Robust PCPs of Proximity and Shorter PCPs

by

Prahladh Harsha

Bachelor of Technology (Computer Science and Engineering),
Indian Institute of Technology, Madras, India. (1998)
S.M. (Electrical Engineering and Computer Science),
Massachusetts Institute of Technology, Cambridge MA. (2000)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY.

September 2004

Author ______________________________
Department of Electrical Engineering and Computer Science
September 1, 2004

Certified by ______________________________
Madhu Sudan
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ______________________________
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Robust PCPs of Proximity and Shorter PCPs

by

Prahladh Harsha

Submitted to the
Department of Electrical Engineering and Computer Science
on September 1, 2004 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy.

Abstract

Probabilistically Checkable Proofs (PCPs) provide a format of rewriting and verifying mathematical proofs that allow efficient probabilistic verification based on probing very few bits of the rewritten proof. The celebrated PCP Theorem asserts that probing a constant number of bits suffices (in fact just 3 bits suffice). A natural question that arises in the construction of PCPs is by how much does this encoding blow up the original proof while retaining low query complexity.

We continue the study of the trade-off between the length of PCPs and their query complexity, establishing the following main results (which refer to proofs of satisfiability of circuits of size n):

1. We present PCPs of length $\exp(o(\log\log n)^2) \cdot n$ that can be verified by making $o(\log\log n)$ Boolean queries.

2. For every $\varepsilon > 0$, we present PCPs of length $\exp(\log^\varepsilon n) \cdot n$ that can be verified by making a constant number of Boolean queries.

In both cases, false assertions are rejected with constant probability (which may be set to be arbitrarily close to 1). The multiplicative overhead on the length of the proof, introduced by transforming a proof into a probabilistically checkable one, is just quasi-polylogarithmic in the first case (of query complexity $o(\log\log n)$), and $2^{(\log n)^\varepsilon}$, for any $\varepsilon > 0$, in the second case (of constant query complexity).

Our techniques include the introduction of a new variant of PCPs that we call "Robust PCPs of proximity". These new PCPs facilitate proof composition, which is a central ingredient in construction of PCP systems. Our main technical contribution is a construction of a "length-efficient" Robust PCP of proximity.

We also obtain analogous quantitative results for locally testable codes. In addition, we introduce a relaxed notion of locally decodable codes, and present such codes mapping k information bits to codewords of length $k^{1+\varepsilon}$, for any $\varepsilon > 0$.

Thesis Supervisor: Madhu Sudan
Title: Professor of Computer Science and Engineering

Credits

This thesis is a result of joint work with Eli Ben-Sasson, Oded Goldreich, Madhu Sudan and Salil Vadhan. I thank all my co-authors for this collaboration. A preliminary version of the results in this thesis appeared in [BGH$^+$04a]. A more elaborate version of these results can be found at [BGH$^+$04b]. Some of the results mentioned in Chapter 5 (Propositions 5.3.4 and 5.3.5) are from the work of Eli Ben-Sasson and Madhu Sudan [BS04]. I am thankful to them for letting me include these results in this thesis.

I would also like to thank the many people who made useful suggestions to me in my research. I would especially like to thank Eli Ben-Sasson, Irit Dinur, Oded Goldreich, Sofya Raskhodnikova, Alon Rosen, Mike Sipser, Madhu Sudan, Salil Vadhan, and Avi Wigderson.

My thesis committee consisted of my advisor Madhu Sudan, Shafi Goldwasser and Mike Sipser.

Acknowledgments

This thesis would not have been possible but for the valuable advice and direction provided by Madhu Sudan, Eli Ben-Sasson and Oded Goldreich. I thank my advisor, Madhu Sudan for his invaluable guidance and support. I have greatly benefited from my collaboration with Eli over the the past four years, who has always been there to discuss and clarify any matter. I thank Oded Goldreich for his advice, direction and encouragement during our interaction over the last one year.

I would also like to thank the many people with whom I had useful discussions during my research. I would especially like to thank Sofya Raskhodnikova, Mike Sipser, and Salil Vadhan.

I am indebted to the the efforts of my math teachers – Kotteswara Rao, Sri Ramaiah, Sheshayya Choudum and ammā, who instilled in me the love for mathematics.

I owe a lot to the research atmosphere at CSAIL. I also thank all the members of the theory group for having created a very conducive atmosphere for me to work in. Special mention to Alantha Newman, my office-mate of several years. I also greatly benefited from a summer at the NEC Laboratories in the excellent company of Irit Dinur, Lance Fortnow, Joe Kilian, Kobbi Nissim and Ronnit Rubinfeld.

Special Thanks to Mike Sipser both for patiently listening to me during various stages of my research and offering me a teaching assistantship for his excellent course "Theory of Computation".

I thank my apartment-mates over several years – Kripa Varanasi, Sridhar Ramachandran, Ramachandran Balakrishnan, and Rajappa Tadepalli for having created a home away from home. My friends in Boston – Amit Deshpande, Krishnan Sriram and several others; have been very encouraging and greatly helped me get over my occasional blues.

Finally, no words can suffice to thank ammā, appā, Pavithra, Ramesh and Ajay for their endless encouragement and who keep me going.

To
அம்மா (ammā) and அப்பா (appā)

Contents

List of Figures

CHAPTER 1

Introduction

The notion of a *proof* is one of the most fundamental concepts in mathematics. A proof is the means by which one person convinces another the truth of an assertion. But how does one convince another of the truth of an assertion? In other words, *What is a* valid *proof?* For instance, do we consider any of the following as plausible proofs of some assertion T?

Assertion T is true since we have never observed it being falsified.

Assertion T is true since it has been published in journal J.

Assertion T is true because Richard Karp believes it to be so.

One of the attempts to formalize the concept of a proof is via the twin-notions of a *proof-system* and *proof-verification*. A proof-system specifies a schema of allowable rules while proof-verification involves checking if a proof satisfies this schema of rules. Thus, the person who provides the proof (henceforth called the *prover*) writes the proof according to the rules specified by the proof-system while the person verifying the proof (henceforth called the *verifier*) checks if the proof is written according to these set of rules (i.e., checks if the proof is valid). Furthermore, we require the proof-system to satisfy the following two properties - (a) For every true assertion, there exists a valid proof in the proof-system, i.e, all true assertions have proofs. We call this requirement the *completeness* of the proof-system. (b) For every false assertion, there does not exist a proof in the proof-system, i.e., false assertions do not have proofs. We call this the *soundness* of the proof-system.

The notion of a proof-system implicitly implies the existence of two parties - the prover and the verifier. Is it necessary to have these two parties, do we need the prover? If the prover P can produce the proof of the assertion, why can't the verifier V produce the proof himself and convince himself of the truth of the assertion without any aid from the prover P?

Ideally, we believe that it is much harder to find the proof of an assertion than to check the correctness of the proof once it is found. Thus, while it is the task of the prover P to derive a valid proof for the assertion, the verifier V merely has to check the validity of the proof. Thus, proofs can be viewed as a labor-saving mechanism. We believe that there exist assertions that take require a huge labor from the prover's side to obtain a valid proof, however, once such a valid proof is found, it is possible to express it in sufficiently simple terms such that even a weak verifier V can check its validity.

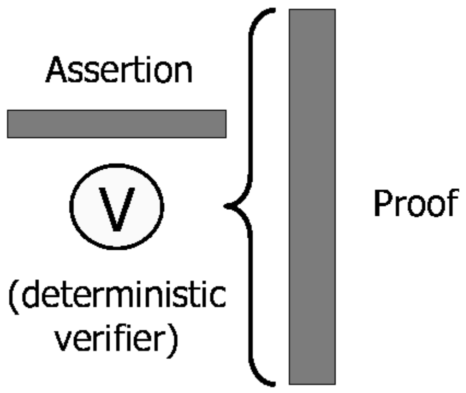

Figure 1-1: Pictorial Representation of a Proof System
A deterministic verifier checks the truth of an assertion by reading every bit of the proof.

1.1 *Complexity Theory via Proofs*

Several concepts in Computational Complexity can be phrased in the language of proofs. The two important computational tasks associated with a proof-system are proof-production (i.e, the work of the prover, the task of actually deriving the proof) and proof-verification (i.e., the work of the verifier). A host of complexity classes can be described by considering the complexity of either of these tasks. The complexity class $\mathbf{P}$ captures precisely the class of family of assertions whose proofs can be obtained in polynomial time in the length of the assertion. On the other hand, the complexity class $\mathbf{NP}$ is the class of family of assertions that have proofs whose validity can be checked by the verifier in polynomial time in the length of the assertion. In this terminology, the question, we had asked earlier, of whether there exist a family of assertions whose proofs can be verified in polynomial time but cannot be obtained in polynomial time is precisely the question "$\mathbf{P} \neq \mathbf{NP}$?", the most famous open-problem today in computer science.

Similar to $\mathbf{NP}$ (and $\mathbf{P}$), we can define the class $\mathbf{NEXP}$ (and $\mathbf{EXP}$) which is precisely the class of family of assertions whose proofs can be verified (produced) in time exponential in the length of the assertion. When we consider the space-complexity of the prover and verifier instead of time-complexity, we obtain the various space-complexity classes like $\mathbf{PSPACE}$, $\mathbf{L}$ etc. For instance, the class deterministic polynomial space, called $\mathbf{PSPACE}$, is the class of family of assertions whose proofs can be obtained by a prover using space polynomial in the length of the assertion. Thus most of the concepts in complexity theory can be expressed in the language of proofs.

The class $\mathbf{NP}$, by definition, is the class of family of assertions whose truth can be verified by a deterministic polynomial time verifier. Goldwasser, Micali and Rackoff [GMR89] studied whether a larger class of assertions could be verified by giving the verifier slightly more power. This led to the notion of *interactive proofs*, called $\mathbf{IP}$, due independently to Goldwasser, Micali and Rackoff [GMR89] and Babai [Bab85]. The class $\mathbf{IP}$ is the class of family of assertions that can be verified by a *probabilistic* polynomial time verifier (as opposed to a deterministic polynomial time verifier) that *interacts* with the prover (as opposed to receiving a fixed proof of membership). In other words, this model allows the verifier to both toss random coins and to talk with the prover (i.e., ask the prover questions). Following the work of Lund, Fortnow, Karloff and Nisan [LFKN92], Shamir [Sha92] proved the surprising result that $\mathbf{IP} = \mathbf{PSPACE}$. In other words, proofs of assertions that require polynomial space to construct, can be verified in probabilistic polynomial time by a verifier that interacts with the prover. The work of Goldwasser, Micali and Rackoff [GMR89] also introduced the notion of *zero-knowledge proofs*, proofs that reveal nothing more than the validity of the assertion. A result due to Goldreich, Micali and Wigderson [GMW91] shows that any problem in $\mathbf{NP}$ has a zero-knowledge proof-system, assuming that a certain cryptographic assumption is true.

Ben-Or *et al.* [BGKW88] considered a model of interactive proofs where the verifier can interact with more than one prover, who are all computationally unbounded but unable to communicate with each other once the interaction with the verifier begins. Their main result was that any family of assertions in $\mathbf{NP}$ has a zero-knowledge proof in this multi-prover model (without any cryptographic assumption). Fortnow, Rompel and Sipser [FRS94] further studied this model of multi-prover interactive proofs ($\mathbf{MIP}$) and showed that any family of assertions in MIP can be characterized as those family of assertions that have exponentially long proofs, which can however be checked by a probabilistic polynomial time verifier. Shortly after the $\mathbf{IP} = \mathbf{PSPACE}$ result by Shamir [Sha92], Babai, Fortnow and Lund [BFL91] proved the surprising result that $\mathbf{MIP} = \mathbf{NEXP}$. Combined with the characterization of Fortnow, Rompel and Sipser [FRS94], this is a very surprising result, because it says that any proof whose validity can be checked in exponential time, can be re-written in such a way that a probabilistic polynomial time verifier can check the validity of the new proof. The verifier, being probabilistic, accepts valid proofs with probability 1 and rejects invalid proofs with probability at least $\frac{1}{2}$. Thus, the verifier is convinced of the validity of the (exponentially long) proof even though it looks at only a negligible (at most polynomial) fraction of the proof.

It is natural to ask whether the above result scales down for polynomial-time verifiable proofs. In other words, can proofs of assertions which are verifiable in deterministic polynomial time be rewritten in such a way that they are checkable by a probabilistic poly-logarithmic time verifier? On the face of it, this question seems meaningless since how can a verifier verify the proof of an assertion without reading the assertion in its entirety (it takes linear time to read the assertion)! However, Babai *et al.* [BFLS91] show that poly-logarithmic time verification is in fact possible provided both the assertion and the proof are encoded suitably. For the same reasons as before, this result is surprising since the verifier reads at most poly-logarithmic bits in the polynomially-long proof, yet is convinced of the validity of (a polynomially long) proof. Given this result, it is natural to ask how many bits of the proof does the probabilistic verifier need to read in order to check the validity of the proof. This leads us to the notion of probabilistically checkable proofs.

1.2 Probabilistically Checkable Proofs

Probabilistically Checkable Proofs [FGL$^+$96, AS98, ALM$^+$98] (a.k.a Holographic proofs [BFLS91]) provide a format of rewriting and verifying proofs that allow efficient probabilistic verification based on probing very few bits of the the rewritten proof. The celebrated PCP Theorem [AS98, ALM$^+$98] asserts that any **NP** proof can be rewritten in such a manner that it is checkable by a probabilistic verifier (as opposed to a deterministic verifier in the usual **NP** setting) by probing merely a constant number of bits in the new proof. For obvious reasons, we will refer to the new proof as a Probabilistically Checkable Proof (PCP). Furthermore, it turns out that there exist PCPs for **NP** proofs that are checkable by a mere three bits; rejecting proofs of false assertions with probability almost $1/2$ (and accepting proofs of valid assertions with probability 1 (cf. [Hås01, GLST98]).

Informally speaking, the PCP Theorem states that for every family of assertions in **NP**, there exists a probabilistic verifier V that checks the validity of the proof of the assertion by probing the proof in only a constant number of locations. The verifier V does so by tossing a logarithmic number of random coins in the size of the assertion, probing the proof at a constant number of bit positions and accepting the proof based on a boolean verdict depending on the assertion, the random coins tossed and the bits read by it. The number of random coins tossed by the verifier is referred to as *randomness complexity (r)* while the maximum number of probes performed by the verifier is referred to as the *query complexity (q)* of the proof-system. Besides these two parameters, the *length* of the PCP is another important parameter. It can easily be observed that the length of the PCP is atmost $q \cdot 2^r$. Thus, since the randomness complexity is logarithmic in the size of the original

NP proof and the query complexity a constant, the new PCP is at most polynomially longer than the original **NP** proof. The PCP Theorem is constructive in the sense that it indicates how to construct this new probabilistically checkable proof (PCP) from the old **NP** proof. Furthermore, this construction is both complete and sound, i.e., valid proofs are always accepted while invalid proofs are accepted with probability at most 1/2.

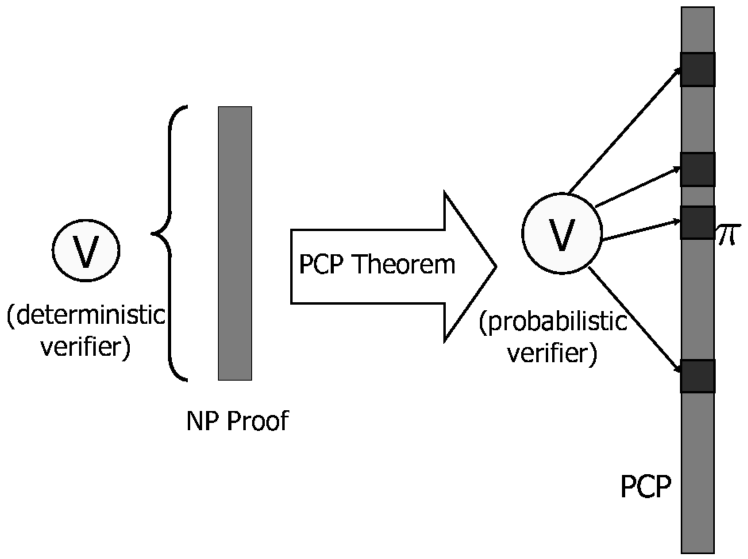

Figure 1-2: The PCP Theorem
Any **NP** proof can be rewritten into a PCP checkable by a probabilistic verifier in at most a constant number of bit locations.

PCPs have had a tremendous impact on the study of combinatorial optimization problems in the last decade. Starting with the seminal result of Feige *et al.* [FGL+96] which demonstrates the connection between this model of proof verification and the hardness of approximation of several combinatorial optimization problems, PCPs have been very effectively used to show that the approximation version of several **NP**-hard combinatorial optimization problems are themselves **NP**-hard. This is a very active area of research and some of the notable papers in this direction include (just to mention a few) [PY91, FGL+96, AS98, ALM+98, LY94, BGLR93, BGS98, Fei98, Hås99, Hås01, Kho04]. However, we will not dwell into this hardness connection of PCPs in this dissertation.

1.3 Contributions of Thesis

1.3.1 Short PCPs

The main focus of this thesis is the following question. *How much longer is the new PCP compared to the original* **NP** *proof while retaining low query complexity?* As mentioned before, the PCP Theorem guarantees that the new proof is at most polynomially larger than the original proof. The main question we address in this thesis is the following: how small can this polynomial be? Can we construct PCPs which are at most a constant times larger than the original **NP** proof retaining constant query complexity. Furthermore, can such a constant blowup be attained with the optimal query complexity of 3 bits?

It is to be noted that historically, in the study of PCPs, optimizing the query complexity has attracted a lot more attention, cheifly motivated by the significance of query complexity to inapproximabilty results (see, for example, [BGLR93, BGS98, Hås01, GLST98, ST00]). However, these works only guarantee that the PCP is of length polynomial in the length of the original **NP** proof.[1] Just to put things in perspective, the original proof of the PCP Theorem [ALM$^+$98] constructed PCPs of nearly cubic length with a query complexity roughly of the order of a million (in order to reject proofs of false assertion with probability at least $1/2$). On the other hand, the 3-query optimal PCPs of [Hås01, GLST98] have length nearly n^{10^6}, still a polynomial!.

Work with regard to optimizing the length of the PCP was initiated by Babai, Levin, Fortnow and Szegedy [BFLS91]. Polishchuk and Spielman [PS94], following the work of Babai *et al.* constructed PCPs for **NP** with a blowup factor of at most n^{ε} and checkable by $O(1/\varepsilon)$ queries for every $\varepsilon > 0$. By blowup, we refer to the ratio of the length of the new PCP constructed to that of the original **NP** proof. Thus, the PCPs of [PS94] were of length $n^{1+\varepsilon}$ and checkable with $O(1/\varepsilon)$ queries for every $\varepsilon > 0$. Later, Harsha and Sudan [HS00] constructed PCPs with a slightly super-quadratic ($n^{2+\varepsilon}$ for every $\varepsilon > 0$) blowup, checkable with just 17 queries. More recently, Ben-Sasson *et al.* [BSVW03], building on the work of Goldreich and Sudan [GS02] constructed PCPs with a blowup factor of just $2^{\sqrt{\log n}}$, checkable with a constant number of queries. In this thesis, we continue this research direction and construct even shorter PCPs. In informal terms, we construct PCPs involving a blowup fact of at most $2^{(\log n)^{\varepsilon}}$ checkable with $O(1/\varepsilon)$ queries for every $\varepsilon > 0$. Furthermore, if we are willing to slightly increase the query complexity to a super-constant $o(\log \log n)$, then we show how to reduce the blowup factor to a mere $\mathrm{quasipoly}\text{-}\log n$ factor (For an exact statement of the results, refer to Section 1.3.3).

[1] We stress that in all the above works as well as in the current work, the PCP can be computed in polynomial-time from the original **NP** proof.

1.3.2 *Why Short PCPs?*

Why are we interested in short PCPs? Why is the length of PCP an interesting parameter? To answer these questions, we first feel that we should not limit ourselves to viewing PCPs as mere tools to prove inapproximability results. In our view, the significance of PCPs extends far beyond their applicability to deriving inapproximability results. The mere fact that **NP**-proofs can be transformed into a format that supports super-fast probabilistic verification is remarkable. From this perspective, the question of how much redundancy is introduced by such a transformation is a fundamental one. PCPs provide a very robust means of rewriting the original **NP** proof in the following sense. Suppose a purported **NP** proof for a false assertion was erroneous at just a single bit location. Yet, the PCP style of rewriting this (erroneous) proof scatters this error all over the place. Thus, *PCPs are to computation and proof-verification what error-correcting codes are to communication.* From this viewpoint, the blowup of the proof size is the natural analogue of the inverse of the rate of the code and is thus a very natural parameter to study. PCPs, as the name suggests, provide a mechanism for efficient automated proof-checking. One of the main reasons, why PCPs are not being used today in practice for automated proof-checking is that the blowup of the proof-size involved in all present constructions of PCPs makes it infeasible to do so. We do not claim that our short PCPs make this feasible, however they are a significant step in that direction.

Furthermore, PCPs have been used not only to derive inapproximability results but also for obtaining positive results (e.g., CS-proofs [Kil92, Mic00] and their applications [Bar01, CGH98]), and the length of the PCP affects the complexity of those applications. PCP constructions are also used to construct certain codes (locally testable codes and "relaxed locally decodable codes"), the proof-size of the PCP is directly related to the rate of these codes.

In any case, the length of PCPs is also relevant to inapproximability results; specifically, it affects their *tightness with respect to the running time* (as noted in [Sze99]). For example, suppose the optimization problem (exact) SAT has complexity $2^{\Omega(n)}$. The original PCP theorem [AS98, ALM$^+$98] implies that the approximation version of SAT, namely MaxSAT requires time 2^{n^α}, for some (small) $\alpha > 0$. The work of Polishchuk and Spielman [PS94] makes α arbitrarily close to 1, whereas the results of [GS02, BSVW03] further improve the lower-bound to $2^{n^{1-o(1)}}$. Our results reduce the $o(1)$ term.[2]

1.3.3 *Main Results*

How short can a PCP be? The answer may depend on the number of bits we are willing to read in order to reject false assertions (say) with probability at least $1/2$. To place our results in perspective, we begin by citing the PCP constructions of Polishchuk and Spielman [PS94]. It is implicit in their

[2] A caveat: it is currently not known whether these improved lower-bounds can be achieved simultaneously with optimal approximation ratios, but the hope is that this can eventually be done.

work [PS94] that, for $\mathbf{NP}$ proofs verifiable in time n, if we are willing to read $n^{0.01}$ bits then the length of the new PCP may be $\widetilde{O}(n)$. That is, stretching the $\mathbf{NP}$ proof by only a poly-logarithmic amount, allows to dramatically reduce the number of bits read (from n to $n^{0.01}$). More precisely:[3]

Theorem 1.3.1 (implicit in [PS94]) *Any $\mathbf{NP}$ proof verifiable in time n can be encoded by a PCP of length $\mathrm{poly}(\log n) \cdot n$ which can be probabilistically verified by probing at most $n^{o(1)}$ bit locations. In fact, for any value of a parameter $m \leq \log n$, there is a PCP having randomness complexity $(1 - m^{-1}) \cdot \log_2 n + O(\log\log n) + O(m \log m)$ and query complexity $\mathrm{poly}(\log n) \cdot n^{1/m}$.*

Recall that the proof length of a PCP is at most $2^r \cdot q$, where r is the randomness complexity and q is the query complexity of the PCP. Thus, the first part of the above theorem follows by setting $m = \log\log n / \log\log\log n$ in the second part.

