitry Gavinsky and Alexander A Sherstov. A separation of NP and coNP in multiparty communication complexity. *arXiv preprint arXiv:1004.0817*, 2010.

[GS10b] Oded Goldreich and Or Sheffet. On the randomness complexity of property testing. *Computational Complexity*, 19(1):99–133, 2010.

[GS12] Oded Goldreich and Igor Shinkar. Two-sided error proximity oblivious testing - (extended abstract). In *APPROX-RANDOM*, pages 565–578, 2012.

[GS13] Lior Gishboliner and Asaf Shapira. Deterministic vs non-deterministic graph property testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:59, 2013.

[GT13] Tom Gur and Omer Tamuz. Testing booleanity and the uncertainty principle. *Chicago Journal of Theoretical Computer Science*, Article 14, 2013.

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[HLNT05] Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing orientation properties. *Electronic Colloquium on Computational Complexity (ECCC)*, 2005.

[HLNT07] Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing properties of constraint-graphs. In *IEEE Conference on Computational Complexity*, pages 264–277, 2007.

[HMP06] Shlomo Hoory, Avner Magen, and Toniann Pitassi. Monotone circuits for the majority function. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28-30 2006, Proceedings*, pages 410–425, 2006.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[Kla03] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 118–134. IEEE, 2003.

[Kla11] Hartmut Klauck. On Arthur Merlin games in communication complexity. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 189–199. IEEE, 2011.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pages 536–547, 2008.

[KR09] Yael Kalai and Guy N. Rothblum. Constant-round interactive proofs for NC^1. Unpublished observation, 2009.

[KR14] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity. Manuscript, 2014.

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 422–442, 2015.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, pages 565–574, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014.

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

[KS08] Tali Kaufman and Madhu Sudan. Algebraic property testing: the role of invariance. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing (STOC)*, pages 403–412. ACM, 2008.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.

[KV10] Tali Kaufman and Michael Viderman. Locally testable vs. locally decodable codes. In *APPROX-RANDOM*, pages 670–682, 2010.

[KW88] Klaus Kriegel and Stephan Waack. Lower bounds on the complexity of real-time branching programs. *ITA*, 22(4):447–459, 1988.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Inf. Process. Lett.*, 17(4):215–217, 1983.

[Lev85] Leonid A. Levin. One-way functions and pseudorandom generators. In *STOC*, pages 363–365, 1985.

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

[LSH65] Philip M. Lewis, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *SWCT (FOCS)*, pages 191–202, 1965.

[LV12] László Lovász and Katalin Vesztergombi. Nondeterministic graph property testing. *arXiv preprint arXiv:1202.5337*, 2012.

[Mei13] Or Meir. IP = PSPACE using error-correcting codes. *SIAM J. Comput.*, 42(1):380–403, 2013.

[Mei14] Or Meir. Locally correctable and testable codes approaching the singleton bound. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:107, 2014.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Information processing letters*, 39(2):67–71, 1991.

[New02] Ilan Newman. Testing membership in languages that have small width branching programs. *SIAM Journal on Computing*, 31(5):1557–1570, 2002.

[New10] Ilan Newman. Property testing of massively parametrized problems - a survey. In *Property Testing*, pages 142–157, 2010.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994.

[PRR01] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Testing parenthesis languages. In *RANDOM-APPROX*, pages 261–272, 2001.

[PRR06] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

[Raz87] A. Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. Notes of the Academy of Science of the USSR: 41(4) : 333-338, 1987.

[Ron08] Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

[Ron09] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[Rot09] Guy N. Rothblum. *Delegating computation reliably: paradigms and constructions*. PhD thesis, Massachusetts Institute of Technology, 2009.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[RTVV98] Ran Raz, Gábor Tardos, Oleg Verbitsky, and Nikolai Vereshagin. Arthur-Merlin games in boolean decision trees. In *Computational Complexity, 1998. Proceedings. Thirteenth Annual IEEE Conference on*, pages 58–67. IEEE, 1998.

[Ruz81] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981.

[RVW13] Guy N. Rothblum, Salil Vadhan, and Avi Wigderson. Interactive proofs of proximity: Delegating computation in sublinear time. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing (STOC)*, 2013.

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

[She11] Alexander A. Sherstov. The communication complexity of gap hamming distance. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:63, 2011.

[She12] Alexander A Sherstov. The multiparty communication complexity of set disjointness. In *Proceedings of the 44th symposium on Theory of Computing*, pages 525–548. ACM, 2012.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 77–82, New York, NY, USA, 1987. ACM.

[Sud92] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 1992. UMI Order No. GAX93-30747.

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs anf the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995.

[Tha16] Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 17:1–17:14, 2016.

[Tre04] Luca Trevisan. Some applications of coding theory in computational complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004.

[Vad00] Salil P. Vadhan. On transformation of interactive proofs that preserve the prover's complexity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 200–207, 2000.

[Val84] Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984.

[Vid11] Thomas Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:51, 2011.

[Vid12] Michael Viderman. A combination of testability and decodability by tensor products. In *APPROX-RANDOM*, pages 651–662, 2012.

[Vid13] Michael Viderman. Strong LTCs with inverse poly-log rate and constant soundness. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:22, 2013.

[Vio09] Emanuele Viola. The sum of d small-bias generators fools polynomials of degree d. *Computational Complexity*, 18(2):209–217, 2009.

[Wil16] Richard Ryan Williams. Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 2:1–2:17, 2016.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM*, 55(1):1, 2008.

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.