Our first main result shows that the query complexity can be reduced dramatically if we are willing to increase the length of the proof slightly. First, with a quasi-poly-logarithmic stretch, the query complexity can be made double-logarithmic:

Main Theorem 1.3.2 *Any $\mathbf{NP}$ proof verifiable in time n can be encoded by a PCP of length $\exp(o(\log\log n)^2) \cdot n$ which can be probabilistically verified by probing at most $o(\log\log n)$ bit-locations. In fact, it has a PCP having randomness complexity $\log_2 n + O\big((\log\log n)^2 / \log\log\log n\big)$ and query complexity $O(\log\log n / \log\log\log n)$.*

We mention that the only prior work claiming query complexity below $\exp(\sqrt{\log n})$ (cf. [GS02, BSVW03]) required stretching the $\mathbf{NP}$ proof by at least a $\exp(\sqrt{\log n})$ factor. With approximately such a stretch factor, these works actually achieved constant query complexity (cf. [GS02, BSVW03]). Thus, Theorem 1.3.2 represents a vast improvement in the query complexity of PCPs that use very short proofs (i.e., in the range between $\exp(o(\log\log n)^2) \cdot n$ and $\exp(\sqrt{\log n}) \cdot n$). The second main result on the other hand considers PCPs that allow probabilistic verification by a *constant* number of queries. We reduce the best known stretch factor from $\exp(\log^{0.5+\varepsilon} n)$ (established in [GS02, BSVW03]) to $\exp(\log^{\varepsilon} n)$, for any $\varepsilon > 0$. That is:

Main Theorem 1.3.3 *For every constant $\varepsilon > 0$, any $\mathbf{NP}$ proof verifiable in time n can be encoded by a PCP of length $\exp(\log^{\varepsilon} n) \cdot n$ which can be probabilistically verified by probing at most a constant number of bit-locations. In fact, it has a PCP having randomness complexity $\log_2 n + \log^{\varepsilon} n$ and query complexity $O(1/\varepsilon)$.*

It may indeed be the case that the trade-off (between length blow-up factors and query complexity) offered by Theorems 1.3.1–1.3.3 merely reflects our (incomplete) state of knowledge. In particular, we wonder whether assertions in $\mathbf{NP}$ can be probabilistically verified by a PCP having proof-length $n \cdot \mathrm{poly}(\log n)$ and constant query complexity.

[3]All logarithms in this work are to based 2, but in some places we choose to emphasize this fact by using the notation $\log_2$ rather than $\log$.

1.3.4 New notions and main techniques

A natural approach to reducing the query complexity in Theorem 1.3.1 is via the "proof composition" paradigm of [AS98]. However, that PCP, as constructed in [PS94], does not seem amenable to composition (when the parameter m is non-constant). Thus, we begin by giving a new PCP construction whose parameters match those in Theorem 1.3.1, but is suitable for composition. As we will see, we cannot afford the standard proof composition techniques, and thus also introduce a new, more efficient composition paradigm.

Our new "proof composition" method refers to two new notions: the notion of a *PCP of proximity* and the notion of a *robust PCP*. Our method is related to the method discovered independently by Dinur and Reingold [DR04]. (There are significant differences between the two methods; as explained in Section 2.2.1.)

PCPs of Proximity. Recall that a standard PCP verifier tests the truth of an assertion by reading the assertion in its entirety and "barely reading" the proof. That is, the PCP verifier is given explicit access to the assertion, but only oracle access to the string that supposedly is a PCP. The verifier checks the truth of the assertion by probabilistically probing this oracle and the query complexity is precisely the maximum number of such oracle accesses. In contrast, in a PCP of proximity(PCPP), the verifier is given oracle access to not only the PCP but also the assertion itself. In other words, a PCPP verifier tests the truth of an assertion by "barely reading" both the assertion and its proof. Needless to say, such a constrained verifier cannot distinguish true assertions from false ones, but it is not required to do so. It is only expected to distinguish between assertions that are true and those which are far from being true (i.e., far in Hamming distance from any true assertion). Thus it only needs to check if the input assertion is close to being true, hence the name – PCP of proximity. However, *its queries to the input assertion oracle are also counted in its query complexity*. As always, we are interested in verifiers with low query complexity. Thus while the query complexity of PCPPs is evaluated more stringently, they are required to do less (i.e., only distingush true assertions from those that are far from true). PCPs of proximity are related to holographic proofs [BFLS91], "PCP spot-checkers" [EKR99] and "Assignment-Testers" [DR04]; see further discussion in Section 2.2.1.

Robust PCPs. Robust PCPs are standard PCPs with a strenghtened soundness guarantee. To discuss robust PCPs, let us recall the soundness guarantee of standard (non-adaptive) PCPs. The standard PCP verifier based on its input assertion and internal random coins determines a set of bit-locations to probe the PCP in and a boolean predicate (on these bit-locations) based on which it accepts/rejects the PCP. The completeness of the verifier states that if the assertion is true, then there is one PCP that causes the boolean predicate to always accept while the soundess guarantees that if the assertion is false, then the boolean predicate rejects with high probability. In other words, for most random coins, the actual bits in the PCP probed by the verifier do not satisfy the boolean predicate. In a robust PCP, we strengthen this latter condition. We require that the bits read by the verifier not only fail to satisfy the boolean predicate, but are in fact significantly *far* from satisfying the boolean predicate. In other words, with high probability, a significant fraction of the bits read by the verifier need to be flipped in order to satisfy the boolean predicate. We call this strenghthened soundness guarantee, robust soundess.

Proof Composition. Our key observation is that "proof composition" works very smoothly when we compose an outer "robust PCP" with an inner "PCP of proximity" instead of standard PCPs. We stress that the flexibility in composing robust PCPs of proximity plays an important role in our ability to derive quantitatively stronger results regarding PCPs. We believe that robust PCPs of proximity may play a similar role in other quantitative studies of PCPs.

1.3.5 Applications to coding problems

The flexibility of PCPs of proximity makes them relatively easy to use towards obtaining results regarding locally testable and decodable codes. In particular, using a suitable PCP of proximity, we obtain an improvement in the rate of locally testable codes (improving over the results of [GS02, BSVW03]). Loosely speaking, a codeword test (for a code C) is a randomized oracle machine that is given oracle access to a string. The tester may query the oracle at a constant number of bit-locations and is required to (always) accept every codeword and reject with (relatively) high probability every string that is "far" from the code. The locally testable codes of [GS02, BSVW03] used codewords of length $\exp(\log^{0.5+\varepsilon} k)\cdot k$ in order to encode k bits of information, for any constant $\varepsilon > 0$. Here we reduce the length of the codewords to $\exp(\log^{\varepsilon} k) \cdot k$. That is:

Theorem 1.3.4 (loosely stated, see Section 12 for details): *For every constant $\varepsilon > 0$, there exists locally testable codes that use codewords of length $\exp(\log^{\varepsilon} k) \cdot k$ in order to encode k bits of information.*

We also introduce a relaxed notion of locally decodable codes, and show how to construct such codes using any PCP of proximity (and ours in particular). Loosely speaking, a code is said to be locally decodable if whenever relatively few location are corrupted, the decoder is able to recover each information-bit, with high probability, based on a constant number of queries to the (corrupted) codeword. This notion was formally defined by Katz and Trevisan [KT00] and the best known locally decodable code has codeword of length that is sub-exponential in the number of information bits. We relax the definition of locally decodable codes by requiring that, whenever few location are corrupted, the decoder should be able to recover most of the individual information-bits (based on few queries) and for the rest of the locations, the decoder may output either the right message bit or a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but is allowed to say "`don't know`" on a few bit-locations. We show that this relaxed notion of local decodability can be supported by codes that have codewords of length that is almost-linear in the number of information bits. That is:

Theorem 1.3.5 (loosely stated, see Section 13 for details): *For every $\varepsilon > 0$, there exists relaxed locally decodable codes that use codewords of length $k^{1+\varepsilon}$ in order to encode k bits of information.*

1.4 Structure of this Thesis

Unfortunately, this thesis is not written in a probabilisitically checkable format that would enable the reader to read just a couple of pages and be assured of the verity of the results. However, we hope that the following outline of the thesis will help the reader.

We start by providing a definitional treatment of PCPs of proximity in Chapter 2 – we recall the standard definition of PCPs in Chapter 2 and then proceed to define the two new variants of PCPs - PCPs of proximity and robust-PCPs. The basic definitions as well as some observations and useful transformations are presented in Section 2.

In Chapter 3, we present the composition theorem for robust PCPs of proximity. We also make several remarks on proof-composition performed in earlier PCP constructions. We explain how our two new PCP variants make composition simpler and smoother than in earlier PCP constructions and help in constructing shorter PCPs.

For starters, we construct a polynomial sized constant query PCP of proximity based on the Hadamard code in Chapter 4. This PCP of proximity though extremely efficient in terms of query complexity performs poorly in terms of proof size (it has exponentially long proofs!). This PCP of proximity is obtained by modifying the inner verifier from Arora *et al.* [ALM$^+$98]. As in [ALM$^+$98], this PCP of proximity will be used in the innermost level of composition.

As mentioned earlier, our proof-composition not only improves the randomness complexity but is also significantly simpler than earlier compositions. This composition can serve as substitute for the original composition in [ALM$^+$98] to yield a simpler proof of the PCP Theorem. However, obtaining our improved parameters with respect to length requires a lot of technical work. Thus, unfortunately we do not have a single construction that both yields a shorter proof of the PCP Theorem and a short PCP simultaneously. So, we first give a simple construction of a robust PCP of proximity (which does not perform great with respect to proof-size, only polynomially long proofs) in Part I and then compose with itself to obtain a simpler proof of the PCP Theorem. In part II, we considerably improve on the construction from Part I to obtain our short PCPs. The main technical meat of this thesis lies in the construction of this short PCP (cf. Part II).

Part I (Proof of the PCP Theorem): In Chapter 5, we use the "new" composition theorem proved in Chapter 3 to give a simpler proof of the PCP Theorem. For this purpose, we modify the (pre-composition) verifiers of [ALM$^+$98] to both test proximity and have robust soundness.

The first 4 chapters along with Part I give a proof of the PCP Theorem and can be read independently of the rest of the thesis. The remaining chapters are devoted to improving the parameters of the PCPs constructed. Part I gives a simpler and almost self-contained proof of the PCP Theorem (modulo the proof of the low-degree test (Appendix A, which we cite from [ALM$^+$98, BSVW03]). Hopefully, this exposition will make the PCP Theorem accessible to a wider audience.

Part II (Short PCPs): The five chapters (Chapter 6–10) in this part contain the technical meat of this thesis and are devoted to proving the main theorems (Theorem 1.3.2 and 1.3.3). We start this part by providing an overview of the construction. The construction itself is presented in Chapters 7–9. We start by presenting a (highly efficient) ordinary PCP (establishing Theorem 1.3.1), which lends itself to the subsequent modifications. In Chapter 8, we augment this PCP with a test of proximity, deriving an analogous PCP of proximity. In Chapter 9 we present a robust version of the PCP of proximity derived in the previous sections. In Chapter 10, we show how to derive Theorems 1.3.2 and 1.3.3, by composing this Robust PCP of proximity with itself multiple times. Specifically, $o(\log\log n)$ compositions are used to derive Theorem 1.3.2 and $1/\varepsilon$ compositions are used to derive Theorem 1.3.3. robust PCP of proximity of the required complexities.

Part III (Applications to Coding): In Chapter III, we present certain application of our PCP construction to Coding Theory. We present improved constructions of locally testable codes. We also introduce a relaxed notion of locally decodable codes, and show how to construct such codes using any PCP of proximity (and ours in particular).

Appendix A: Low-Degree Test: The low-degree test is a crucial ingredient in most construction of PCPs including ours (with the significant exception of [DR04]). The low-degree test is procedure to test if an arbitrary function is close to some polynomial of low degree. In Appendix A, we formally define this problem and quote the required results in this area from [ALM$^+$98, BSVW03].

CHAPTER 2

PCPs and variants

In this chapter, we recall the standard definition of PCPs (in Section 2.1) and then present the formal definitions of our two variants of PCPs: PCPs of proximity and Robust PCPs (in Sections 2.2 and 2.3, respectively). We discuss other related notions (holographic proofs, property testing, assignment testers etc.) that have appeared in literature in Section 2.2.1. Finally, we conclude making several observations on the two newly defined objects: PCPs of proximity and Robust PCPs (in Section 2.4).

First for some notation.

Notation: We will extensively refer to the *relative* distance between strings/sequences over some alphabet Σ: For $u, v \in \Sigma^\ell$, we denote by $\Delta(u, v)$ the fraction of locations on which u and v differ (i.e., $\Delta(u, v) \triangleq |\{i : u_i \neq v_i\}|/\ell$, where $u = u_1 \cdots u_\ell \in \Sigma^\ell$ and $v = v_1 \cdots v_\ell \in \Sigma^\ell$). We say that u is *δ-close* to v (resp., *δ-far* from v) if $\Delta(u, v) \leq \delta$ (resp., $\Delta(u, v) > \delta$). The relative distance of a string to a set of strings is defined in the natural manner; that is, $\Delta(u, S) \triangleq \min_{v \in S}\{\Delta(u, v)\}$. Occasionally, we will refer to the absolute Hamming distance, which we will denote by $\overline{\Delta}(u, v) \triangleq |\{i : u_i \neq v_i\}|$.

In this thesis, we will construct PCP verifiers for several languages based on circuits. Except when otherwise noted, all *circuits* in this thesis have fan-in 2 and fan-out 2, and we allow arbitrary unary and binary Boolean operations as internal gates. The *size* of a circuit is the number of gates. We will refer to the following languages associated with circuits: the $\mathbf{P}$-complete language CIRCUIT VALUE, defined as CKTVAL $= \{(C, w) : C(w) = 1\}$; and the $\mathbf{NP}$-complete CIRCUIT SATISFIABILITY, defined as CKTSAT $= \{C : \exists w C(w) = 1\}$; and the also $\mathbf{NP}$-complete NONDETERMINISTIC CIRCUIT VALUE, defined as NCKTVAL $= \{C : (C, w) : \exists z C(w, z) = 1\}$. (In the latter, we assume that the partition of the variables of C into w-variables and z-variables is explicit in the encoding of C.)

2.1 *Standard PCPs*

We begin by recalling the formalism of a PCP verifier. Throughout this work, we restrict our attention to *nonadaptive* verifiers, both for simplicity and because one of our variants (namely robust PCPs) only makes sense for nonadaptive verifiers.

Definition 2.1.1 (PCP verifiers)

- *A* verifier *is a* probabilistic polynomial-time *algorithm V that, on an input x of length n, tosses $r = r(n)$ random coins R and generates a* sequence *of $q = q(n)$ queries $I = (i_1, \ldots, i_q)$ and a circuit $D : \{0,1\}^q \to \{0,1\}$ of size at most $d(n)$.*

  We think of V as representing a probabilistic oracle machine that queries its oracle π for the positions in I, receives the q answer bits $\pi|_I \triangleq (\pi_{i_1}, \ldots, \pi_{i_q})$, and accepts iff $D(\pi|_I) = 1$.

- *We write $(I, D) \overset{R}{\leftarrow} V(x)$ to denote the queries and circuit generated by V on input x and random coin tosses, and $(I, D) = V(x; R)$ if we wish to specify the coin tosses R.*

- *We call r the* randomness complexity, *q the* query complexity, *and d the* decision complexity *of V.*

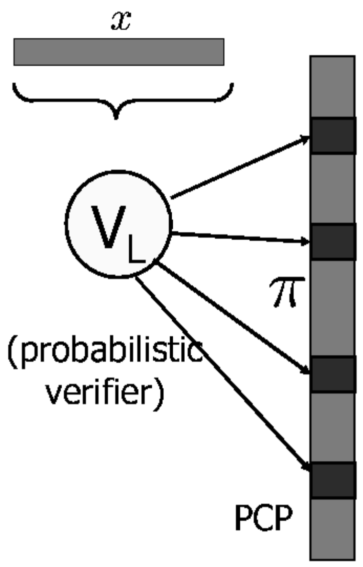

Figure 2-1: PCP Verifier

The standard PCP verifier checks that the string x is in the language L be reading all of x and probabilistically probing the proof π.

For simplicity in these definitions, we treat the parameters r, q, and d above (and other parameters below) as functions of only the input length n. However, at times we may also allow them to depend on other parameters, which should be understood as being given to the verifier together with the input. We now present the standard notion of PCPs, restricted to perfect completeness for simplicity.

Definition 2.1.2 (standard PCPs) *For a function $s : \mathbb{Z}^+ \to [0,1]$, a verifier V is a* probabilistically checkable proof system *for a language L with* soundness error *s if the following two conditions hold for every string x:*

Completeness: *If $x \in L$ then there exists π such that $V(x)$ accepts oracle π with probability 1. Formally,*

$$\exists \pi \qquad \Pr_{(I,D)\overset{R}{\leftarrow} V(x)} [D(\pi|_I) = 1] = 1.$$

Soundness: *If $x \notin L$ then for every oracle π, the verifier $V(x)$ accepts π with probability strictly less than s. Formally,*

$$\forall \pi \qquad \Pr_{(I,D)\overset{R}{\leftarrow} V(x)} [D(\pi|_I) = 1] < s(|x|).$$

If s is not specified, then it is assumed to be a constant in $(0,1)$.

Our main goal in this work is to construct *short* PCPs that use *very few queries*. Recalling that the length of a (nonadaptive) PCP is upper-bounded by $2^{r(n)} \cdot q(n)$, we focus on optimizing the (trade-off between) randomness and query complexities.

We will focus on constructing PCPs for the **NP**-complete problem CIRCUIT SATISFIABILITY, defined as CKTSAT $= \{C : \exists w \; C(w) = 1\}$. Recall that every language in $\mathbf{NTIME}(t(n))$ reduces to CKTSAT in time $O(t(n) \log t(n))$ (cf. [HS66, PF79, Coo88]), and so a nearly linear-sized PCP for CKTSAT implies PCPs for $\mathbf{NTIME}(t(n))$ of size nearly linear in $t(n)$ for every polynomial $t(n)$.

2.2 PCPs of Proximity

We now present a variation of PCPs in which the verifier checks the membership of a string in a language by having oracle access not only to the proof but also to the string whose membership it is checking, i.e., the verifier not only barely reads the proofs but also barely reads the string whose membership it is checking. As expected, such a constrained verifier cannot distinguish between strings which are in the language from ones that are not. This variant is only expected to distinguish between string which are in the language from ones that are far from being in the language (where far is measured in terms of Hamming distance). Thus these new verifiers check that the input is *close* to an element of the language, hence the name – PCPs of Proximity. The advantage of this variant is that they allow for very smooth composition (for more details refer Section 3.1).

For greater generality, we will divide the input into two parts (x, y), giving the verifier x explicitly and y as an oracle, and we only count the verifier's queries to the latter. Thus we consider languages consisting of pairs of strings, which we refer to as a pair language. One pair language to keep in mind is the CIRCUIT VALUE problem: $\textsc{CktVal} = \{(C, w) : C(w) = 1\}$. For a pair language L, we define $L(x) = \{y : (x, y) \in L\}$. For example, $\textsc{CktVal}(C)$ is the set of satisfying assignments to C. It will be useful below to treat the two oracles to which the verifier has access as a single oracle, thus for oracles π^0 and π^1, we define the concatenated oracle $\pi = \pi^0 \circ \pi^1$ as $\pi_{b,i} = \pi^b_i$.

Definition 2.2.1 (PCPs of proximity (PCPPs)) *For functions $s, \delta : \mathbb{Z}^+ \to [0, 1]$, a verifier V is a* probabilistically checkable proof of proximity *(PCPP) system for a pair language L with* proximity parameter *δ and* soundness error *s if the following two conditions hold for every pair of strings (x, y):*

Completeness: *If $(x, y) \in L$, then there exists π such that $V(x)$ accepts oracle $y \circ \pi$ with probability 1. Formally,*

$$\exists\pi \qquad \Pr_{(I,D)\overset{R}{\leftarrow}V(x)}[D((y \circ \pi)|_I) = 1] = 1.$$

Soundness: *If y is $\delta(|x|)$-far from $L(x)$, then for every π, the verifier $V(x)$ accepts oracle $y \circ \pi$ with probability strictly less than $s(|x|)$. Formally,*

$$\forall\pi \qquad \Pr_{(I,D)\overset{R}{\leftarrow}V(x)}[D((y \circ \pi)|_I) = 1] < s(|x|).$$

If s and δ are not specified, then both are assumed to be constants in $(0, 1)$.

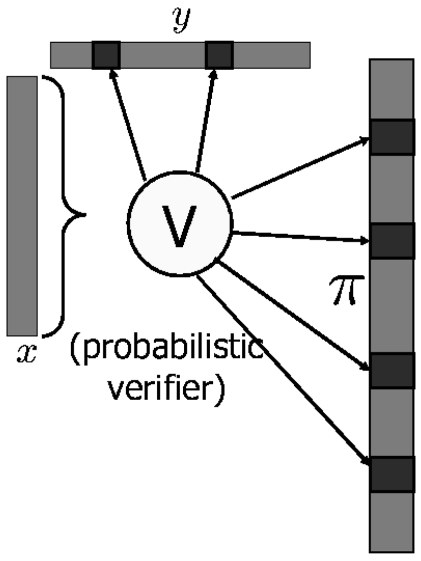

Figure 2-2: PCP of Proximity

The PCPP verifier checks that the pair (x, y) is in the pair-language L by reading all of x and probabilistically probing the string y and the proof π.

Note that the parameters (soundness, randomness, etc.) of a PCPP are measured as a function of the length of x, the explicit portion of the input.

In comparing PCPPs and PCPs, one should note two differences that have conflicting effects. On one hand, the soundness criterion of PCPPs is a relaxation of the soundness of PCPs. Whereas, a PCP is required to reject (with high probability) every input that is not in the language, a PCPP is only required to reject input pairs (x, y) in which the second element (i.e., y) is far from being suitable for the first element (i.e., y is far from $L(x)$). That is, in a PCPP, nothing is required in the case that y is close to $L(x)$ and yet $y \notin L(x)$. On the other hand, the query complexity of a PCPP is measured more stringently, as it accounts also for the queries to the input-part y (on top of the standard queries to the proof π). This should be contrasted with a standard PCP that has free access to all its input, and is only charged for access to an auxiliary proof. To summarize, PCPPs are required to do less (i.e., their performance requirements are more relaxed), but they are charged for more things (i.e., their complexity is evaluated more stringently). Although it may not be a priori clear, the stringent complexity requirement prevails. That is, PCPPs tend to be more difficult to construct than PCPs of the same parameters. For example, while CIRCUIT VALUE has a trivial PCP (since it is in **P**), a PCPP for it implies a PCP for CIRCUIT SATISFIABILITY:

Proposition 2.2.2 *If* CIRCUIT VALUE *has a PCPP, then* CIRCUIT SATISFIABILITY *has a PCP with identical parameters (randomness, query complexity, decision complexity, and soundness).*

An analogous statement holds for any pair language L and the corresponding projection on first element $L_1 \triangleq \{x : \exists y \text{ s.t. } (x, y) \in L\}$; that is, if L has a PCPP then L_1 has a PCP with identical parameters.

Proof: A PCP π that C is satisfiable can be taken to be $w \circ \pi'$, where w is a satisfying assignment to C and π' is a PCPP that $(C, w) \in$ CKTVAL. This proof π can be verified using the PCPP verifier. The key observation is that if $C \notin$ CIRCUIT SATISFIABILITY then there exists no w that is 1-close to CIRCUIT VALUE(C), because the latter set is empty. ■

Note that we only obtain a standard PCP for CIRCUIT SATISFIABILITY, rather than a PCP of proximity. Indeed, CIRCUIT SATISFIABILITY is not a pair language, so it does not even fit syntactically into the definition of a PCPP. However, we can give a PCPP for the closely related (and also **NP**-complete) pair language NONDETERMINISTIC CIRCUIT VALUE. Recall that is the language NCKTVAL $= \{(C, w) : \exists z C(w, z) = 1\}$ (where the variables of C are explicitly partitioned into w-variables and z-variables).

Proposition 2.2.3 *If* CIRCUIT VALUE *has a PCPP with proximity parameter $\delta(n)$, soundness $s(n)$, randomness $r(n)$, query complexity $q(n)$, and decision complexity $d(n)$, then* NONDETERMINISTIC CIRCUIT

VALUE *has a PCPP with proximity parameter* $2\delta(4n)$*, soundness* $s(4n)$*, randomness* $r(4n)$*, query complexity* $q(4n)$*, and decision complexity* $d(4n)$*.*

Proof: Given a circuit $C(\cdot,\cdot)$ of size n whose variables are partitioned into one group of size k and another of size ℓ, we transform it into a new circuit $C'(\cdot,\cdot)$ of size $n' = 4n$ in which the first group has size $k' \geq \ell$ and the second group has size ℓ. Specifically, we set $t = \lceil \ell/k \rceil$ and $k' = t \cdot k$, and define $C'(x', y)$ to be a circuit that checks whether $x' = x^t$ for some x such that $C(x, y) = 1$. It can be verified that this can be done in size $n + 3tk \leq 4n$ (over the full binary basis). In addition, if w is δ-far from being extendable to a satisfying assignment of C, then w^t is δ-far from being extendable to a satisfying assignment of C'.

Now, the NCKTVAL-verifier, on explicit input C and input oracle $w \in \{0,1\}^k$, will construct C' as above and expect a proof oracle of the form $z \circ \pi$, where $z \in \{0,1\}^m$ and π is a PCPP that $w^t \circ z$ satisfies C' as constructed above. That is, the NCKTVAL-verifier will simulate the CKTVAL-verifier on explicit input C', input oracle $w^t \circ z$ (which can easily be simulated given oracle access to w and z), and proof oracle π. Completeness can be verified by inspection. For soundness, suppose that w is 2δ-far from being extendable to a satisfying assignment of C. Then w^t is 2δ-far from being extendable to a satisfying assignment of C', which implies that, for any z, $w^t \circ z$ is δ-far from satisfying C'. Thus, by the soundness of the CKTVAL-verifier, the acceptance probability is at most $s(n') = s(4n)$, for any proof oracle π. ■

Various observations regarding PCPs of proximity are presented in Section 2.4.

2.2.1 Related Work

The notion of a PCP of proximity is related (and equivalent to) to notions that have appeared in the literature. PCP of proximity are implicit in the low-degree testers that utilize auxiliary oracles (e.g., an oracle that provides the polynomial representing the value of the function restricted to a queried line); cf. [AS98, ALM^{+}98] to test if a given function (given as a table of values) is close to a low-degree polynomial.

Relation to holographic proofs. Firstly, the notion of a PCP of proximity generalizes the notion of holographic proofs set forward by Babai, Fortnow, Levin, and Szegedy [BFLS91]. In both cases, the verifier is given oracle access to the input, and we count its probes to the input in its query complexity. The key issue is that holographic proofs refer to inputs that are presented in an error-correcting format (e.g., one aims to verify that a graph that is *represented by an error-correcting encoding of its adjacency matrix* (or incidence list) is 3-colorable). In contrast, a PCP of proximity refers to inputs that are presented in any format but makes assertions only about their proximity to acceptable inputs (e.g., one is interested in whether a graph, represented by its adjacency matrix (or incidence list), *is 3-colorable or is far from being 3-colorable*).

Relation to Assignment Testers [DR04] In addition, PCPPs play an important role also in the work of Dinur and Reingold [DR04]; again. They define an identical object which they call "Assignment Testers", objects that check if a implicitly given assignment is close to a satisfying assignment of a explicitly given circuit. Both our use and their use of PCPPs is for facilitating "proof composition" (see Section 3.4 for more details). Our definition was motivated by our goal to construct short PCPs while their motivation was to "combinatorialize" the proof of the PCP Theorem. Surprisingly both these differing goals led to the same object and offers further evidence that this is a very natural object to study.

2.2.2 *Relation to Property Testing*

Following Ergün *et al.* [EKR99], we view PCPs of proximity as an extension of property testing [RS96, GGR98]. Loosely speaking, a property tester is given oracle access to an input and is required to distinguish the case in which the input has the property from the case in which it is far (say in Hamming distance) from any input having the property. Typically, the interest is in testers that query their input on few bit-locations (or at the very least on a sub-linear number of such locations).

In a PCP of proximity such a tester (now called a verifier) is also given oracle access to an alleged proof. The requirements from a PCPP for a pair language L refer only to its performance on the ("gap") promise problem $\Pi = (\Pi_Y, \Pi_N)$, where $\Pi_Y = L$ and $\Pi_N = \{(x, y) : y \text{ is } \delta\text{-far from } L(x)\}$. That is, this PCPP is only required to (always) accept inputs in Π_Y and reject (with high probability) inputs in Π_N (whereas nothing is required with respect to inputs not in $\Pi_Y \cup \Pi_N$). Such a gap problem corresponds to the notion of approximation in *property testing* [RS96, GGR98].[1] Indeed, property testers are equivalent to PCPP verifiers that have no access to an auxiliary proof π. Thus the relation between property testing and PCPPs is analogous to the relation between $\mathbf{BPP}$ and $\mathbf{NP}$ (or $\mathbf{MA}$). Put differently, while property testing provides a notion of approximation for decision procedures, PCP of proximity provides a notion of approximation for (probabilistic) verification procedures. In both cases, approximation means that inputs in the language should be accepted (when accompanied with suitable proofs) while inputs that are far from the language should be rejected (no matter what false proof is provided). For example, the problem of testing Bipartiteness can be cast by the pair language $L = \{(n, G) : \text{the } n\text{-vertex graph } G \text{ is bipartite}\}$, where the first (i.e., explicit) input is only used to specify the length of the second (i.e., non-explicit) input G,

[1]This notion of approximation (of decision problems) should not be confused with the approximation of (search) optimization problems, which is also closely related to PCPs [FGL+96, ALM+98].

to which the tester has oracle access (measured in its query complexity). We comment that the formulation of pair languages allows to capture more general property testing problems where more information about the property (to be tested) itself is specified as part of the input (e.g., by a circuit, as in CKTVAL).

In both property testers and PCPs of proximity, the interest is in testers/verifiers that query their input (and proof oracle) in only a small (preferably constant, and certainly sublinear) number of bit-locations. It turns out that PCPPs are provably stronger than property testers; that is, there are (natural) separations between property testers and PCPs of proximity. (Some of the following examples were pointed out in [EKR99].) In the adjacency matrix model (cf. [GGR98]), Bipartiteness has a PCP of proximity in which the verifier makes only $O(1/\delta)$ queries and rejects any graph that is δ-far from being bipartite with probability at least $2/3$. (The proof-oracle consists of an assignment of vertices to the two parts, and the verifier queries the assignment of the end-points of $O(1/\delta)$ random edges. This construction also generalizes to k-colorability, and in fact any generalized graph partition property (cf. [GGR98]) with an efficient one-sided tester.) In contrast, Bogdanov and Trevisan [BT04] showed that any tester for Bipartiteness that rejects graphs that are δ-far from being bipartite must make $\Omega(\delta^{-3/2})$ queries. More drastic separations are known in in the incidence-lists (bounded-degree) model (of [GR02]): testing Bipartiteness (resp., 3-colorability) of n-vertex graphs has query complexity $\Omega(\sqrt{n})$ [GR02] (resp., $\Omega(n)$ [BOT02]), but again a PCP of proximity will only use $O(1/\delta)$ queries.

Another example comes from the domain of codes. For any good code (or "even" any code of linear distance), there exists a PCP of proximity for the property of being a codeword that makes a constant number of queries.[2] This stands in contrast to the linear lower-bound on the query-complexity of codeword testing for some (good) linear codes, proved by Ben-Sasson *et al.* [BHR03].

Needless to say, there may be interesting cases in which PCPs of proximity do not out-perform property testers.

2.3 *Robust Soundness*

In this section, we present a *strengthening* of the standard PCP soundness condition. Instead of asking that the bits that the verifier reads from the oracle are merely rejected with high probability, we ask that the bits that the verifier reads are *far* from being accepted with high probability. The main motivation for this notion is that, in conjunction with PCPPs, it allows for a very simple composition without the usual costs of "parallelization".

[2]Indeed, this is a special case of our extension of the result of Babai *et al.* [BFLS91], discussed in Section 2.2.1. On the other hand, this result is simpler than the locally testable code mentioned in Section 1.3.5, because here the PCP of proximity is not part of the codeword.

Definition 2.3.1 (robust soundness) *For functions $s, \rho : \mathbb{Z}^+ \to [0,1]$, a PCP verifier V for a language L has* robust-soundness error *s with* robustness parameter *ρ if the following holds for every $x \notin L$: For every oracle π, the bits read by the verifier V are ρ-close to being accepted with probability strictly less than s. Formally,*

$$\forall \pi \quad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [\exists a \text{ s.t. } D(a) = 1 \text{ and } \Delta(a, \pi|_I) \leq \rho] < s(|x|).$$

If s and ρ are not specified, then they are assumed to be constants in $(0,1)$. PCPPs with robust-soundness are defined analogously, with the $\pi|_I$ being replaced by $(y \circ \pi)|_I$.

Note that for PCPs with query complexity q, robust-soundness with any robustness parameter $\rho < 1/q$ is equivalent to standard PCP soundness. However, there can be robust PCPs with large query complexity (e.g. $q = n^{\Omega(1)}$) yet constant robustness, and indeed such robust PCPs will be the main building block for our construction.

Robust PCPs are implicit in the "parallelized PCPs" of [ALM$^+$98]. In fact, robust soundness can be immediately obtained from "parallelized PCPs" (see Proposition 2.4.6). However, we will not construct robust PCPs in this manner. We instead obtain robust soundness by "bundling" the queries of the (non-robust) PCPs into a non-constant number of bundles so that robustness is satisfied at the "bundle-level". Robustness at the bit-level is then obtained by encoding the bundles by a good error-correcting code.

Various observations regarding robust PCPs are presented in Section 2.4. We briefly mention here the relation of robustness to parallelization; specifically, when applied to a robust PCP, the simple query-reduction technique of Fortnow *et al.* [FRS94] performs less poorly than usual (i.e., the resulting soundness is determined by the robustness parameter rather than by the number of queries).

2.4 *Various observations and transformations*

Most of this subsection refers to robust PCPs, but we start with an observation regarding PCPs of proximity.

2.4.1 *Queries vs. proximity*

Intuitively, the query complexity of a PCPP should depend on the proximity parameter δ. The following proposition confirms this intuition.

Proposition 2.4.1 (queries vs. proximity) *Suppose pair-language L has a PCPP with proximity parameter δ, soundness error $1-\varepsilon$, and query complexity q. Suppose further that there exists $(x,y) \in L$ such that $|x| = n$ and $|y| = m$, such that if we let $z \in \{0,1\}^m$ be a random string of relative Hamming distance $\delta' \triangleq \delta'(x)$ from y, we have*

$$\Pr_z[z \textit{ is } \delta\textit{-far from } L(x)] \geq \gamma \triangleq \gamma(x).$$

Then

$$q > \frac{\varepsilon \cdot \gamma}{\delta'}$$

In particular, if $L = \textsc{CktVal}$, then $q \geq \varepsilon/(\delta + O(1/n))$.

The first part of Proposition 2.4.1 does not specify the relation of δ' to δ (although, surely, $\delta' > \delta$ must hold for any $\gamma > 0$, because $\Delta(z, L(x)) \leq \Delta(z,y) = \delta'$). The second part relies on the fact that, for $\textsc{CktVal}$, one may set δ' as low as $\delta + O(1/n)$.

Proof: By completeness, there exists an oracle π such that the PCPP verifier $V(x)$ accepts oracle $y \circ \pi$ with probability 1. Consider $z = y \oplus \eta$, where $\eta \in \{0,1\}^m$ is a uniformly distributed string with relative Hamming weight δ'. If we invoke $V(x)$ with oracle to $z \circ \pi$, then the probability (over the choice of η) that any of the positions read by V has been changed is at most $q \cdot \delta'$. Thus, $V(x)$ rejects oracle $(y \oplus \eta) \circ \pi$ with probability at most $q \cdot \delta'$.

On the other hand, by assumption z is δ-far from $L(x)$ with probability at least γ, in which case $V(x)$ should reject oracle $z \circ \pi$ with probability greater than ε, by the PCPP soundness. Thus $V(x)$ should reject with probability greater than $\gamma \cdot \varepsilon$ (over the choice of z and the coin tosses of V), and we conclude that $q \cdot \delta' > \gamma \cdot \varepsilon$, as desired.

For the application to $\textsc{CktVal}$, let $C : \{0,1\}^m \to \{0,1\}$ be a circuit of size n that accepts only the all-zeroes string 0^m, for $m = \Omega(n)$. Then we have $(C, 0^m) \in \textsc{CktVal}$, but for every $\delta' > \delta$ and every string z of relative Hamming weight δ', we see that (C, z) is δ-far from satisfying C. Setting $\gamma = 1$ and δ' such that $\delta' m$ is the least integer greater than δm completes the proof. ∎

2.4.2 *Expected robustness*

Occasionally, we will be interested in a variant of robust-soundness, which refers to distance on average rather than with high probability.

Definition 2.4.2 (expected robustness) *For a function $\rho : \mathbb{Z}^+ \to [0,1]$, a PCP has* expected robustness ρ *if for every $x \notin L$, we have*

$$\forall \pi, \mathrm{E}_{(I,D) \overset{R}{\leftarrow} V(x)}[\Delta(\pi|_I, D^{-1}(1))] > \rho(|x|).$$

Expected robustness for PCPPs is defined analogously.

We now present several generic transformations regarding robustness and soundness. Although we only state them for PCPs, all of these results also hold for PCPPs, with no change in the proximity parameter. The following proposition relates robust-soundness to expected robustness.

Proposition 2.4.3 (robust-soundness vs. expected robustness) *If a PCP has robust-soundness error $1 - \varepsilon$ with robustness ρ, then it has expected robustness $\varepsilon \cdot \rho$. On the other hand, if a PCP has expected robustness ρ, then for every $\varepsilon \leq \rho$, it has robust-soundness error $1 - \varepsilon$ with robustness parameter $\rho - \varepsilon$.*

Expected robustness can easily be amplified to standard robustness *with low robust-soundness error*, using any averaging (a.k.a. oblivious) sampler (cf., [Gol97]). Combined with Proposition 2.4.3, we get a (soundness) error reduction for robust PCPs. For example, using the expander-neighborhood sampler of [GW97], we have:

Lemma 2.4.4 (error reduction via expander neighborhoods) *If a language L has a PCP with expected robustness ρ, randomness complexity r, query complexity q, and decision complexity d, then for every two functions $s, \gamma : \mathbb{Z}^+ \to [0,1]$, then L has PCP having*

- *robust-soundness error s with robustness parameter $\rho - \gamma$,*
- *randomness complexity $r + O(\log(1/s) + \log(1/\gamma))$,*
- *query complexity $O(1/(s\gamma^2)) \cdot q$, and*
- *decision complexity $O(1/(s\gamma^2)) \cdot d$*

2.4.3 *Non-Boolean PCPs*

The next few transformations involve non-Boolean PCPs. That is, PCPs where the oracle returns symbols over some larger alphabet $\Sigma = \{0,1\}^a$ rather than bits; we refer to $a = a(n)$ as the answer length of the PCP. (Often non-Boolean PCPs are discussed in the language of multi-prover interactive proofs, but it is simpler for us to work with the PCP formulation.)

Robust-soundness of a non-Boolean PCP is defined in the natural way, using Hamming distance over the alphabet Σ. (In the case of a robust non-Boolean PCPP, we still treat the input oracle as binary.)

The first transformation provides a way of converting non-Boolean PCPs to Boolean PCPs in a way that preserves robust-soundness.

Lemma 2.4.5 (alphabet reduction) *If a language L has a non-Boolean PCP with answer length a, query complexity q, randomness complexity r, decision complexity d, and robust-soundness error s with robustness parameter ρ, then L has a Boolean PCP with query complexity $O(a \cdot q)$, randomness complexity r, decision complexity $d + O(a \cdot q)$, and robust-soundness error s with robustness parameter $\Omega(\rho)$. If, instead of robust-soundness, the non-Boolean PCP has expected robustness ρ, then the Boolean PCP has expected robustness $\Omega(\rho)$.*

The proof uses a good error-correcting code (i.e., constant relative distance and rate). Furthermore, to obtain decision complexity $d + O(a \cdot q)$ we should use a code having linear-size circuits for encoding (cf. [Spi96]). Using more classical codes would only give decision complexity $d + \widetilde{O}(a \cdot q)$, which is actually sufficient for our purposes.

Proof: This transformation is analogous to converting non-Boolean error-correcting codes to Boolean ones via "code concatenation". Let V be the given non-Boolean PCP verifier, with answer length a. Let $\mathrm{ECC} : \{0,1\}^a \to \{0,1\}^b$ for $b = O(a)$ a binary error-correcting code of constant relative minimum distance, which can be computed by an explicit circuit of size $O(a)$. We will augment the original oracle π having a-bit entries with an additional oracle τ having b-bit entries, where τ_i is supposed to be $\mathrm{ECC}(\pi_i)$. (We note that including the original oracle simplifies the argument as well as frees us from assuming a noiseless decoding algorithm.)

Our new verifier V'(x), on oracle access to $\pi \circ \tau$, will simulate $V(x)$, and for each query i made by V, will query the a bits in π_i and the b bits in τ_i, for a total of $q \cdot (a+b)$ binary queries. That is, if V queries positions $I = (i_1, \ldots, i_q)$, V' will query positions $I' = ((0,i_1), \ldots, (0,i_q), (1,i_1), \ldots, (1,i_q))$. If V outputs a decision circuit $D : (\{0,1\}^a)^q \to \{0,1\}$, V' will output the circuit $D' : (\{0,1\}^a)^q \times (\{0,1\}^b)^q \to \{0,1\}$ defined by

$$D'(x_1, \ldots, x_q, y_1, \ldots, y_q) = D(x_1, \ldots, x_q) \wedge C(x_1, \ldots, x_q, y_1, \ldots, y_q),$$

where

$$C(x_1, \ldots, x_q, y_1, \ldots, y_q) = \bigwedge_{i=1}^{q} (y_i = \mathrm{ECC}(x_i)).$$

Since ECC can be evaluated by a circuit of size $O(a)$, we see that $|D'| = |D| + O(a \cdot q)$, as desired.

For completeness of V', we note that any accepting oracle π for V can be augmented to an accepting oracle for V' by setting $\tau_i = \mathrm{ECC}(\pi_i)$ for all i. For soundness of V', suppose $x \notin L$ and let

(π, τ) be any pair of oracles. Define a "decoded" oracle $\hat{\pi}$ by setting $\hat{\pi}_i$ to be the string $x \in \{0,1\}^a$ which minimizes the distance between $\mathrm{ECC}(x)$ and τ_i. We will relate the robustness of V on oracle $\hat{\pi}$ to the robustness of V' on oracles π and τ. Specifically, let $\beta > 0$ be a constant such that the (absolute) minimum distance of ECC is greater than $2\beta \cdot (a+b)$. Then we will show that for every sequence R of coin tosses and for every $\alpha > 0$, if the bits read by $V'(x;R)$ from $\pi \circ \tau$ are $\alpha\beta$-close to being accepted, then the bits read by V from $\hat{\pi}$ are α-close to being accepted. Thus, both robustness parameters (standard and expected) decrease by at most a factor of β.

Consider any sequence R of coin tosses, let $(I, D) = V(x;R)$, and write $I = (i_1, \ldots, i_q)$. Suppose that $(\pi_{i_1}, \ldots, \pi_{i_q}, \tau_{i_1}, \ldots, \tau_{i_q})$ is $\alpha\beta$-close to some $(\pi'_{i_1}, \ldots, \pi'_{i_q}, \tau'_{i_1}, \ldots, \tau'_{i_q})$ that satisfies $D' = D \wedge C$. Then, for at least a $1-\alpha$ fraction of $j \in [q]$, the pair (π_{i_j}, τ_{i_j}) is β-close to $(\pi'_{i_j}, \tau'_{i_j}) = (\pi'_{i_j}, \mathrm{ECC}(\pi'_{i_j}))$. For such j, the choice of β implies that $\mathrm{ECC}(\pi'_{i_j})$ is the closest codeword to τ_{i_j} and hence $\hat{\pi}_{i_j} = \pi'_{i_j}$. Since the π''s satisfy D, we conclude that $\hat{\pi}$'s are α-close to satisfying D, as desired. ∎

The usual "parallelization" paradigm of PCPs [LS91, ALM$^+$98] converts a Boolean PCP with many queries into a non-Boolean PCP with a constant number of queries, where this is typically the first step in PCP composition. As mentioned in the introduction, we cannot afford parallelization, and robust-soundness will be our substitute. Nevertheless, there is a close (but not close enough for us) connection between parallelized PCPs and PCPs with robust-soundness:

Proposition 2.4.6 (parallelization vs. robustness)

1. *If a language L has a non-Boolean PCP with answer length a, query complexity q, randomness complexity r, decision complexity d, and soundness error s, then L has a* (Boolean) *PCP with query complexity $O(a \cdot q)$, randomness complexity r, decision complexity $d + O(a \cdot q)$, and robust-soundness error s with robustness parameter $\rho = \Omega(1/q)$.*

2. *If a language L has a* (Boolean) *PCP with query complexity q, randomness complexity r, decision complexity d, and expected robustness ρ, then L has a 2-query non-Boolean PCP with answer length q, randomness complexity $r + \log q$, decision complexity $d + O(1)$, and soundness error $1 - \rho$.*

Thus, for constant soundness and constant robustness parameter, q-query robust (Boolean) PCPs are essentially equivalent to constant-query non-Boolean PCPs with answer length $\Theta(q)$. However, note that in passing from robust-soundness to a 2-query non-Boolean PCP, the randomness complexity increases by $\log q$. It is precisely this cost that we cannot afford, and hence we work with robust-soundness in the rest of the thesis.

Proof: For Part 1, note that any non-Boolean PCP with query complexity q and soundness error s has robust-soundness error s for any robustness parameter $\rho < 1/q$. Thus, the claim follows from Lemma 2.4.5.

Turning to Part 2, let V be a robust PCP verifier for L with the stated parameters. We use the usual query-reduction technique for PCPs [FRS94], and observe that when applied to a robust PCP, the detection probability (i.e., one minus the soundness error) does not deteriorate by a factor of q as usual. Instead, the detection probability of the resulting 2-query (non-Boolean) PCP equals the expected robustness of V.[3] Specifically, the 2-query non-Boolean PCP verifier V' is defined as follows:

- V' expects two oracles, one Boolean oracle π corresponding to the oracle for V, and a second oracle τ with answer length q, indexed by random strings of V.

- On input x, the verifier V' selects a random string R for V and $j \overset{\text{R}}{\leftarrow} [q]$, and computes $(I, D) = V(x; R)$, where $I = (i_1, \ldots, i_q)$. It sets $I' = (R, i_j)$ (which means that the queries for the values τ_R and π_{i_j}) and $D'(a, b) = [(D(a) = 1) \wedge (a_j = b)]$; that is, it accepts if and only if $[D(\tau_R) = 1] \wedge [(\tau_R)_j = \pi_{i_j}]$.

It can be verified that the probability that V' rejects a false assertion is precisely the expected robustness of V. In particular, suppose that $V'(x)$ accepts the oracle pair (π, τ) with probability p. We may assume, without loss of generality, that $D(\tau_R) = 1$ for any R, where $(\cdot, D) = V(x; R)$. Then, it follows that the expected (relative) distance of $\pi|_I$ from τ_R, where $(I, D) = V(x; R)$ for a random R, equals $1 - p$ (because $1 - p = \Pr_{R,j}[(\tau_R)_j \neq \pi_{i_j}]$, which in turn equals $\mathrm{E}_R[\Delta(\tau_R, \pi|_I)]$). This means that on the average, π is $(1 - p)$-close to assignments that satisfy the corresponding decision circuits. Thus, if $x \notin L$ then $1 - p > \rho$, and $p < 1 - \rho$ follows. ■

2.4.4 *Robustness vs. proximity*

Finally, for PCPPs, we prove that the robustness parameter is upper-bounded by the proximity parameter.

Proposition 2.4.7 (robustness vs. proximity) *Suppose a pair-language L has a PCPP with proximity parameter δ and expected robustness ρ. Suppose further that there exists $(x, y) \in L$ such that $|x| = n$ and $|y| = m$, such that if we let $z \in \{0,1\}^m$ be a random string at relative Hamming distance $\delta' \triangleq \delta'(x)$ from y, we have*

$$\Pr_z[z \text{ is } \delta\text{-far to } L(x)] \geq \gamma \triangleq \gamma(x).$$

[3]It may be more instructive (alas more cumbersome) to discuss what is happening in terms of ordinary robustness. Suppose that V has robust-soundness error $s = 1 - d$ with respect to robustness ρ. The standard analysis ignores the robustness and asserts that the 2-query (non-Boolean) PCP has soundness error $s' = 1 - d'$, where $d' = d/q$. This crude analysis implicitly assumes the trivial bound (i.e., $1/q$) of the robustness parameter. A more refined analysis takes advantage of the actual bound of the robustness parameter, and asserts that the 2-query (non-Boolean) PCP has soundness error $s' = 1 - \rho \cdot d$.

Then

$$\rho \leq \delta'/\gamma.$$

In particular, if $L = \textsc{CktVal}$, *then* $\rho \leq \delta + O(1/n)$.

Proof: The proof is similar to that of Proposition 2.4.1. By completeness, there exists an oracle π such that the PCPP verifier $V(x)$ accepts oracle $y \circ \pi$ with probability 1. If we run $V(x)$ with oracle $z \circ \pi$ instead, then bits read by V have expected distance at most δ' from being accepted, where the expectation is over the choices of z (even when fixing the coins of V).

On the other hand, z is δ-far from $L(x)$ with probability at least γ, and for any such fixed z the bits read by V from $z \circ \pi$ should have expected distance greater than ρ from being accepted (over the coin tosses of V). Thus, the expected distance of $z \circ \pi$ from being accepted is greater than $\gamma \cdot \rho$, where here the expectation is over the choice of z and the coin tosses of V. We conclude that $\delta' > \gamma \cdot \rho$, as desired.

Recall that in the proof of Proposition 2.4.1, we have demonstrated the existence of a pair (C, w) such that any string z at distance $\delta' = \delta + O(1/n)$ from w it holds that w is δ-far from satisfying C. Setting $\gamma = 1$, the second part follows. ∎

CHAPTER 3

Composition of Robust PCPs of Proximity

In this chapter, we show how PCPs (more specifically, robust PCPs of proximity) can be composed with each other to obtain PCPs with improved query complexity. The *proof composition* paradigm to reduce query complexity of PCPs was discovered by Arora and Safra [AS98]. However, proof composition in all earlier constructions of PCPs (since [AS98]) was very involved and depended on the actual semantics of the PCPs being composed (see Section 3.1.1 for more details). Our key observation is that composition works very smoothly when we compose an outer "robust PCP" with an inner "PCP of proximity".

Our method is related to the method discovered independently by Dinur and Reingold [DR04]. (There are significant differences between the two methods; as explained in Section 3.4, where we also discuss the relation to Szegedy's work [Sze99].)

3.1 Why composition?

Why do we need composition? To answer this question, let us recall our original goal - we want to construct nearly-linear sized PCPs which are checkable with a constant number of queries. However, we do not know how to *directly* construct PCPs of even polynomial size which are checkable with a constant number of queries[1]. What we instead know to construct are polynomial sized

[1]The PCPs of [ALM$^+$98] are polynomial sized and checkable with $O(1)$ queries, however they are constructed via proof-composition.

PCPs which are checkable with a poly-logarithmic number of queries (For instance, the holographic proofs of [BFLS91] yield such PCPs). Arora and Safra discovered that these PCPs can be "magically composed" with themselves to yield PCPs also of polynomial size but significantly less query complexity - $\mathrm{poly}\log\log n$ instead of $\mathrm{poly}\log n$.

In the following section, we discuss the main idea behind this "magic composition". We then describe some of the difficulties encountered in earlier uses of proof-composition. We finally show how our composition theorem removes these difficulties and that composition is simpler if we work with robust PCPs of proximity instead of standard PCPs. We also explain how this variant of PCPs – robust PCP of proximity and the composition theorem is essential for our construction of short PCPs.

3.1.1 *Proof Composition in earlier PCP constructions*

Let us start with the polynomial sized PCPs of [BFLS91] which are checkable with a constant number of queries. This PCP works as follows: To verify the truth of an assertion verifiable in time n, the PCP verifier tosses $O(\log n)$ random coins, probes a polynomial sized PCP in $\mathrm{poly}\log n$ bit locations and accepts the proof (i.e., the PCP) based on a boolean verdict that depends on the $\mathrm{poly}\log n$ bit locations (and of course, the assertion and the random coins tossed by the verifier). The main problem with this verifier is that it reads too many bits of the proof ($\mathrm{poly}\log n$) to check if this boolean verdict is satisfied or not. Is it possible to do this by reading fewer bits? On the face of it, this seems impossible since the boolean verdict depends on all the $\mathrm{poly}\log n$ bits. However, this is similar to what a PCP verifier does: a PCP verifier checks the correctnes of a proof without reading all the bits of the proof. So we could potentially use another PCP verifier to check if the boolean verdict is satisfied by the $\mathrm{poly}\log n$ bits without actually reading all the $\mathrm{poly}\log n$ bits. This is the main idea of proof composition as introduced by Arora and Safra [AS98].

In other words, instead of the outer verifier (which we shall call V_{out} for convenience) checking if the boolean verdict is satisfied by the $\mathrm{poly}\log n$ bits, it transfers control to an inner PCP verifier V_{in} which supposedly performs this check with the help of an additional proof (another PCP). However, if this inner verifier V_{in} were a standard PCP verifier, then it would at the very least have to read its entire assertion (which are the $\mathrm{poly}\log n$ bits in this case) to check if it satisfies the boolean verdict. Hence, this does not reduce the query complexity of the composed verifier. For this purpose, the composition of [AS98] required an inner verifier V_{in} which "barely read" the assertion whose truth it was checking. This notion of a verifer that barely reads its assertion was formalised using the holographic proof verifiers of Babai *et al.* [BFLS91], where the assertion was presented in a specific encoded format. This solved the problem of having to deal with verifiers that "barely read" the assertion, but added the following issue instead. The new inner holographic proof verifier V_{in} could verify the assertion only if it is presented in a specified encoded format, however the assertion which it needs to verify (i.e., the $\text{poly}\log n$ bits) is not written according to this encoding. For this purpose, one needs to verify that the assertion presented in the encoded format is indeed the encoding of the assertion in the unencoded format. The semantics of arranging this is complex. In earlier compositions, this was earlier performed by a step called "parallelization". Parallelization involved aggregating the queries of the outer verifier V_{out}, converting the boolean outer PCP verifier (with super-constant query complexity) into a non-Boolean outer PCP verifier (with constant query complexity) and checking if the encoding is correct. This step involved a moderate cost in randomness complexity (and hence in the proof size) of the resultant PCP verifier, however this cost is too expensive for the parameters we seek. To get around this issue, we instead use PCPs of proximity for our inner verifier V_{in}. Recall that PCPs of proximity do not expect their explicit input (i.e., the assertion) to be in any specific encoding, i.e., PCPs of proximity are similar to the holographic proof verifiers of [BFLS91] but for the fact that they do not expect their input to be encoded. As a result of this, the entire issue of checking the consistency of encoding does not arise in this case. This saves the randomness that is required to perform the parallelization step, thus helping us build shorter PCPs.

As seen from the above discussion, PCP verifiers compose with a PCP verifier of proximity syntactically. However, this does not give any meaningful object for the following reason: The soundness of the outer PCP verifier V_{out} guarantees that if the assertion is true, then the $\text{poly}\log n$ bits satisfy the boolean verdict and if the assertion is false then the $\text{poly}\log n$ bits violate the boolean verdict with high probability. Thus, the inner verifier V_{in} needs to distinguish between the two cases where (a) the $\text{poly}\log n$ bits satisfy the boolean verdict and (b) the $\text{poly}\log n$ bits violate the boolean verdict. However, as the inner verifier V_{in} is a PCP verifer of proximity, it can only distinguish between inputs that satisfy the boolean verdict and those that are far from satisfying the input. Hence, though the composition of V_{out} and V_{in} can be performed syntactically, the semantics of the two verifiers do not match. For this purpose, we strengthen the soundness of the outer verifier V_{out} and use a robust PCP verifier instead of a standard PCP verifier for the outer verifier V_{out}. It can now be easily verified that the two verifiers – the outer robust PCP verifier and inner PCP verifier of proximity compose both syntactically and semantically. The exact details of this composition are given in Section 3.2.

The key observation is that "proof composition" works very smoothly when we compose an outer "robust PCP" with an inner "PCP of proximity". We need neither worry about how many queries the outer "robust PCP" makes nor care about what coding the inner "PCP of proximity" uses in its proof oracle (much less apply the same encoding to the outer answers). All that we should make sure is that the lengths of the objects match and that the distance parameter in the robustness condition (of the outer verifier) is at least as big as the distance parameter in the proximity condition (of the inner verifier). We note that all earlier uses of the composition theorem (as indicated above) worked with "parallelized" outer PCP verifiers (i.e., non-Boolean PCP verifiers with constant query complexity). However, as observed in Proposition 2.4.6, parallelized PCPs are essentially equivalent to robust PCPs if we ignore a extra $\log q$ term in the randommness complexity of parallelized PCPs. This is the same improvement in the randomness complexity due to the new composition (robust PCPs with PCPs of proximity) mentioned in end of the earlier paragraph .

3.2 Composition Theorem

As promised, a robust "outer" PCP composes very easily with an "inner" PCPPs. Loosely speaking, we can compose such schemes provided that the decision complexity of the outer verifier matches the input length of the inner verifier, and soundness holds provided that the robustness parameter of the outer verifier upper-bounds the proximity parameter of the inner verifier. Note that composition does not refer to the query complexity of the outer verifier, which is always upper-bounded by its decision complexity.

Theorem 3.2.1 (Composition Theorem) *Suppose that for functions $r_{\mathrm{out}}, r_{\mathrm{in}}, d_{\mathrm{out}}, d_{\mathrm{in}}, q_{\mathrm{in}} : \mathbb{N} \to \mathbb{N}$, and $\varepsilon_{\mathrm{out}}, \varepsilon_{\mathrm{in}}, \rho_{\mathrm{out}}, \delta_{\mathrm{in}} : \mathbb{N} \to [0, 1]$, the following hold:*

- *Language L has a* robust PCP *verifier V_{out} with randomness complexity r_{out}, decision complexity d_{out}, robust-soundness error $1 - \varepsilon_{\mathrm{out}}$, and robustness parameter ρ_{out}.*
- CIRCUIT VALUE *has a* PCPP *verifier V_{in} with randomness complexity r_{in}, query complexity q_{in}, decision complexity d_{in}, proximity parameter δ_{in}, and soundness error $1 - \varepsilon_{\mathrm{in}}$.*
- *$\delta_{\mathrm{in}}(d_{\mathrm{out}}(n)) \leq \rho_{\mathrm{out}}(n)$, for every n.*

Then, L has a (standard) *PCP, denoted V_{comp}, with*

- *randomness complexity $r_{\mathrm{out}}(n) + r_{\mathrm{in}}(d_{\mathrm{out}}(n))$,*
- *query complexity $q_{\mathrm{in}}(d_{\mathrm{out}}(n))$,*
- *decision complexity $d_{\mathrm{in}}(d_{\mathrm{out}}(n))$, and*
- *soundness error $1 - \varepsilon_{\mathrm{out}}(n) \cdot \varepsilon_{\mathrm{in}}(d_{\mathrm{out}}(n))$.*

Furthermore, the computation of V_{comp} (i.e. evaluating $(I, D) \leftarrow V_{\mathrm{comp}}(x; R)$) can be performed by some universal algorithm with black-box access to V_{out} and V_{in}. On inputs of length n, this algorithm runs in time n^c for some universal constant c, with one call to V_{out} on an input of length n and one call to V_{in} on an input of length $d_{\mathrm{out}}(n)$. In addition:

- *If* (instead of being a PCP) *the verifier V_{out} is a PCPP with proximity parameter $\delta_{\text{out}}(n)$ then V_{comp} is a PCPP with proximity parameter $\delta_{\text{out}}(n)$.*

- *If V_{in} has robust-soundness with robustness parameter $\rho_{\text{in}}(n)$, then V_{comp} has robust-soundness with robustness parameter $\rho_{\text{in}}(d_{\text{out}}(n))$.*

Proof: We will use the inner PCPP to verify that the oracle positions selected by the (robust) outer-verifier are close to being accepted by the outer-verifier's decision circuit. Thus, the new proof will consist of a proof for the outer verifier as well as proofs for the inner verifier, where each of the latter corresponds to a possible setting of the outer verifier's coin tosses (and is intended to prove that the bits that should have been read by the outer-verifier satisfy its decision circuit). We will index the positions of the new (combined) oracle by pairs such that (out, i) denotes the i'th position in the part of the oracle that represents the outer-verifier's proof oracle, and (R, j) denotes the j'th position in the R'th auxiliary block (which represents the R-th possible proof oracle (for the inner verifier's), which in turn is associated with the outer-verifier's coins $R \in \{0,1\}^{r_{\text{out}}}$). For notational convenience, we drop the input length n from the notation below; all parameters of V_{out} are with respect to length n and all parameters of V_{in} with respect to length $d_{\text{out}}(n)$. With these conventions, here is the description of the composed verifier, $V_{\text{comp}}(x)$:

1. Choose $R \overset{\text{R}}{\leftarrow} \{0,1\}^{r_{\text{out}}}$.

2. Run $V_{\text{out}}(x; R)$ to obtain $I_{\text{out}} = (i_1, \ldots, i_{q_{\text{out}}})$ and D_{out}.

3. Run $V_{\text{in}}(D_{\text{out}})$ (on random coin tosses) to obtain $I_{\text{in}} = ((b_1, j_1), \ldots, (b_{q_{\text{in}}}, j_{q_{\text{in}}}))$ and D_{in}.

   (Recall that V_{in}, as a PCPP verifier, expects two oracles, an input oracle and a proof oracle, and thus makes queries of the form (b, j), where $b \in \{0,1\}$ indicates which oracle it wishes to query.)

4. For each $\ell = 1, \ldots, q_{\text{in}}$, determine the queries of the composed verifier:

   (a) If $b_\ell = 0$, set $k_\ell = (\text{out}, i_{j_\ell})$; that is, V_{in}'s queries to its input oracle are directed to the corresponding locations in V_{out}'s proof oracle. Recall that the j-th bit in V_{in}'s input oracle is the j-th bit in the input to D_{out}, which in turn is the i_j-th bit in the proof oracle of V_{out}.

   (b) If $b_\ell = 1$, set $k_\ell = (R, j_\ell)$; that is, V_{in}'s queries to its R'th possible proof oracle are directed to the corresponding locations in the auxiliary proof. Recall that the j-th bit in the proof oracle that V_{in} is using to verify the claim referring to the outer-verifier coins R is the j-th bit in the R-th block of the auxiliary proof.

5. Output $I_{\text{comp}} = (k_1, \ldots, k_{q_{\text{in}}})$ and D_{in}.

The claims about V_{comp}'s randomness, query, decision, and computational complexities can be verified by inspection. Thus we proceed to check completeness and soundness.

Suppose that $x \in L$. Then, by completeness of the outer verifier, there exists a proof π_{out} making V_{out} accept with probability 1. In other words, for every $R \in \{0,1\}^{r_{\mathrm{out}}}$, if we set $(I_{\mathrm{out}}, D_{\mathrm{out}}) = V_{\mathrm{out}}(x; R)$, we have $D_{\mathrm{out}}(\pi_{\mathrm{out}}|_{I_{\mathrm{out}}}) = 1$. By completeness of the inner verifier, there exists a proof π_R such that $V_{\mathrm{in}}(D_{\mathrm{out}})$ accepts the oracle $\pi_{\mathrm{out}}|_{I_{\mathrm{out}}} \circ \pi_R$ with probability 1. If we set $\pi(t, \cdot) = \pi_t(\cdot)$ for all $t \in \{\mathrm{out}\} \cup \{0,1\}^{r_{\mathrm{out}}}$, then V_{comp} accepts π with probability 1.

Suppose that $x \notin L$, and let π be any oracle. Define oracles $\pi_t(\cdot) = \pi(t, \cdot)$. By the robust-soundness (of V_{out}), with probability greater than $\varepsilon_{\mathrm{out}}$ over the choices of $R \in \{0,1\}^{r_{\mathrm{out}}}$, if we set $(I_{\mathrm{out}}, D_{\mathrm{out}}) = V_{\mathrm{out}}(x; R)$, then $\pi_{\mathrm{out}}|_{I_{\mathrm{out}}}$ is ρ_{out}-far from satisfying D_{out}. Fixing such an R, by the PCPP-soundness of V_{in} (and $\delta_{\mathrm{in}} \leq \rho_{\mathrm{out}}$), it holds that $V_{\mathrm{in}}(D_{\mathrm{out}})$ rejects the oracle $\pi_{\mathrm{out}}|_{I_{\mathrm{out}}} \circ \pi_R$ (or, actually, any proof oracle augmenting the input oracle $\pi_{\mathrm{out}}|_{I_{\mathrm{out}}}$) with probability greater than $\varepsilon_{\mathrm{in}}$. Therefore, $V_{\mathrm{comp}}(x)$ rejects oracle π with probability at least $\varepsilon_{\mathrm{out}} \cdot \varepsilon_{\mathrm{in}}$.

The additional items follow by similar arguments. If V_{out} is a PCPP verifier, then the input is of the form (x, y), where y is given via oracle access. In this case, throughout the proof above we should replace references to the oracle π_{out} with the oracle $y \circ \pi_{\mathrm{out}}$, and for soundness we should consider the case that y is δ_{out}-far from $L(x)$. If V_{in} has robust-soundness, then at the end of the soundness analysis, we note that not only is $\pi_{\mathrm{out}}|_{I_{\mathrm{out}}} \circ \pi_R$ rejected with probability greater than $\varepsilon_{\mathrm{in}}$ but rather it is ρ_{in}-far from being accepted by V_{in} (and hence also by V_{comp}). ■

Ideas similar to some of ours are implicit in Szegedy [Sze99]. In particular, notions of robustness and proximity are implicit in [Sze99], where a robust PCP of proximity is composed with itself in a way that is similar to our composition theorem. We mention that Szegedy does not seek to obtain PCPs with improved parameters, but rather to suggest a framework for deriving nicer proofs of existing results such as Polishchuk and Spielman [PS94].

3.3 *Building block for composition*

Equipped with the new composition theorem, we now need to construct a randomness-efficient robust PCP of proximity, which will serve as our basic building element, which we then compose with itself to obtain our short PCPs. We precisely do this in Chapters 4–9.

As mentioned in the introduction, the composition theorem can serve as a substitute for the original Composition Theorem in the derivation of the original PCP Theorem [ALM$^+$98][2]. Thus, all the technical work in Part II is not forced by the new notion of robust PCPPs, but rather is aimed at constructing ones which have *nearly linear length*.

[2]We do exactly this in Chapter 5, where we prove the PCP Theorem

The initial PCP. Our new proof of Theorem 1.3.1 modifies the constructions of Polishchuk and Spielman [PS94] and Harsha and Sudan [HS00]. The latter construction was already improved in [GS02, BSVW03] to reduce the length of PCPs to $n \cdot 2^{\tilde{O}(\sqrt{\log n})}$. Our results go further by re-examining the "low-degree test" (query-efficient tests that verify if a given function is close to being a low-degree polynomial) and observing that the small-bias sample sets of [BSVW03] give an even more significant savings on the randomness complexity of low-degree tests than noticed in their work.

For starters, PCP constructions tend to use many (i.e., a super-constant number of) functions and need to test if each is a low-degree polynomial. In prior results, this was performed efficiently by combining the many different functions on, say m variables, into a single new one on $m + 1$ variables, where the extra variable provides an index into the many different old functions. Testing if the new function is of low-degree, implicitly tests all the old functions. Such tricks, which involve introducing a few extra variables, turn out to be too expensive in our context. Furthermore, for similar reasons, we can not use other "parallelization" techniques [FRS94, LS91, ALM^{+}98, GS00, Raz98], which were instrumental to the proof composition technique of [AS98]. Going back to the PCP derived in Theorem 1.3.1, we adapt it for our new composition method by introducing a "bundling" technique that offers a randomness efficient alternative to parallelization.

Indeed, Theorems 1.3.2 and 1.3.3 are proved by first extending Theorem 1.3.1 to provide a robust PCP of proximity of similar complexities, and then applying the new "proof composition" method. We stress that our contribution is in providing a proof of Theorem 1.3.1 that lends itself to a modification that satisfies the robustness property, and in establishing the latter property. In particular, the aforementioned "bundling" is applied in order to establish the robustness property. Some care is also due when deriving Theorem 1.3.2 using a non-constant number of "proof compositions". In particular, Theorem 1.3.2 (resp., Theorem 1.3.3) is derived in a way that guarantees that the query complexity is linear rather than exponential in the number of "proof compositions", where the latter is $o(\log \log n)$ (resp., $1/\varepsilon$).

We stress that the flexibility in composing robust PCPs of proximity plays an important role in our ability to derive quantitatively stronger results regarding PCPs. We believe that robust PCPs of proximity may play a similar role in other quantitative studies of PCPs. We note that the standard PCP Theorem of [AS98, ALM^{+}98] can be easily derived using a much weaker and simpler variant of our basic robust PCP of proximity, and the said construction seems easier than the basic PCPs used in the proof composition of [AS98, ALM^{+}98].

3.4 Relation to Assignment Testers of Dinur and Reingold

As stated earlier, our "proof composition" method is related to the method discovered independently by Dinur and Reingold [DR04]. The use a variant of PCPs called *assignment testers* which are identical to PCPs of proximity. Both methods use the same notion of PCPs of proximity, but while our method refers to the new notion of robustness (i.e., to the robustness of the outer verifier) the method of Dinur and Reingold refers to the number of (non-Boolean) queries (made by the outer verifier). Indeed, the method of Dinur and Reingold uses a (new) parallelization procedure (which reduces the number of queries by a constant factor), whereas we avoid parallelization altogether (but rather use a related "bundling" of queries into a non-constant number of "bundles" such that robustness is satisfied at the bundle-level). The main part of the bundling technique takes place at the level of analysis, without modifying the proof system at all. Specifically, we show that the answers read by the verifier can be partitioned into a non-constant number of (a-priori fixed) "bundles" so that on any no instance, with high probability a constant fraction of the bundles read should be modified to make the verifier accept. We stress that the fact that certain sets of queries (namely those in each bundle) are always made together is a feature that our particular proof system happens to have (or rather it was somewhat massaged to have). Once "robust soundness" is established at the "bundle level," we may just modify the proof system so that the bundles become queries and the answers are placed in (any) good error-correcting format, which implies robustness at the bit level. We stress that we cannot afford the cost of any known parallelization procedure, because at the very least these procedures increase the length of the proof by a factor related to the answer length, which is far too large in the context of Theorem 1.3.1 (which in turn serves as the starting point for all the other results in this work). We comment that the parallelization procedure of [DR04] is combinatorial (albeit inapplicable in our context), whereas our "bundling" relies on the algebraic structure of our proof system.

Relation to Szegedy's work [Sze99]. Some of the ideas presented in the current work are implicit in Szegedy's work [Sze99]. In particular, notions of robustness and proximity are implicit in [Sze99], in which a robust PCP of proximity (attributed to [PS94]) is composed with itself in a way that is similar to our composition theorem. We note that Szegedy does not seek to obtain PCPs with improved parameters, but rather to suggest a framework for deriving nicer proofs of existing results such as [PS94]. Actually, he focuses on proving the main result of [PS94] (i.e., a PCP of nearly linear length and constant number of queries) using as building block a robust PCP of proximity that has length $\widetilde{O}(n)$ and makes $\widetilde{O}(\sqrt{n})$ queries (plus the constant-query PCP of [ALM$^+$98]).

We note that the aforementioned robust PCP of proximity is not presented in [Sze99], but is rather attributed to [PS94]. Indeed, observe that Theorem 1.3.1 above (due to [PS94]) achieves $\widetilde{O}(n)$ length and $\widetilde{O}(\sqrt{n})$ queries when the parameter $m = 2$. Thus, Szegedy's assertion is that this PCP can be strengthened to be a robust PCP of proximity, similarly to our main construct (specifically, Theorem 6.1.1, specialized to $m = 2$). However, our main construct achieves stronger parameters than those claimed in [Sze99], especially with respect to robust soundness. Indeed, the parameters claimed in [Sze99] only allow for the robust PCP of proximity to be composed with itself a constant number of times. In the language of Section 2, the soundness and robustness parameters obtained in [Sze99] are unspecified functions of the proximity parameter. In retrospect, it seems that the ideas of [PS94] may lead to a robust PCP of proximity with robustness that is at best linearly related to the proximity parameter; this would make the query complexity increase exponentially with the number of compositions (as discussed in Section 10.2.2). As mentioned above, a significant amount of our effort is aimed at ensuring that our robust PCP of proximity has sufficiently strong parameters to be composed a nonconstant number of times and moreoever to ensure that the query complexity grows only linearly rather than exponentially with the number of compositions. (See Section 10.2.2 for further explanation.)

CHAPTER 4

A constant-query, exponential-sized PCP of proximity

In this chapter, we construct a constant query PCP of proximity of exponential size (i.e., it uses polynomial amount of randomness) based on the Hadamard code. This PCP verifier will serve as the innermost verifier in all our uses of the Composition Theorem to finally reduce the query complexity to a constant. Since we use this PCP of proximity only in the innermost level of composition, we do not bother to construct a robust PCPP. However, its robustness is at least the reciprocal of its query complexity, which is a constant since this verifier is a constant-query PCPP.

4.1 Constant query PCP of proximity

For constructing such a verifier, we note that the Hadamard-code-based inner verifer from Arora *et al.* [ALM$^+$98] can be converted in to a PCP of proximity. The inner verifier of [ALM$^+$98] accesses $O(1)$ input oracles, where the ith oracle is supposed to provide the Hadamard *encoding* of some string w_i, and verifies that their concatenation satisfies some given circuit C.

Here we simplify this verifier to work with a single string w and the verifier accesses a *single input oracle* that represents this string itself (not some encoding of it), and verifies that w is close to an assignment acceptable by the circuit C given as explicit input.

Theorem 4.1.1 *There exists a constant $\delta_0 > 0$ such that there exists a PCP of proximity for* CIRCUIT VALUE *(for circuits of size n) with randomness complexity $O(n^2)$, query complexity $O(1)$, perfect completeness, soundness error $1 - \delta$, and proximity parameter 5δ for any $\delta \leq \delta_0$. That is, inputs that are δ-far from* *satisfying the circuit are rejected with probability at least* $\min(\delta, \delta_0)/5$.

Notice that we do not claim robustness of this PCP of proximity. This is because we don't intend to use this verifier (or any verifier derived from it) as the outer verifier during composition. However, this verifier is robust (in a trivial sense). Indeed, any PCP of proximity with $O(1)$ query complexity is trivially ρ-robust for some constant $\rho > 0$ (since the relative distance between two query patterns is lower-bounded by the inverse of number of bits queried).

Proof: Let V denote the claimed verifier. We first list the oracles used by V, then we describe the tests that V performs, and finally we will verify that V's complexities are as claimed and analyze its performance (most notably its soundness and proximity).

Oracles. Let C be a circuit with n gates on m input bits. The verifier accesses an input oracle $W : [m] \to \{0,1\}$ (representing a string $w \in \{0,1\}^m$), and a proof oracle $\Pi = (A, B)$, with $A : \{0,1\}^n \to \{0,1\}$ and $B : \{0,1\}^{n \times n} \to \{0,1\}$.

To motivate the verifier's tests, we describe what is expected from the oracles in the "completeness" case, i.e., when $C(w) = 1$. The input oracle, by definition, gives the string w, i.e., $W[i] = w_i$. Now let $z \in \{0,1\}^n$ be the string of values of all the gates of the circuit C (including the input, the internal gates, and the output gate(s)). W.l.o.g., assume $z = w \circ y$, where y represents the values assumed for internal gates. The oracle A is expected to give the values of all linear functions at z (over GF(2)); and the oracle B is supposed to give the value of all quadratic functions at z. More precisely $A = A[x]_{x \in \{0,1\}^n}$ is expected to be $A[x] = \sum_{i=1}^n x_i z_i = x^T z$ (where x and z are being thought of as column vectors). Similarly, $B = B[M]_{M \in \{0,1\}^{n \times n}}$ is expected to be $B[M] = \sum_{i,j} M_{ij} z_i z_j = z^T M z$ (where M is an $n \times n$ matrix). In order to verify that w satisfies C, the verifier will verify that A and B have indeed been constructed according to some string z as above, that z represents an accepting computation of the circuit, and finally that A is the encoding of some string $w' \circ y$ where w' is close to the string w given by the input oracle W.

Tests. Given the circuit C, the verifier first constructs polynomials $P_1(z), \ldots, P_n(z)$ as follows. Viewing the variables $\{z_i\}$ as representing the values at the individual gates of the circuit C (with $z_1, \ldots, z_m$ being the input gates), the polynomial $P_i(z)$ is the quadratic polynomial (over GF(2)) expressing the constraint imposed by the i'th gate of the circuit on an accepting computation. For example:

$$P_i(z) = \begin{cases} z_i - z_j z_k & \text{if the } i\text{th gate is an AND gate with inputs from gates } j \text{ and } k. \\ z_i - z_j - z_k + z_j z_k & \text{if the } i\text{th gate is an OR gate with inputs from gates } j \text{ and } k. \\ z_i - (1 - z_j) & \text{if the } i\text{th gate is a NOT gate with input from gate } j. \\ z_i - (z_j + z_k) & \text{if the } i\text{th gate is a PARITY gate with inputs from gates } j \text{ and } k. \\ 1 - z_j & \text{if the } i\text{th gate is an output gate with input from gate } j. \\ 0 & \text{if the } i\text{th gate is an input gate (i.e. } i \leq m). \end{cases}$$

Note that $z = w \circ y$ reflects the computation of C on an acceptable input w iff $P_i(z) = 0$ for every $i \in [n]$. The verifier conducts the following tests:

Codeword tests: These tests refer to (A, B) being a valid encoding of some string $z \in \{0,1\}^n$. That is, these tests check that both A and B are linear functions, and that B is consistent with A. In the latter check, the verifier employs a self-correction procedure (cf. [BLR93]) to the oracle B. (There is no need to employ self-correction to A, because it is queried at random locations.)

Linearity of A: Pick x_1, x_2 uniformly at random from $\{0,1\}^n$ and verify that $A[x_1 + x_2] = A[x_1] + A[x_2]$.

Linearity of B: Pick M_1, M_2 uniformly at random from $\{0,1\}^{n \times n}$ and verify that $B[M_1 + M_2] = B[M_1] + B[M_2]$.

Consistency of A and B: Pick x_1, x_2 uniformly at random from $\{0,1\}^n$ and M uniformly from $\{0,1\}^{n \times n}$ and verify that $B[M + x_1 x_2^T] - B[M] = A[x_1]A[x_2]$.

Circuit test: This test checks that the string z encoded in (A, B) represents an accepting computation of C; that is, that $P_i(z) = 0$ for every $i \in [n]$. The test checks that a random linear combination of the P_i's evaluates to 0, while employing self-correction to A and B.

Pick $\alpha_1, \ldots, \alpha_n \in \{0,1\}$ uniformly and independently and let $\sum_{k=1}^n \alpha_k P_k(z) = c_0 + \sum_i \ell_i z_i + \sum_{i,j} Q_{i,j} z_i z_j$. Pick $x \in \{0,1\}^n$ and $M \in \{0,1\}^{n \times n}$ uniformly at random. Verify that $c_0 + (A[x + \ell] - A[x]) + (B[M + Q] - B[M]) = 0$.

Proximity test: This test checks that the m-bit long prefix of the string z, encoded in A, matches (or is close to) the input oracle W, while employing self-correction to A.

Pick $j \in [m]$ and $x \in \{0,1\}^n$ uniformly. Let $e_j \in \{0,1\}^n$ denote the vector that is 1 in the jth coordinate and 0 everywhere else. Verify that $W[j] = A[x + e_j] - A[x]$.

The verifier accepts if all the tests above accept, else it rejects.

Resources. The verifier uses $O(n^2)$ random bits and makes $O(1)$ binary queries.

Completeness. It is straightforward to see that if w, the string given by W satisfies C, then letting z be the set of values of the gates of C and letting $A[x] = x^T z$ and $B[M] = z^T M z$ will satisfy all tests above. Thus the verifier has perfect completeness.

Soundness (with proximity). It follows directly from the analysis of [ALM$^+$98] that there exists a $\delta_0 > 0$ such that for every $\delta \leq \delta_0$, if the Codeword tests and the Circuit test above accept with probability at least $1 - \delta$ then the oracle A is 2δ-close to the Hadamard encoding of some string $z = w' \circ y$ such that $C(w')$ accepts. Now we augment this soundness with a proximity condition. Suppose the verifier also accepts the Proximity test with probability at least $1-\delta$. Then we have that $w_j \neq A[x+e_j] - A[x]$ with probability at most δ. Furthermore the events $A[x+e_j] \neq (x+e_j)^T z$, and $A[x] \neq x^T z$ happen with probability at most 2δ each. Thus, with probability at least $1 - 5\delta$ (over the possible choices of j and x), both $w_j = A[x+e_j] - A[x]$ and $A[x+e_j] - A[x] = (x+e_j)^T z - x^T z$ hold. Since $(x+e_j)^T z - x^T z = e_j^T z = z_j = w'_j$, it follows that, with probability at least $1 - 5\delta$ (over the choices of j), $w_j = w'_j$. In other words, the string w represented by the oracle W is at distance at most 5δ away from some string w' that is accepted by the circuit C. ∎

Part I

Proof of the PCP Theorem

CHAPTER 5

Proof of the PCP Theorem

5.1 Introduction

In this chapter, we give a proof of the PCP Theorem using the composition theorem from Chapter 3. This construction seems easier than the original proof of the PCP Theorem [AS98, ALM$^+$98]. This chapter along with the earlier four chapters gives a self-contained proof of the PCP Theorem (modulo the proof of the low-degree test analysis in Appendix A, which we cite from [ALM$^+$98]). Hopefully, this presentation will make the proof of the PCP Theorem accessible to a wider audience. The main theorem proved in this chapter is as follows:

Theorem 5.1.1 (PCP Theorem [AS98, ALM$^+$98]) *Satisfiability of circuits of size n can be probabilistically verified by probing a PCP of length $\mathrm{poly}(n)$ in $O(1)$ bit locations and tossing at most $O(\log n)$ random coins.*

Since CIRCUIT SATISFIABILITY is a complete language for $\mathbf{NP}$, this implies that any $\mathbf{NP}$ proof verifiable in time n can be encoded by a PCP of length $\mathrm{poly}(n)$ that is probabilistically verifiable by probing at most $O(1)$ bit locations.

The new composition theorem plays a key role in the simplification of the proof. We note that Dinur and Reingold [DR04] also obtain an alternate proof of the PCP Theorem using a similar composition theorem (see Section 3.4 for similarities and differences between our and their approaches). Both our work and theirs (especially) arose from a study of the proof of the PCP Theorem. Their chief motivation was to "combinatorialize" the proof of the PCP Theorem while ours was to construct shorter PCPs. In this chapter, neither do we attempt to give a "purely combinatorial" proof of the PCP Theorem nor do we construct very short PCPs. Instead the main purpose of this chapter is to demonstrate that a simpler (though yet algebraic) proof of the PCP Theorem can be obtained via the new composition theorem. In fact, our construction both in this chapter and later ones relies heavily on algebra, in contrast to the constructions of Dinur and Reingold – polynomials play a vital role in all our constructions.

5.1.1 Overview of Proof

The main construct in this chapter is the following robust PCP of proximity, which is constructed in Section 5.4.

Theorem 5.1.2 (ALMSS-type Robust PCP of proximity) *For all $n \in \mathbb{Z}^+$ and $\delta \in (0,1)$,* CIRCUIT VALUE *has a robust PCP of proximity (for circuits of size n) with the following parameters*

- *randomness $O(\log n)$,*
- *decision complexity $\mathrm{poly}(\log n)$, which also upper-bounds the query complexity.*
- *perfect completeness, and*
- *for proximity parameter δ, the verifier has robust-soundness error $1 - \Omega(\delta)$ with robustness parameter $\Omega(1)$.*

The above construct and the constant-query, exponential-sized PCPP described in Section 4 form the two building blocks for our composition. Note that the robust PCPP described in Theorem 5.1.2 has query complexity $\mathrm{poly}\log n$. We further reduce the query complexity to $\mathrm{poly}\log\log n$ by composing this robust PCPP with itself. This verifier is then composed with the Hadamard-based constant-query, exponential-sized PCPP for CKTVAL to obtain a constant-query, polynomial-sized PCPP for CKTVAL. This immediately yields a PCP for CIRCUIT SATISFIABILITY with similar parameters. The details of these compositions is given in Section 5.2, completing the proof of the PCP Theorem (Theorem 5.1.1).

Our main construct can be obtained by modifying the (pre-composition) PCP verifiers of [ALM$^+$98] to both have robust soundness and test proximity. For the sake of completeness, we present the complete construction of these robust PCPPs. Robust soundness can be obtained automatically from the "parallelized PCPs" of [ALM$^+$98] (see Proposition 2.4.6). However, in our construction, we obtain robust soundness by "bundling" the queries of the PCP verifier together. These robust PCPs are then converted into robust PCPs of proximity by augmenting them with appropriate "proximity tests".

For starters, we first construct robust PCPPs for two specific problems related to polynomials in Section 5.3 and then describe how a robust PCPP for CIRCUIT VALUE can be constructed using the robust PCPPs for these two problems in Section 5.4, thus proving Theorem 5.1.2.

5.2 *Composing the Main Construct*

The PCP Theorem (Theorem 5.1.1) is proved by constructing a constant query, polynomial-sized PCP for CIRCUIT SATISFIABILITY. Recall from Proposition 2.2.2, that it suffices to construct a PCP of proximity for CIRCUIT VALUE with similar parameters. This PCP of proximity is in turn constructed by composing the robust PCP of proximity described in Theorem 5.1.2 with itself. Note that a single application of the robust PCP of proximity of Theorem 5.1.2 reduces the query complexity from n to $\mathrm{poly}\log n$. Thus, two applications of this robust PCPP reduce the query complexity to $\mathrm{poly}\log\log n$. We then finally compose with the Hadamard code based constant query, exponential-sized PCPP constructed in Chapter 4 to obtain a constant query, polynomial-sized PCPP for CIRCUIT VALUE. The Hadamard-based verifier uses super-logarithmic amount of randomness (in fact, quadratic), however we can afford to use this verifier as we use it only in the innermost level of composition. Each of these composition is performed by invoking the Composition Theorem 3.2.1.

Proof (of Theorem 5.1.1): Let V_0 be the verifier obtained from Theorem 5.1.2. We compose V_0 with itself to obtain the verifier V_1. While doing so, we use the largest possible proximity parameter for the inner verifier (V_0): that is, we set the proximity parameter of the inner verifier to equal the robustness of the outer verifier. Note that the inner verifier works on circuits of size $\mathrm{poly}\log n$ and proximity parameter $\Omega(1)$ since the decision complexity and the robustness of the outer verifier are $\mathrm{poly}\log n$ and $\Omega(1)$ respectively. For circuits of size n and proximity parameter δ, the parameters of the composed verifier V_1 is as follows:

- randomness: $r_1(n) = O(\log n) + O\left(\log(\mathrm{poly}\log n)\right) = O\left(\log n\right)$.
- decision complexity: $d_1(n) = \mathrm{poly}\log(\mathrm{poly}\log n) = \mathrm{poly}\log\log n$.
- robustness: $\rho_1 = \Omega(1)$.
- soundness error: $s_1 = 1 - \Omega(\delta)\cdot\Omega(1) = 1 - \Omega(\delta)$.

We then compose V_1 with the Hadamard-based inner verifier V_h of Theorem 4.1.1 to obtain our final verifier V. The query complexity of V_h and hence that of V is constant. The randomness complexity of V is $r(n) \triangleq r_1(n) + r_h(d_1(n)) = r_1(n) + (\mathrm{poly}\log\log n)^2$, because $r_h(\ell) = O(\ell^2)$. Thus, $r(n) = O(\log n)$. On proximity parameter δ_h, the soundness error of V_h is $s_h = 1 - \Omega(\delta_h)$. Setting $\delta_h = \rho_1 = \Omega(1)$, we conclude that the soundness error of V on proximity parameter δ is $1 - \Omega(\delta)\cdot\Omega(1) = 1 - \Omega(\delta)$.

A soundness error of $1/2$ can be obtained by performing a constant number of repetitions of V. This increases the randomness and query complexity of the final verifier by at most a constant factor of $O(1/\delta)$. This yields a constant-query, $O(\log n)$ randomness PCPP verifier for CIRCUIT

VALUE with proximity parameter δ and soundness error $1/2$ for any constant $\delta \in (0,1)$. By Proposition 2.2.2, we have a PCP verifier for CIRCUIT SATISFIABILITY with similar parameters. This completes the proof of Theorem 5.1.1, modulo the construction of the robust PCPP described in Theorem 5.1.2.

■

5.3 *Robust PCPPs for Two Problems*

In this section, we construct robust PCPs of proximity for two problems related to polynomials – "low-degree testing" and "zero-on-subcube" problems. These constructions will serve as a preliminary step in our construction of a robust PCPP for CIRCUIT VALUE in Section 5.4. The low-degree testing problem is the problem of testing if a given function is close to some low-degree polynomial while the zero-on-subcube is a related testing problem of deciding whether a given function is close to a low-degree polynomial that vanishes on some subcube. Both these problems play a key role in almost all constructions of PCPs (including ours).

Talking about polynomials, the following lemma will come very useful.

Lemma 5.3.1 (Schwartz-Zippel Lemma [Sch80, Zip79]) *Suppose $p : F^m \to F$ is a non-zero m-variate polynomial of total degree at most d over the finite field F. Then*

$$\Pr_{x \in F^m}\Big[p(x) = 0\Big] \leq \frac{d}{|F|}$$

In other words, for a sufficiently large field, a non-zero polynomial cannot be zero at too many places.

Proof: We prove this by induction on the number of variables. The base case is easy: a non-zero univariate polynomial of degree d can have at most d zeros. Now, assume that we have proved the lemma for all $(m-1)$ variate polynomials. Suppose p is a non-zero m-variate polynomial in the variables $x_1, \ldots, x_m$. We can write p as a polynomial in the variable x_m (with $(m-1)$-variate polynomials as coefficients) as follows:

$$p(x_1, \ldots, x_n) = \sum_{i=0}^{k} p_i(x_1, \ldots, x_{m-1}) \cdot x_m^i$$

for some $k \leq d$ and $(m-1)$-variate polynomials $p_i, i = 1, \ldots, k$ in the variables $x_1, \ldots, x_{m-1}$. Furthermore, wlog, we can assume that the polynomial p_k is a non-zero polynomial in the $(m-1)$ variables $x_1, \ldots, x_{m-1}$. Clearly, the degree of p_k is at most $d-k$. By induction, for a random choice of $(x_1, \ldots, x_{m-1})$, the polynomial p_k is zero with probability at most $deg(p_k)/|F| \leq (d-k)/|F|$. For the values $(x_1, \ldots, x_{m-1})$ such that p_k is non-zero, for a random choice of the remaining variable

x_m, p can be zero with probability at most $k/|F|$. Hence, p is zero with probability at most $((d-k)+k)/|F| = d/|F|$. Thus proved. ■

5.3.1 *Low Degree Testing*

Our first problem related to polynomials is the classical problem of low-degree testing. Let F be a finite field and m, d be positive integers. Given a function $f : F^m \to F$ as a table of evaluations (for each point in the space F^m), the problem of low-degree testing is the problem of testing whether the function f is close to some polynomial $p : F^m \to F$ of total degree at most d. Here, distance (and hence closeness) between two functions $h_1, h_2 : F^m \to F$ is measured as the fractional number of points on which the functions disagree (i.e., the fractional Hamming distance between h_1 and h_2). Clearly, if the verifier is allowed to read the value of f at all points in the space F^m, then a simple interpolation solves this problem. The interesting case arises when the verifier is constrained to read only a few entries (say, at most a sub-linear fraction) of the entire table of values.

The low-degree testing problem is a well-studied problem. It was introduced by Rubinfeld and Sudan [RS96] and is an essential ingredient in almost all PCP constructions (with the significant exception of [DR04]). A number of different PCPPs with increasingly better parameters have been proposed for this problem (Line-Point Test [RS96, AS98, ALM$^+$98, AS97, BSVW03], Plane-Point Test [RS97]). Though the notion of PCPP is formally defined in this thesis ([BGH$^+$04a, BGH$^+$04b]), PCPPs are implicit in most low-degree testing. In fact, one of the earliest (implicit) uses of PCPP were for low-degree testing.

Almost all known PCPPs for this problem use the following basic property of polynomials (or some variant of it): The restriction of a multi-variate polynomial $f : F^m \to F$ of total degree at most d to any line in the space F^m is a univariate polynomial of degree at most d. Furthermore, if the size of the field F is sufficiently large compared to the degree d, then the converse also holds. That is, if the restriction of a function $f : F^m \to F$ to all lines in F^m is a univariate polynomial of degree at most d, then the function f is a multi-variate polynomial of total degree at most d. Arora *et al.* [ALM$^+$98] further strengthen this converse to obtain the following: if the restriction of a function $f : F^m \to F$ is a univariate polynomial of degree at most d for "most" lines in F^m, then the function f is "close" to a multi-variate polynomial of total degree at most d.

This suggests the following natural PCP of proximity for this problem, also called the LINE–POINT–TEST. The LINE–POINT–TEST works as follows: Besides the input oracle f, the LINE–POINT–TEST has oracle access to an auxiliary oracle $f_{\mathbb{L}}$, which we will call the "lines-oracle". For every line $\mathcal{L}$ in F^m, the lines-oracle $f_{\mathbb{L}}$ returns a univariate polynomial of degree at most d. This univariate polynomial is supposedly the restriction of the function f to the line $\mathcal{L}$. The LINE–POINT–TEST chooses a random line $\mathcal{L}$, queries the lines-oracle $f_{\mathbb{L}}$ on the line $\mathcal{L}$ and the input oracle f on a random point x on $\mathcal{L}$. It accepts iff the polynomial $f_{\mathbb{L}}(\mathcal{L})$ agrees with f at the point x.

The LINE–POINT–TEST is a PCPP verifier: the verifier has oracle access to the input $f : F^m \to F$ and the auxiliary lines oracle functions as the proof oracle to the verifier. Clearly, the LINE–POINT–TEST has perfect completeness. The soundness of this test is given by the following theorem due to Arora *et al.* [ALM$^+$98].

Theorem 5.3.2 ([ALM$^+$98], Theorem 65) *There exists universal constants $0 < \delta_0 < 1$ and $\alpha > 0$ such that the following holds. For all integers $m, d > 0$, $\delta < \delta_0$ and fields F of size at least αd^3, if $f : F^m \to F$ and $f_{\mathbb{L}} : \mathbb{L} \to P_d$ are two functions such that f is at least 2δ-far from any m-variate polynomial of degree at most d, then we have the following:*

$$\Pr[\text{LINE–POINT–TEST}^{f;\, f_{\mathbb{L}}} = \text{reject}] > \delta.$$

However, we will not employ this verifier for the low-degree testing problem. We will instead use a variant of this verifier, which we will call the PCPP–LDT verifier. Unlike the LINE–POINT–TEST, the PCPP–LDT will not have any auxiliary oracle. The only oracle it accesses is the input oracle $f : F^m \to F$. In this sense, the PCPP–LDT verifier is, in fact, a property tester for the low-degree testing problem. The formal details of the PCPP–LDT verifier are as follows:

PCPP–LDT$^f_{m,d}$ (i.e., with oracle access to the input function $f : F^m \to F$)

1. Choose a random line $\mathcal{L}$ in F^m (by choosing two random points in F^m) and query the oracle f on all points on the line $\mathcal{L}$.
2. Verify if the restriction of f to the line $\mathcal{L}$ is a univariate polynomial of degree at most d. If so, accept else reject.

Complexity of Verifier* PCPP–LDT*: The PCPP verifier makes $|F|$ queries each of which expects as an answer an element of the field F (i.e., a string of length $\log |F|$). Hence, the total (bit) query complexity is $|F| \log |F|$. The verifier chooses two random points in F^m. Hence, the randomness complexity is at most $2m \log |F|$. Clearly, this verifier accepts all polynomials p of total degree at most d (i.e., it has perfect completeness). Also, note that this PCPP verifier does not require a proof oracle at all, it only queries the input oracle f.

We now estimate the expected robustness of the PCPP–LDT verifier as follows: Suppose the field F is of size at least αd^3 and $\delta < \delta_0$ where α and δ_0 are the universal constants specified in Theorem 5.3.2. For each line $\mathcal{L}$ in F^m, define $f_{\text{lines}}(\mathcal{L})$ to be the degree d univariate polynomial having maximum agreement with f on $\mathcal{L}$, breaking ties arbitrarily. For any line $\mathcal{L}$, the PCPP–LDT verifier accepts $f|_{\mathcal{L}}$ if this restriction exactly matches with the polynomial $f_{\text{lines}}(\mathcal{L})$. Suppose, $f|_{\mathcal{L}}$ does not exactly agree with the polynomial $f_{\text{lines}}(\mathcal{L})$. We then observe that the number of points on the line $\mathcal{L}$ where the value of f needs to be changed in order to make the verifier accept is exactly the number of points on which the two functions $f|_{\mathcal{L}}$ and $f_{\text{lines}}(\mathcal{L})$ differ. In other words, the distance of $f|_{\mathcal{L}}$ to satisfying PCPP–LDT is precisely $\Delta(f|_{\mathcal{L}}, f_{\text{lines}}(\mathcal{L}))$.

By inspection, the probability that LINE–POINT–TEST$^{f;\, f_{\text{lines}}}$ rejects the points-oracle f and lines-oracle f_{lines} as defined above equals $\mathrm{E}_{\mathcal{L}}[\Delta(f|_{\mathcal{L}}, f_{\text{lines}}(\mathcal{L}))]$. By Theorem 5.3.2, if f is 2δ-far from every total degree d polynomial, then LINE–POINT–TEST$^{f, f_{\text{lines}}}$ rejects with probability at least δ. Thus, the expected number of points on the random line $\mathcal{L}$ where the value of f needs to be changed in order to make the verifier accept is at least δ. This completes the robustness analysis of the PCPP–LDT verifier. Summarizing, we have the following lemma.

Lemma 5.3.3 *Let α and δ_0 be the universal constants that appear in Theorem 5.3.2. For all integers $m, d > 0$, $\delta < \delta_0$ and fields F of size at least αd^3, the* PCPP–LDT *verifier has randomness complexity $2m \log |F|$, query complexity $|F| \log |F|$ and perfect completeness. Furthermore, if the input oracle f is 2δ-far from any polynomial of total degree d, then the expected number of points on the random line $\mathcal{L}$ where the value of f needs to be changed in order to make the* PCPP–LDT *verifier accept is at least δ.*

5.3.2 Zero on Subcube

Our next problem related to polynomials is the "zero on subcube" problem. Let F be a finite field, H a subset of F and m, d be positive integers. Given a function $f : F^m \to F$ as a table of evaluations, the "zero on subcube" problem is the problem of testing whether f is close to some polynomial $p : F^m \to F$ of total degree at most d, that vanishes on the entire subcube H^m (in other words, $p|_{H^m} \equiv 0$). As in the case of the low-degree testing problem, if the verifier is allowed to read the value of f at all points in the space F^m, then a simple interpolation solves this problem. Hence, the interesting case arises when the verifier is constrained to read only a few entries (say, at most a sub-linear fraction) of the entire table of values.

We now describe a PCPP verifier to solve the "zero on subcube" problem. As a first step, we run the PCPP–LDT verifier on the function f to check if f is close to some low-degree polynomial. Thus, if f is far from any low-degree polynomial, then this initial step rejects. We are now left with handling those functions f that are close to some low-degree polynomials.

First for some notation. For any subset $S \subseteq F$, let $g_S : F \to F$ be the univariate polynomial of degree $|S|$ that vanishes exactly on the points of S. In other words, $g_S(x) = \prod_{s \in S}(x - s)$. For notational convenience, we will some times refer to the polynomial p as p_0. Consider the following sequence of divisions, for $i = 1, \ldots, m$.

$$p_{i-1}(x_1, \ldots, x_m) \quad \equiv \quad g_H(x_i) \cdot q_i(x_1, \ldots, x_m) + p_i(x_1, \ldots, x_m) \qquad (5.1)$$

where in the i^{th} step the polynomial p_{i-1} ($p = p_0$ in the first step) is divided by the univariate polynomial $g_H(x_i)$ of degree $|H|$ to obtain quotient q_i and remainder p_i. Hence, each of the quotient polynomials q_i is of total degree at most $d - |H|$ and the remainder polynomials p_i are of total degree at most d. Furthermore, since at the i^{th} stage we are dividing by a univariate polynomial in the variable x_i of degree $|H|$, it can be inductively shown that the degree of the polynomial p_i in each of the first i variables $x_1, \ldots, x_i$ is less than $|H|$. Hence, p_m is a polynomial of degree less than $|H|$ in each of its variables. Adding all the identities in Equation (5.1) for $i = 1, \ldots, m$, we have the following identity:

$$p(x_1, \ldots, x_m) \equiv \sum_{i=1}^{m} g_H(x_i) \cdot q_i(x_1, \ldots, x_m) + p_m(x_1, \ldots, x_m)$$

Since $g_H(h) = 0$ for all $h \in H$, we have that $p(x) = p_m(x)$ for all $x \in H^m$. Hence, the polynomial p vanishes on H^m iff the polynomial p_m vanishes on H^m. But, p_m is polynomial of degree less than $|H|$ in each of its variables. Hence, p_m vanishes on H^m if and only if it is the zero-polynomial. Summarizing these observations, we have the following proposition.

Proposition 5.3.4 ([BS04]) [1] *Suppose F is a field, H a subset of F and m, d be positive integers. A polynomial $p_0 : F^m \to F$ of total degree at most d is zero on the entire subcube H^m iff there exists polynomials $q_1, \ldots, q_m : F^m \to F$ of total degree at most $d - |H|$ each and another sequence of polynomials $p_1, \ldots, p_m$ of total degree at most d such that*

$$\begin{aligned} p_{i-1}(x_1, \ldots, x_m) &\equiv g_H(x_i) \cdot q_i(x_1, \ldots, x_m) + p_i(x_1, \ldots, x_m), \quad i = 1, \ldots, m && (5.2) \\ p_m(x_1, \ldots, x_m) &\equiv 0 && (5.3) \end{aligned}$$

This proposition suggests the following natural PCP of proximity for the "zero on subcube" problem: To check if f is close to some polynomial $p : F^m \to F$ of total degree at most d that vanishes on the subcube H^m, the verifier expects as proof, the set of functions $P_1, \ldots, P_m, Q_1, \ldots, Q_m : F^m \to F$ (each as a table of values), which are supposedly the polynomials $p_1, \ldots, p_m, q_1, \ldots, q_m$ indicated above. The verifier runs the PCPP–LDT verifier on each of the functions $f, P_1, \ldots, P_m$, $Q_1, \ldots, Q_m$ to verify that they are close to low-degree polynomials. The verifier then chooses a random point $x = (x_1, \ldots, x_m) \in F^m$ and queries each of the functions $f, P_1, \ldots, P_m, Q_1, \ldots, Q_m$ at the point x. It then checks if the following equations are satisfied at the point x. As before, for

[1]Another related (and simpler) characterization of a polynomial that vanishes on the subcube H^m follows from the above discussion.

Proposition 5.3.5 ([BS04]) *Suppose F is a field, H a subset of F and m, d be positive integers. A polynomial $p : F^m \to F$ of total degree at most d is zero on the entire subcube H^m iff there exists polynomials $q_1, \ldots, q_m : F^m \to F$ of total degree at most $d - |H|$ each such that*

$$p(x_1, \ldots, x_m) \equiv \sum_{i=1}^{m} g_H(x_i) \cdot q_i(x_1, \ldots, x_m).$$

However, we will not use this characterization. Ben-Sasson and Sudan [BS04] use this simpler characterization in their construction of even shorter PCPs notational convenience, we will sometimes refer to the function f as P_0.

$$\text{DivisionCheck}: \quad P_{i-1}(x_1,\dots,x_m) = g_H(x_i)\cdot Q_i(x_1,\dots,x_m) + P_i(x_1,\dots,x_m), \quad i = 1,\dots,m \tag{5.4}$$

$$\text{IdentityCheck}: \quad P_m(x_1,\dots,x_m) = 0 \tag{5.5}$$

If all the above checks pass, the verifier accepts else it rejects.

Clearly, if f is a polynomial of total degree at most d that vanishes on the subcube H^m, then there exist functions $P_1,\dots,P_m,Q_1,\dots,Q_m$ (by Proposition 5.3.4) such that the above verifier accepts with probability 1. Now, for the soundness, suppose f is far from any polynomial of total degree at most d that vanishes on H^m. If any of the functions $f,P_1,\dots,P_m,Q_1,\dots,Q_m$ is 2δ-far from being a low-degree polynomial, then the corresponding PCPP–LDT test recognizes the error. So we can assume that each of the functions are 2δ-close to being low-degree. Let $p_0,p_1,\dots,p_m,q_1,\dots,q_m$ be the low-degree polynomials that are 2δ-close to the functions $f,P_1,\dots,P_m,Q_1,\dots,Q_m$ respectively. Now these polynomials must not satisfy either the identity (5.2) for some $i=1,\dots,m$ or the identity (5.3) since otherwise f would be close to a low-degree polynomial that vanishes on H^m. Suppose the polynomials p_{i-1},p_i and q_i do not satisfy the identity (5.2). Then by the Schwartz-Zippel Lemma (Lemma 5.3.1), for most points (in fact for at least $1-\frac{d}{|F|}$ fraction of the points) in F^m, the identity must be violated. Since, the functions P_{i-1},P_i and Q_i are 2δ-close to the polynomials p_{i-1},p_i and q_i, the functions P_{i-1},P_i and Q_i must violate Equation (5.4) for at least $1-\frac{d}{|F|}-3\cdot 2\delta$ fraction of the points. Thus, the verifier detects the error with probability at least $1-\frac{d}{|F|}-6\delta$. The case when $p_m \not\equiv 0$ is similar.

The above discussion shows that this verifier is a PCPP verifier for the "zero-on-subcube" problem. However, recall that we had intended to construct a robust PCPP for this problem. A robust verifier on a NO instance not only has to reject with high probability but also has to reject "strongly" with high probability, i.e., with high probability, the bits read by the robust verifier must be far from satisfying the tests of the verifier. Unfortunately, this verifier is not robust for the following reason: Suppose the input function $f : F^m \to F$ is 2δ-far from any polynomial of total degree that vanishes on H^m. We then know that one of the several tests performed by the verifier detects this fact with non-negligible probability. However, this test is only one of the $O(m)$ tests performed by the verifier (the verifier performs $(2m+1)$ runs of the PCPP–LDT, m Division Checks and 1 Identity Check). Hence, the bits read by this test comprise a small fraction of the total query complexity of the verifier. For instance, the number of bits read by a single PCPP–LDT is about $1/m$ times the query complexity. This causes the robustness of the verifier to drop by a factor of at least m. Hence, the above described verifier is not robust.

To "robustify" the verifier, we "bundle" the various functions in the proof oracle so that the inputs required for the several test instances can be read together. This helps us construct a robust

PCPP verifier, albeit over a larger alphabet. Instead of $2m$ different functions $\{P_i\}, \{Q_i\}$ in the proof oracle, we have one oracle Π which bundles together the data of all these functions. The oracle $\Pi : F^m \to F^{2m}$ is supposed to satisfy $\Pi(x) = (P_1(x), \ldots, P_m(x), Q_1(x) \ldots, Q_m(x))$ for all $x \in F^m$. The new verifier now chooses a random line $\mathcal{L}$ in F^m, queries the input oracle f and Π for all points along the line $\mathcal{L}$. It then unbundles $\Pi(\mathcal{L})$ to obtain the value of each of the functions $\{P_i\}$ and $\{Q_i\}$ on the line $\mathcal{L}$. It can now perform the low-degree test for each of the functions $\{f, P_1, \ldots, P_m, Q_1, \ldots, Q_m\}$ and also the division check and identity check for each point along the line $\mathcal{L}$. The formal description of this verifier is as follows:

ROBUST-PCPP–ZERO-ON-SUBCUBE$^{f;\,\Pi}_{m,H,d}$
(i.e., with oracle access to the input function $f : F^m \to F$ and proof oracle $\Pi : F^m \to F^{2m}$ where H is a subset of the field F)

1. Choose a random line $\mathcal{L}$ in F^m and query the oracles f and Π on all points on the line $\mathcal{L}$.
2. Unbundle $\Pi(\mathcal{L})$ to obtain the value of the functions $P_i, Q_i, i = 1 \ldots, m$ on all points on the line $\mathcal{L}$.
3. Reject if the restriction of any of the functions $f, P_1, \ldots, P_m$ to the line $\mathcal{L}$ is not a univariate polynomial of degree at most d or if the restriction of any of the functions $Q_1, \ldots, Q_m$ to the line $\mathcal{L}$ is not a univariate polynomial of degree at most $d - |H|$.
4. For each point x on the line $\mathcal{L}$,
   (a) For each $i = 1, \ldots, m$, verify that the Division Check (Equation (5.4)) is satisfied for this value of i.
   (b) Verify that the Identity Check (Equation (5.5)) is satisfied.
5. Accept if none of the above tests fail, else reject.

Remark 5.3.6 *The input oracle f returns elements of F while the proof oracle Π returns elements of F^{2m}. We will however assume that both the oracles f and Π return elements of F^{2m}. This can be implemented as follows: On input x, the oracle f returns the element $(f(x), 0, 0, \ldots, 0) \in F \times 0^{2m-1} \subseteq F^{2m}$. This assumption will simplify the soundness analysis of the verifier.*

Complexity of Verifier* ROBUST-PCPP–ZERO-ON-SUBCUBE*: The PCPP verifier makes $|F|$ queries each to the input oracle f and the proof oracle Π. By Remark 5.3.6, both the input oracle f and the proof oracle Π return elements of F^{2m}. Hence, the total (bit) query complexity is $4m|F| \log |F|$. The randomness complexity is at most $2m \log |F|$. Also, if the input function f is indeed a polynomial of total degree at most d that vanishes on the subcube H^m, then by Proposition 5.3.4, there exists a proof oracle Π that causes the verifier to accept with probability 1 (i.e., it has perfect completeness).

The robust-soundness of the verifier is given by the following claim.

Lemma 5.3.7 *Let α and δ_0 be the universal constants that appear in Theorem 5.3.2. For all integers $m, d > 0$, $\delta < \delta_0$ and fields F of size at least $\max\{\alpha d^3, d/(1-7\delta)\}$, if f is 2δ-far from any polynomial of total degree d that vanishes on the subcube H^m, then for any proof oracle $\Pi : F^m \to F^{2m}$, for a random line $\mathcal{L}$ the expected distance of $(f|_{\mathcal{L}}, \Pi|_{\mathcal{L}})$ from satisfying the* ROBUST-PCPP–ZERO-ON-SUBCUBE *verifier is at least $\delta/2$.*

Proof: Suppose f is 2δ-far from any polynomial of total degree d that vanishes on the subcube H^m. Consider the behavior of the verifier on this input function f and any proof oracle $\Pi : F^m \to F^{2m}$. Unbundle the proof oracle $\Pi : F^m \to F^{2m}$ to obtain the individual functions $P_i : F^m \to F, Q_i : F^m \to F, i = 1, \ldots, m$. Let $p_0, p_1, \ldots, p_m$ be the closest polynomials of total degree at most d to the functions $f, P_1, \ldots, P_m$ respectively and $q_1, \ldots, q_m$ the closest polynomials of total degree at most $d - |H|$ to the functions $Q_1, \ldots, Q_m$ respectively (if there is more than one polynomial, break ties arbitrarily). Recall that the verifier makes three different types of tests – the low-degree test (in Step. 3) and the division check (in Step. 4(a)) and the identity check (in Step. 4(b)). The input for all these tests is the same, namely $(f|_{\mathcal{L}}, \Pi|_{\mathcal{L}})$. The verifier could detect an inconsistency in any of these tests. Corresponding to these three tests, we have the following three cases. For notational convenience, we will sometimes refer to the function f as P_0.

Case I: Either f is 2δ-far from the polynomial p_0 or one of the functions Q_i is 2δ-far from the corresponding polynomial q_i or one of the functions P_i is 2δ-far from the corresponding polynomial p_i. Wlog, assume that P_1 is 2δ-far from the polynomial p_1. In this case, the analysis of the PCPP–LDT verifier shows that the expected distance of $P_1|_{\mathcal{L}}$ from being a degree d polynomial is at least δ. Translating this to the bundled alphabet, we have that the expected distance of $\Pi|_{\mathcal{L}}$ from satisfying the low-degree test is at least δ. However, $\Pi|_{\mathcal{L}}$ is only half the input read by the verifier. Hence, the expected distance of the total input read by the verifier, which is $(f|_{\mathcal{L}}, \Pi|_{\mathcal{L}})$, from satisfying the low-degree test is at least $\delta/2$.

Case II: Suppose for some i each of the functions P_{i-1}, P_i and Q_i are close to the corresponding polynomials p_{i-1}, p_i and q_i, but the Polynomial Identity (5.2) does not hold for this value of i. Then by the Schwartz-Zippel Lemma 5.3.1, for at least $1 - \frac{d}{|F|}$ fraction of the points in F^m, the polynomials p_{i-1}, p_i and q_i violate Equation (5.2) for this i. Let B be the set of points in F^m where the polynomials p_{i-1}, p_i and q_i violate (5.2) and where each of the functions P_{i-1}, P_i and Q_i agrees with its corresponding closest polynomial. The fractional size of B is at least $1 - \frac{d}{|F|} - 6\delta$ since each of the functions is at least 2δ-close to its corresponding to its corresponding polynomial. For the verifier to accept in Step. 4(a) for this value of i, either $f|_{\mathcal{L}}$ or $\Pi|_{\mathcal{L}}$ must be changed for each of the points in the set $B \cap \mathcal{L}$. Since $\mathcal{L}$ is a random line, the expected size of $B \cap \mathcal{L}$ is at least $1 - \frac{d}{|F|} - 6\delta$. Hence, the expected distance of $(f|_{\mathcal{L}}, \Pi|_{\mathcal{L}})$ from satisfying the Division Check in Step. 4(a) for this value of i is at least $\frac{1}{2} \cdot \left(1 - \frac{d}{|F|} - 6\delta\right)$.

Case III: Suppose the function P_m is 2δ-close to the polynomial p_m but the polynomial p_m is not identically zero. Then, by an analysis similar to that in Case II, it can be shown that the expected distance of $(f|_{\mathcal{L}}, \Pi|_{\mathcal{L}})$ from satisfying the Identity Check in Step. 4(b) is at least $\frac{1}{2} \cdot \left(1 - \frac{d}{|F|} - 2\delta\right)$.

If none of the above occur, it must be the case that f is 2δ-close to the polynomial p of total degree at most d which vanishes on H^m, contradicting the hypothesis. Hence, one of the three cases must occur.

Since the input read is the same for all these cases, the expected robustness is at least

$$\min\left\{\frac{\delta}{2}, \frac{1}{2}\left(1 - \frac{d}{|F|} - 6\delta\right), \frac{1}{2}\left(1 - \frac{d}{|F|} - 2\delta\right)\right\}$$

which is $\delta/2$ since $7\delta + \frac{d}{|F|} \leq 1$. ∎

5.4 *A robust PCPP for* Circuit Value

In this section, we construct the robust PCPs of proximity for Circuit Value described in Theorem 5.1.2. This construction proceeds in two steps. In the first step, we construct (in Section 5.4.1) a robust PCP for CktSAT using the robust PCPs for the two problems described in Section 5.3. In the second step, we convert this robust PCP into a robust PCP of proximity for CktVal by augmenting with an appropriate proximity test (in Section 5.4.2).

5.4.1 *A robust PCP for* Circuit Satisfiability

In this section, we present a preliminary version of Theorem 5.1.2, without the proximity properties. More specifically, we construct a robust PCP for CktSAT. The construction of the robust-PCP proceeds in two steps. First, we give an algebraic description of the input circuit C. We then show how this algebraic description can be reduced to the 'zero on subcube" problem. We now use the robust PCP for the "zero on subcube" to construct a robust PCP for CktSAT.

Algebraic description of circuit

Recall that our circuits have fan-in 2 and fan-out 2. Furthermore, we will assume that there are only three types of gates – binary OR-gate, unary NOT-gate and the unary OUTPUT-gate. Any circuit with arbitrary unary and binary Boolean gates can be transformed into one of the above form.

Let C be the input circuit of size n on k input variables. Let us index the input Boolean variables of the circuit as $w_1, \ldots, w_k$ and the remaining gates of the circuit as $w_{k+1}, \ldots, w_n$ with w_n being the index of the output variable. Usually an assignment refers to a mapping from the input variables to Boolean values, but we will refer to an "extended" assignment which is a mapping from the set of all variables (input and gate) to Boolean values. Thus, an assignment to C is described by a map $A : [n] \to \{0,1\}$. Note that an "extended" assignment contains within it the assignment to the input variables. For any gate (OR, NOT or OUTPUT), we refer to any assignment to the input and output variables of the gate that violates the functionality of the gate as an *invalid configuration* for that gate. For instance, an assignment that maps both the input variables of a OR-gate to 1 and the output variable to 0 is an invalid configuration for that OR-gate. Similarly, an assignment that maps the output variable to 0 is an invalid configuration for the OUTPUT-gate (since the output always needs to be 1 for a satisfying assignment). In this terminology, a satisfying assignment does not lead to an invalid configuration for any gate and conversely, only a satisfying assignment has this property.

To obtain a algebraic description of the circuit C, choose a field F and any set $H \subseteq F$ such that $\{0,1\} \subseteq H$ ([2]) and $|H^m| = n$. We will identify the set of elements $[n] = \{1, \dots, n\}$ with the set H^m. We this identification, we can view the assignment A as a mapping $A : H^m \to \{0,1\} \subseteq H \subseteq F$.

Any assignment $S : H^m \to H$ can be interpolated to obtain a polynomial $\widehat{S} : \mathrm{F}^m \to \mathrm{F}$ of degree at most $|H|$ in each variable (and hence a total degree of at most $d = m|H|$) such that $\widehat{S}|_{H^m} = S$ (i.e., the restriction of $\widehat{S}$ to H^m coincides with the function S). Conversely, any polynomial $\widehat{S} : \mathrm{F}^m \to \mathrm{F}$ can be interpreted as an assignment from H^m to F by considering the function restricted to the subcube H^m.

Thus, corresponding to every assignment $A : [n] \to \{0,1\}$ we have an interpolated polynomial $\widehat{A} : F^m \to F$ of degree at most $|H|$ in each variable. Furthermore, any function $R : F^m \to F$ can be interpreted as a Boolean assignment by assigning 0 to "false", 1 to "true" and all other field values to "I don't know".

Based on the circuit C, we present a polynomial transformation rule that converts the assignment polynomial $\widehat{A} : F^m \to F$ to another polynomial $p_{(\widehat{A})} : F^{3m+3} \to F$ on a slightly larger domain F^{3m+3} such that $p_{(\widehat{A})}|_{H^{3m+3}} \equiv 0$ iff $\widehat{A}$ is a satisfying assignment (i.e, the polynomial $p_{(\widehat{A})}$ vanishes on the subcube H^{3m+3} if and only if $\widehat{A}$ encodes a satisfying assignment). For the transformation rule, we first encode the circuit C as function $C' : [n]^3 \times H^3 \to \{0,1\}$. The function C' is defined as follows:

$$C'(i_1, i_2, i_3, b_1, b_2, b_3) = \begin{cases} 1 & \text{The elements } b_1, b_2, b_3 \text{ are in the set } \{0,1\} \text{ and there exists a gate} \\ & \text{whose input variables and output variables are in the set} \\ & \{w_{i_1}, w_{i_2}, w_{i_3}\} \text{ and } (w_{i_1} = \overline{b_1}) \wedge (w_{i_2} = \overline{b_2}) \wedge (w_{i_3} = \overline{b_3}) \text{ is an} \\ & \text{invalid configuration for this gate.} \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

For instance, if there exists an OR-gate in the circuit C with input variables w_2 and w_7 and whose

[2]We don't need to use $\{0,1\}$; any embedding of {true, false} will do.

output variable is w_{11}, then $C'(2,7,11,0,0,1) = 1$ since $w_2 = 1, w_7 = 1$ and $w_{11} = 0$ is an invalid configuration for this OR-gate. Similarly, if w_{21} is the index of the output variable, then $C'(21,21,5,1,1,0) = 1$ since $w_{21} = 0$ is an invalid configuration for OUTPUT-gate (the output must be 1 for a satisfying assignment).

Recalling that $H^m \approx [n]$, we can view the function C' as a mapping $C' : H^{3m+3} \to \{0,1\} \subseteq F$. We then construct a polynomial $\widehat{C} : F^{3m+3} \to F$ in $3m+3$ variables of degree at most H in each variable (and hence a total degree of at most $(3m+3)|H|$) such that $\widehat{C}|_{H^{3m+3}} \equiv C'$. The polynomial transformation rule mapping the polynomial $q : F^m \to F$ to $p_{(q)} : F^{3m+3} \to F$ is as follows:

$$\begin{aligned} p_{(q)}(x_1,\ldots,x_{3m+3}) = \widehat{C}(x_1,\ldots,x_{3m+3}) \cdot \big(q(x_1,\ldots,x_m) - x_{3m+1}\big) \\ \cdot \big(q(x_{m+1},\ldots,x_{2m}) - x_{3m+2}\big) \cdot \big(q(x_{2m+1},\ldots,x_{3m}) - x_{3m+3}\big) \end{aligned} \tag{5.7}$$

When the polynomial q is the assignment polynomial $\widehat{A} : F^m \to F$ corresponding to the assignment $A : [n] \to \{0,1\}$ and the input $(x_1,\ldots,x_{3m+3})$ to the polynomial $p_{(\widehat{A})}$ is in the set H^{3m+3} (i.e., $(x_1,\ldots,x_{3m+3}) = (i_1,i_2,i_3,b_1,b_2,b_3)$ for some $i_1,i_2,i_3 \in H^m$ and $b_1,b_2,b_3 \in H$), we have

$$p_{(\widehat{A})}(i_1,i_2,i_3,b_1,b_2,b_3) = \widehat{C}(i_1,i_2,i_3,b_1,b_2,b_3)\big(A(i_1)-b_1\big)\big(A(i_2)-b_2\big)\big(A(i_3)-b_3\big) \tag{5.8}$$

For any $(i_1,i_2,i_3,b_1,b_2,b_3) \in H^{3m+3}$, the above quantity is non-zero exactly when $\{b_1,b_2,b_3\} \subseteq \{0,1\}$ and there exists a gate whose input and output variables are in the set $\{w_{i_1}, w_{i_2}, w_{i_3}\}$ and if the assignment A allows the invalid configuration $(w_{i_1} = \overline{b_1}) \wedge (w_{i_2} = \overline{b_2}) \wedge (w_{i_3} = \overline{b_3})$. Hence, $p_{(\widehat{A})}|_{H^{3m+3}} \equiv 0$ if and only if the the assignment polynomial $\widehat{A}$ encodes a satisfying assignment A. This holds even when the assignment represented by $\widehat{A}$ has some variables set to "I don't know".

Note that if q is a polynomial of total degree at most $d = m|H|$, then the polynomial transformation rule maps it to a polynomial $p_{(q)} : F^{3m+3} \to F$ of total degree at most $d' = (6m+3)|H|$ (since $\widehat{C}$ is polynomial of total degree at most $(3m+3)|H|$).

Summarizing the above discussion, we have the following proposition:

Proposition 5.4.1 *There exists a polynomial reduction $\mathcal{R}$ which maps every circuit C of size n, a finite field F and parameter $m \le \log n / \log\log n$ to a polynomial transformation rule $\big(\{q : F^m \to F\} \longrightarrow \{p_{(q)} : F^{3m+3} \to F\big)$ given by Equation (5.7) where H a subset of F satisfying $\{0,1\} \subseteq H \subseteq F$ and $|H| = n^{1/m}$. The polynomial transformation rule satisfies the following properties:*

- *if C is satisfiable, then there exists a polynomial $\widehat{A} : F^m \to F$ of total degree $m|H|$ such that $p_{(\widehat{A})}|_{H^{3m+3}} \equiv 0$.*
- *if C is not satisfiable, then for all functions $q : F^m \to F$, the restriction $p_{(q)}|_{H^{3m+3}}$ is not identically zero.*

Furthermore, the mapping $\big(q \longrightarrow p_{(q)}\big)$ maps polynomials q of total degree $d = m|H|$ to polynomials $p_{(q)}$ of total degree at most $d' = (6m+3)|H|$.

Remark 5.4.2 *We observe that if C is a satisfiable circuit, then any polynomial $\widehat{A} : F^m \to F$ that satisfies $p_{(\widehat{A})}|_{H^{3m+3}} \equiv 0$ contains within it a satisfying assignment to the circuit C. Specifically, let I be the subset of H that corresponds to the input variables. Then the restriction of the function $\widehat{A}$ to the set I, $\widehat{A}|_I : [k] \to F$ is a satisfying assignment to the circuit C. Conversely, every satisfying assignment $A : [k] \to \{0,1\}$ to C can be extended to a polynomial $\widehat{A} : F^m \to F$ of total degree $m|H|$ such that $p_{(\widehat{A})}|_{H^{3m+3}} \equiv 0$ and $\widehat{A}|_I$ contains the satisfying assignment.*

Robust PCP for CKTSAT

Given Proposition 5.4.1 and a robust-PCPP for the "zero-on-subcube" problem, it is easy to construct a robust-PCP for CKTSAT. The verifier expects as proof the table of values $\tilde{A} : F^m \to F$ and $P : F^{3m+3} \to F$ which are supposedly the evaluations of the polynomials $\widehat{A} : F^m \to F$ and $p_{(\widehat{A})} : F^{3m+3} \to F$. The verifier then checks that $\tilde{A}$ and P are low-degree polynomials. It then checks that the function P is the function obtained by the polynomial transformation rule from the function $\tilde{A}$ (i.e., it checks if Equation (5.7) is satisfied at a few random points). It then uses an additional proof oracle $\Pi : F^{3m+3} \to F^{6m+6}$ (as in the "zero-on-subcube" problem) to check that the function P vanishes on the subcube H^{3m+3}. The formal details of the robust PCP-Verifier are as follows:

ROBUST-PCP–CIRCUIT-SAT$_m^{\tilde{A},P,\Pi}(C, F)$
(i.e., with oracle access to the proof oracles $\tilde{A} : F^m \to F, P : F^{3m+3} \to F$ and $\Pi : F^{3m+3} \to F^{6m+6}$ where F is a finite field.)

1. Choose a subset H of the field F of size $|H| = n^{1/m}$. Set $d = m \cdot n^{1/m}$ and $d' = (6m+3)n^{1/m}$.
2. Run PCPP–LDT$_{m,d}^{\tilde{A}}$ (to check that $\tilde{A}$ is close to a polynomial of total degree at most d).
3. Run ROBUST-PCPP–ZERO-ON-SUBCUBE$_{3m+3,H,d'}^{P;\,\Pi}$ (to check if P is close to a polynomial of total degree at most d' that vanishes on the subcube H^{3m+3}).
4. Choose a random line $\mathcal{L}$ in F^{3m+3}.
5. For every point $x = (z_1, z_2, z_3, y_1, y_2, y_3)$ on the line $\mathcal{L}$ where $z_1, z_2, z_3 \in F^m$ and $y_1, y_2, y_3 \in F$,
   - Query the oracle P on point x and the oracle $\tilde{A}$ on the points z_1, z_2, z_3 and reject if the following relation is violated.

$$P(x) = \widehat{C}(x)\big(\tilde{A}(z_1) - y_1\big)\big(\tilde{A}(z_2) - y_2\big)\big(\tilde{A}(z_3) - y_3\big) \tag{5.9}$$

6. Accept if none of the above tests reject.

Remark 5.4.3 *1. For the present, we provide both the input circuit C and the field F as explicit inputs to the verifier. However, later (specifically, in the proof of Theorem 5.1.2 in Section 5.4.3), we will show how to construct a suitable sized field F using the size of the circuit and thus remove the field F from the list of inputs to the verifier.*

2. As in the case of the ROBUST-PCPP–ZERO-ON-SUBCUBE *verifier, we will assume that all the oracles $\tilde{A}, P, \Pi$ return elements of $F^{(6m+6)}$. This will simplify the soundness analysis of the verifier.*

Complexity of the ROBUST-PCP–CIRCUIT-SAT ***Verifier:*** The PCP verifier makes $O(|F|)$ queries to each of the proof oracles $\tilde{A}, P$ and Π. By Remark 5.4.3(2), all the oracles return elements from F^{6m+6}. Hence, the total (bit) query complexity is $O(m|F|\log|F|)$. The randomness complexity is at most $O(m \log |F|)$. Also, if the circuit C is satisfiable, then there exists proof oracles $\tilde{A}, P$ and Π which cause the verifier to accept with probability 1 (i.e., it has perfect completeness).

The robust-soundness of the verifier is given by the following lemma. We will prove a stronger statement than required. Given a function $\tilde{A} : F^m \to F$, let $\widehat{A} : F^m \to F$ be the polynomial of total degree $d = m|H|$ that is closest to this function. Recall from Remark 5.4.2 that the polynomial $\widehat{A} : F^m \to F$ supposedly has the satisfying assignment embedded within it. Let $I \subset F^m$ be the set of locations in F^m that contains the assignment (i.e., $\widehat{A}|_I$ is supposedly the satisfying assignment).

Lemma 5.4.4 *Let α, δ_0 be the universal constants that appear in Theorem 5.3.2. For any integer $m, d > 0$, $\delta < \delta_0$, fields F, subset $H \subseteq F$ such that $d' = (6m+3)|H|$ and $|F| \geq \max\{\alpha d'^3, d'/(1-9\delta)\}$, the following holds. Suppose $\tilde{A}$ is not 2δ-close to any polynomial $\widehat{A}$ of total degree $d = m|H|$ such that $C(\widehat{A}|_I) = 1$. Then for any pair of proof oracles $P : F^{3m+3} \to F$ and $\Pi : F^{3m+3} \to F^{6m+6}$, with probability at least $\Omega(\delta)$, at least $\Omega(\delta)$ fraction of the input read by the* ROBUST-PCP–CIRCUIT-SAT *verifier needs to be modified in order to make the verifier accept.*

Proof: Let $\widehat{A} : F^m \to F$ and $p : F^m \to F$ be the polynomial of degree $d = m|H|$ and $d' = (6m+3)|H|$ that are closest to the functions $\tilde{A} : F^m \to F$ and $P : F^{3m+3} \to F$ respectively.

To analyze the expected robustness of the ROBUST-PCP–CIRCUIT-SAT verifier, we will analyze the expected robustness of each of the individual steps of the verifier. Since each of the individual steps reads only a fraction of the total input read by the verifier, we first calculate the fraction of the total input read by each of the individual steps. The total input read by the verifier comprises of the following: (a) Evaluation of $\tilde{A}$ along a random line by PCPP–LDT in Step. 2, (b) Evaluation of P and Π along a random line by ROBUST-PCPP–ZERO-ON-SUBCUBE in Step. 3. and (c) Evaluation of P along a random line $\mathcal{L}$ and evaluation of $\tilde{A}$ along 3 lines in Step. 5. Hence, the total input read by the verifier comprises of evaluations of several functions over seven lines. Thus, the fraction of the input read by each of the individual steps is as follows: (a) PCPP–LDT – $\frac{1}{7}^{th}$ of total input in

Step. 2, (b) ROBUST-PCPP–ZERO-ON-SUBCUBE – $\frac{2}{7}^{th}$ of the total input in Step. 3.and (c) $\frac{4}{7}^{th}$ of the total input in Step. 5.

By hypothesis, we have that $\widehat{A}|_I$ is not a satisfying assignment for the circuit C. Then one of the following three cases must hold.

Case I: $\widehat{A}$ is at least 2δ-far from $\tilde{A}$.

In this case by the soundness analysis of PCPP–LDT (Lemma 5.3.3), we have that on expectation at least δ fraction of the input read by Step. 2 must be changed in order to make this step accept. However, the fraction of input read by Step. 2 comprises only $\frac{1}{7}^{th}$ of the total input read by the verifier. Hence, on expectation at least $\delta/7$ fraction of the total input read by the verifier needs to be modified in order to make the verifier accept.

Case II: Either P is at least 2δ-far from p or p does not vanish on H^{3m+3}.

We have two further sub-cases corresponding to these two possibilities. (a) P is 2δ-far from p. (b) P is 2δ-close to p however $p|_{H^{3m+3}} \not\equiv 0$. In the latter sub-case, p must be the only polynomial that is 2δ-close to P since the distance between any two distinct polynomials of total degree d' is at least $1 - \frac{d'}{|F|}$ and we have $1 - \frac{d'}{|F|} \geq 4\delta$. In either sub-case, we have that P is at least 2δ-far from any polynomial that vanishes on H^{3m+3}. The soundness analysis of ROBUST-PCPP–ZERO-ON-SUBCUBE (Lemma 5.3.7) then comes into play. We have that on average at least $\delta/2$ fraction of the input read by Step. 3 needs to be modified in order to make this step accept. The fraction of input read by this step is $\frac{2}{7}^{th}$ that of the total input read by the verifier. Hence, on average at least $\delta/7$ fraction of the total input read by the verifier needs to be modified in order to make this step accept.

Case III: Both $\tilde{A}$ is 2δ-close to $\widehat{A}$ and P is 2δ-close to p and furthermore $p|_{H^{3m+3}} \equiv 0$.

Then, it must be the case that the polynomial p is not the polynomial $p_{(\widehat{A})}$ obtained by the polynomial transformation rule (5.7). Since, otherwise by Proposition 5.4.1 and Remark 5.4.2, $\widehat{A}|_I$ would be a satisfying assignment for C which contradicts the hypothesis. Hence, the polynomials p and $\widehat{A}$ violate Equation (5.7). By the Schwartz-Zippel Lemma 5.3.1, they should violate it this identity in at least $1 - \frac{d'}{|F|}$ fraction of the points in F^{3m+3}. Let B be the set of points in $x = (z_1, z_2, z_3, y_1, y_2, y_3) \in F^{3m} \times F^3 = F^{3m+3}$ where the Identity (5.7) is violated and where the following equalities hold: $P(x) = p(x), \tilde{A}(z_1) = \widehat{A}(z_1), \tilde{A}(z_2) = \widehat{A}(z_2)$, and $\tilde{A}(z_3) = \widehat{A}(z_3)$. The fractional size of B is at least $1 - \frac{d'}{|F|} - 8\delta$ since each of the functions $P, \tilde{A}$ is 2δ-close to the corresponding polynomials p and $\widehat{A}$. For the verifier to accept in Step. 4, either $P|_{\mathcal{L}}$ or $\tilde{A}|_{\mathcal{L}}$ must be changed for each of the points in the set $B \cap \mathcal{L}$. Since $\mathcal{L}$ is a random line, the expected size of $B \cap \mathcal{L}$ is at least $1 - \frac{d'}{|F|} - 8\delta \geq \delta$. Recall that the fraction of input read by this step is $\frac{4}{7}^{th}$ that of the total input read by the verifier. Hence, on average at least $4\delta/7 \geq \delta/7$ fraction of the total input read by the verifier needs to be modified in order to make the verifier accept.

Combining the three cases we have that on average at least $\delta/7$ fraction of the total input read by the ROBUST-PCP–CIRCUIT-SAT verifier needs to be modified in order to make the verifier accept. By an averaging argument, it follows that with probability $\delta/14$, at least $\delta/14$ fraction of the input read by the ROBUST-PCP–CIRCUIT-SAT verifier needs to be modified in order to make the verifier accept. This concludes the proof of the lemma. ∎

5.4.2 *Augmenting with the proximity test*

In this section, we modify the PCP for CIRCUIT SATISFIABILITY and construct a PCP of proximity for CIRCUIT VALUE while maintaining all the complexities. (Recall that the latter is stronger than the former, via Proposition 2.2.2.) We do so by adding a proximity test to the PCP–VERIFIER defined in Section 5.4.1. This new proximity test, as the name suggests, checks the closeness of the input to the satisfying assignment that is supposed to be encoded in the proof oracle (see Remark 5.4.2). This check is done by locally decoding a bit of the input from its encoding and comparing it with the actual input oracle.

Recall that a PCPP verifier is supposed to work as follows: The verifier is given explicit access to a circuit C with n gates on k input bits and oracle access to the input w in the form of an input oracle $W : [k] \to \{0,1\}$. The verifier should accept W with probability 1 if it is a satisfying assignment and reject it with high probability if it is δ-far from any satisfying assignment.

Recall that the function $\tilde{A} : F^m \to F$ is supposed to contain within it an assignment (See Remarks 5.4.2). Let $I \subseteq H^m \subset F^m$ be the set of locations in F^m that contain the assignment. The ROBUST-PCPP–CIRCUIT-VAL in addition to the tests of the ROBUST-PCP–CIRCUIT-SAT performs the ROBUST PROXIMITY TEST to check if the assignment given by $\tilde{A}|_I$ matches with the input oracle W. Specifically,

ROBUST-PCPP–CIRCUIT-VAL$_m^{W;\ \tilde{A},P,\Pi}(C,F)$.

1. Run ROBUST-PCP–CIRCUIT-SAT$_m^{\tilde{A},P,\Pi}(C,F)$ and reject if it rejects.
2. ROBUST PROXIMITY TEST

   Choose a random position $i \xleftarrow{R} \{1,\ldots,k\}$ in the input. Let $x \in I$ be the point corresponding to i in H^m. Choose a random line $\mathcal{L}$ through x. Query oracle $\tilde{A}$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}$ to $\mathcal{L}$ is not a polynomial of degree at most $d = m \cdot |H|$. Query the input oracle W at location i and reject if $W[i] \neq \tilde{A}(x)$.

Remark 5.4.5 *As in the case of the* ROBUST-PCP–CIRCUIT-SAT *verifier, we provide both the circuit C and the field F as input to the* ROBUST-PCPP–CIRCUIT-VAL *verifier. Later (in Sec 5.4.3), we will show*

how to construct a suitable sized field F from the size of the circuit and thus remove the field F from the list of inputs to the verifier.

Complexity of Verifier ROBUST-PCPP–CIRCUIT-VAL*:* In addition to the randomness required by the ROBUST-PCP–CIRCUIT-SAT verifier, this verifier requires randomness to choose a random point in the input and a random line in F^m. Thus, the total randomness is at most $O(m \log |F| + \log n)$. The verifier makes the following additional queries: 1 query to the input oracle W, which returns a bit and $|F|$ queries to the oracle $\tilde{A}$ which returns elements from F^{6m+6} (see Remark 5.4.3). Hence, the net query complexity is upper bounded by $O(m|F| \log |F|)$. It easily follows from Remark 5.4.2 that this verifier has perfect completeness.

Lemma 5.4.6 *Let α, δ_0 be the universal constants that appear in Theorem 5.3.2. For any integer $m, d' > 0$, proximity parameter $\delta \in (0, 1)$, fields F, subset $H \subseteq F$ such that $d' = (6m + 3)|H|$ and $|F| \geq \max\{\alpha d'^3, d'/(1 - 7\delta_0/2)\}$, the following holds. Suppose W is δ-far from satisfying the circuit C. Then for any proof oracle $\Gamma = (\tilde{A} : F^m \to F, P : F^{3m+3} \to F, \Pi : F^{3m+3} \to F^{6m+6})$, with probability at least $\Omega(\delta)$, either a constant (i.e., $\Omega(1)$) fraction of the entire input read in the proof oracle Γ or the entire input in the input oracle W (i.e, $W[i]$) needs to be changed in order to make the either* ROBUST-PCPP–CIRCUIT-VAL *verifier accept.*

Proof: If the input oracle W is δ-far from satisfying the circuit C, one of the following must happen.

Case I: $\tilde{A}$ is not δ_0-close to any polynomial $\widehat{A}$ of degree $d = m|H|$ such that $C(\widehat{A}|_I) = 1$.

Then by Lemma 5.4.4, we conclude that with probability at least $\Omega(\delta_0)$ (= constant), at least $\Omega(\delta_0)$ (= constant) fraction of the input read by ROBUST-PCP–CIRCUIT-SAT needs to be modified in order to make the verifier accept. Observe ROBUST-PCP–CIRCUIT-SAT verifier reads at least a constant fraction of the entire input read by the ROBUST-PCPP–CIRCUIT-VAL verifier. Hence, with constant probability, at least a constant fraction of the entire input read in the proof oracle Γ by the ROBUST-PCPP–CIRCUIT-VAL needs to be modified in order to make the verifier accept.

Case II: $\tilde{A}$ is δ_0-close to some polynomial $\widehat{A}$ of degree $d = m|H|$ such that $C(\widehat{A}|_I) = 1$.

Since W is δ-far from any satisfying assignment, the assignment given by $\widehat{A}|_I$ must be δ-far from W. With probability greater than δ over the choice of $i \in \{1, \ldots, k\}$ (and the corresponding point $x \in I$ in H^m), we have $W[i] \neq \widehat{A}(x)$. If this occurs, the only way to make the ROBUST PROXIMITY TEST accept is to either change $W[i]$ or change $\tilde{A}|_{\mathcal{L}}$ to a degree d polynomial other than $\widehat{A}|_{\mathcal{L}}$. Since $\mathcal{L}$ is a random line through x, every point on $\mathcal{L}$ other than x is a a uniformly random point in F^m. Recall that $\tilde{A}$ and $\widehat{A}$ are δ_0-close. By a Markov argument it follows that for every fixed value of x and a random line $\mathcal{L}$ through x, with probability at least $1 - 4\delta_0$, $\tilde{A}|_{\mathcal{L}\setminus\{x\}}$ and $\widehat{A}|_{\mathcal{L}\setminus\{x\}}$ are at least $1/4$-close. This implies that $\tilde{A}|_{\mathcal{L}}$ cannot be a polynomial of degree d other than $\widehat{A}|_{\mathcal{L}}$ (since two distinct polynomials agree in at most d points, and $(d-1)/|F| < 1/4$). Thus, with probability at least δ over the choice of i, we have $W[i] \neq \widehat{A}(x)$, and even after we fix such an i, we have with probability at least $1 - 4\delta_0$ that $\tilde{A}|_{\mathcal{L}\setminus\{x\}}$ and $\widehat{A}|_{call\setminus\{x\}}$ have distance at at most $1/4$. Hence, with probability at least $\delta(1 - 4\delta_0) = \Omega(\delta)$, either all of $W[i]$ or at least a constant fraction of the portion of the proof oracle read by the verifier needs to be modified in order to make the verifier accept.

Since we do not know which of the two cases occur, we can only guarantee the weaker of the two claims. Thus, with probability at least $\Omega(\delta)$, either a constant (i.e., $\Omega(1)$) fraction of the entire input read in the proof oracle Γ or the entire input in the input oracle W (i.e, $W[i]$) needs to be changed in order to make the either ROBUST-PCPP–CIRCUIT-VAL verifier accept. This concludes the proof of the lemma. ∎

5.4.3 *Converting to Binary Alphabet*

The soundness guarantee of the ROBUST-PCPP–CIRCUIT-VAL verifier (Lemma 5.4.6) showed that with constant probability, at least a constant fraction of the symbols read in the proof oracle or the entire portion of the input oracle read needs to be modified in order to make the verifier accept. Recall that the symbols in the proof oracle read by the verifier are from the alphabet F^{6m+6}. Hence, we have so far only proved robustness of the verifier over the large alphabet F^{6m+6}. However, we need to prove robustness over the binary alphabet. In other words, we need to show that if the symbols read by the verifier are bits (and not elements of some large alphabet like F^{6m+6}) even then the verifier has good robustness. We thus have the task of transforming the robust PCPP verifier over the alphabet $\Sigma = F^{6m+6}$ to one over the binary alphabet. This task is analogous to converting non-Boolean error correcting codes to Boolean ones via “code catenation”. This transformation is exactly the same transformation as the one in the proof of Lemma 2.4.5. However, we cannot directly use Lemma 2.4.5 as we may apply the “code concatenation” process only to the proof oracle Γ and not to the input oracle W. However, this is not a problem, because the input oracle is already binary and has good robustness.

Let ECC : $\{0,1\}^{\log|\Sigma|} \to \{0,1\}^b$ for $b = O(\log|\Sigma|)$ be a binary error-correcting code of constant relative minimum distance, which can be computed by an explicit circuit of size $O(\log|\Sigma|)$[Spi96]. We augment the original proof oracle Γ, viewed now as having $\log|\Sigma|$-bit long entries (i.e., elements of Σ) with an additional oracle Υ having b-bit long entries, where $\Upsilon(x)$ is supposed to be $\mathrm{ECC}(\Gamma(x))$.

Our new verifier V, on oracle access to the input W and proof $\Gamma \circ \Upsilon$, will simulate the ROBUST-

PCPP–CIRCUIT-VAL. The queries to the input oracle are performed just as before however, for each query $x \in F^m$ in the proof oracle Γ made by ROBUST-PCPP–CIRCUIT-VAL, V will query the corresponding $\log|\Sigma|$ bits in $\Gamma(x)$ and the b bits in $\Upsilon(x)$. Thus, the query complexity of V is at most $\log|\Sigma| + b$ times the number of queries issued by the earlier verifier. Since $b = O(\log|\Sigma|)$, the query complexity of the new verifier V is a constant times that of the previous one. The randomness is exactly the same. The action of the new verifier V is as follows: Suppose ROBUST-PCPP–CIRCUIT-VAL issues queries $x_1, \ldots, x_{q_1}$ to the proof oracle Γ, and queries $i_1, \ldots, i_{q_2}$ to the input oracle, then V issues queries $x_1, \ldots, x_{q_1}$ to the proof oracle Γ, a similar set of queries $x_1, \ldots, x_{q_1}$ to the proof oracle Υ and $i_1, \ldots, i_{q_2}$ to the input oracle. V accepts $(\Gamma(x_1), \ldots, \Gamma(x_{q_1}), \Upsilon(x_1), \ldots, \Upsilon(x_{q_1}), W(i_1), \ldots$ $dots, W(i_{q_2}))$ iff the ROBUST-PCPP–CIRCUIT-VAL accepts $(\Gamma(x_1), \ldots, \Gamma(x_{q_1}), W(i_1), \ldots, W(i_{q_2}))$ and $\Upsilon(x_i) = \mathrm{ECC}(\Gamma(x_i))$ for all $i = 1, \ldots, q_1$. It is straightforward to check that V has perfect completeness if ROBUST-PCPP–CIRCUIT-VAL has perfect completeness. The robust soundness of the verifier is given by the following lemma.

Lemma 5.4.7 *Let α, δ_0 be the universal constants that appear in Theorem 5.3.2. For any integer $m, d' > 0$, proximity parameter $\delta \in (0,1)$, fields F, subset $H \subseteq F$ such that $d' = (6m+3)|H|$ and $|F| \geq \max\{\alpha d'^3, d'/(1 - 7\delta_0/2)\}$, the following holds. Suppose W is δ-far from satisfying the circuit C. Then for any proof oracle $\Gamma \circ \Upsilon$, with probability at least $\Omega(\delta)$, either a constant (i.e., $\Omega(1)$) fraction of the entire input read in the proof oracle $\Gamma \circ \Upsilon$ or the entire input in the input oracle W (i.e, $W[i]$) needs to be changed in order to make the either verifier V accept.*

It is to be noted that the robustness of the proof oracle is not the same as similar parameters in Lemmas 5.4.7, but weaker by a constant factor as suggested in Lemma 2.4.5.

Finally, we conclude by proving Theorem 5.1.2.

Proof (of Theorem 5.1.2): Theorem 5.1.2 is proved using the verifier V described in the earlier paragraph. Recall that V is the verifier obtained by transforming the ROBUST-PCPP–CIRCUIT-VAL verifier to a binary alphabet. We set $m = \log n / \log\log n$ and choose F to be a field of size $\alpha d'^3$. For sufficiently large n, this setting of parameters satisfies all the requirements of Lemma 5.4.7. Also for this value of m, we have that $n^{1/m} = \log n$. Recall that $|H| = n^{1/m}, d = m|H|$ and $d' = (6m+3)|H|$. Hence, $d = mn^{1/m} = O(\log^2 n), d' = O(mn^{1/m}) = O(\log^2 n)$ and $|F| = \mathrm{poly}\log n$. The randomness complexity of the ROBUST-PCPP–CIRCUIT-VAL verifier (i.e., before the transformation to the binary alphabet) is then $O(m \log|F|) = O(\log n)$ while the query complexity is $O(m|F|\log|F|) = \mathrm{poly}\log n$. The decision complexity of this verifier is also $\mathrm{poly}\log n$. As mentioned in the earlier paragraph, the transformation from the alphabet Σ to the binary alphabet maintains the randomness complexity while the query (and decision) complexity increase by at most a constant factor. Hence, the randomness, query and decision complexities of the verifier are as before but with weaker constants.

We now have to prove the soundness guarantee of this verifier. So far, we have considered the proof and input oracle separately. Hence the robustness in Lemma 5.4.7 were expressed separately for the proof and input oracles. We can consider them together by giving equal weights to the two oracle portions in the decision circuits. This weighting may increase the query (and decision) complexity increase by at most a factor of 2, and has no affect on any other parameter. This weighting results in the following soundness guarantee. If W is δ-far from any satisfying assignment, then with probability at least $\Omega(\delta)$, at least a constant fraction of the entire input read by the verifier needs to be modified in order to make the verifier accept.

We have thus obtained a robust PCPP verifier for CKTVAL for all proximity parameters $\delta \in (0,1)$ with randomness complexity $O(\log n)$, query (and decision complexity) $\mathrm{poly}\log n$ and robust soundness error $1-\Omega(\delta)$ with $\Omega(1)$ (i.e., constant) robustness parameter. This proves Theorem 5.1.2. ∎

Part II

Short PCPs

CHAPTER 6

Introduction

6.1 Introduction - Main Construct

In this part, we prove our main construct, which is the technical meat of this thesis. This construct forms the basic building block for our composition. The main theorems of this thesis (Theorem 1.3.2 and 1.3.3) are proved by composing the main construct with itself several times.

Throughout this part, n denotes the length of the explicit input given to the PCPP verifier, which in case of CIRCUIT VALUE is defined as the size of the circuit (given as explicit input). As stated before, our main results rely on the following highly efficient robust PCP of proximity.

Theorem 6.1.1 (Main Construct) *There exists a universal constant c such for all $n, m \in \mathbb{Z}^+$, $0 < \delta, \gamma < 1/2$ satisfying $n^{1/m} \geq m^{cm}/(\gamma\delta)^3$ and $\delta \leq \gamma/c$,* CIRCUIT VALUE *has a robust PCP of proximity (for circuits of size n) with the following parameters*

- *randomness* $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$,
- *decision complexity* $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$, *which also upper-bounds the query complexity.*[1]
- *perfect completeness, and*
- *for proximity parameter δ, the verifier has robust-soundness error γ with robustness parameter $(1 - \gamma)\delta$.*

[1] In fact, we will upper-bound the query complexity by $q = n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ and show that the verifier's decision can be implemented by a circuit of size $\widetilde{O}(q)$, which can also be bounded by $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ with a slightly larger unspecified polynomial.

We comment that the condition $\delta < \gamma/c$ merely means that we present robust PCPs of proximity only for the more difficult cases (when δ is small).

The main point to be noted is that the above construct is extremely efficient in randomness. For instance, suppose we set $m = 2$ and δ, γ suitably to satisfy the hypothesis of the theorem. Then, Theorem 6.1.1 states that CKTVAL has a robust PCPP with randomness complexity $\log n/2 + O(\log\log n)$ and query complexity $\sqrt{n}\cdot\text{poly}\log n$. The amount of randomness ($\approx \log n/2$) is less than the number of bits required to even index an element in the proof ($\approx \log n$)! The query complexity is not constant, in fact it is approximately $\sqrt{n}$. However, we can use composition to reduce the query complexity.

6.1.1 Outline of this Part

This part is devoted to proving Theorem 6.1.1 and showing how this implies the main theorems of the thesis (Theorem 1.3.1, 1.3.2 and 1.3.3). We begin this part by explaining some of the salient points in our construction which helps us save on randomness (Section 6.2). The construction is very technical, so we provide an overview of the construction in Section 6.3. The actual construction itself is presented in Chapters 7–9. We start by presenting a (highly efficient) ordinary PCP (establishing Theorem 1.3.1), which lends itself to the subsequent modifications. In Chapter 8, we augment this PCP with a test of proximity, deriving an analogous PCP of proximity. In Chapter 9 we present a robust version of the PCP of proximity derived in the previous sections. In Chapter 10, we show how to derive Theorems 1.3.2 and 1.3.3, by composing this Robust PCP of proximity with itself multiple times.

6.2 Saving on Randomness

In this section, we highlight the salient points in the construction of our basic building block (proof of Theorem 6.1.1) which helps us save on randomness. For this purpose, we mention some of the intermediate steps in our construction, explain how earlier PCP constructions handled these steps and how our construction improves on the amount of randomness being used in these steps. The main steps where we save on randomness can be broadly classified into the following five items.

1. Reduction to Algebraic Problem

2. Randomness for composition

3. Bundling without adding a variable

4. Derandomized Low-Degree Test

5. Overcoming the $2^{\sqrt{\log n}}$ barrier via PCPs of Proximity

It is to be noted that some of these improvements (not all) were present in earlier constructions of short PCPs.

Before looking at each of these items, it is useful to have a bird's eyeview of a typical algebraic construction of a PCP. Most algebraic constructions of PCPs (which includes all constructions with the exception of that of Dinur and Reingold) proceed along the following lines: In the first step, CIRCUIT SATISFIABILITY is reduced to some algebraic problem (for instance, Polynomial Constraint Satisfaction), which is easily amenable to PCP constructions. In this algebraic problem, the satisfying assignment A which is the standard $\mathbf{NP}$-proof for CKTSAT is encoded by a low-degree polynomial $\widehat{A}$. Instead of the $\mathbf{NP}$-proof A, the prover provides as proof the table of evaluations of this polynomial $\widehat{A}$. To verify that the low-degree polynomial $\widehat{A}$ is in fact an encoding of a satisfying assignment, the prover provides another low-degree polynomial P (also as a table of evaluations) which can be computed using some local computation rules given the polynomial $\widehat{A}$. The polynomial P satisfies the property that it is zero on some sub-cube if and only if it is obtained from a polynomial $\widehat{A}$ that encodes a satisfying assignment. Thus, the PCP verifier needs to make the following checks:

1. check that the functions $\widehat{A}$ and P (both functions provided to it as tables of evaluations) are close to low-degree polynomials,

2. check that the polynomial P is obtained consistently from $\widehat{A}$ (i.e., P is in fact the polynomial that is obtained from $\widehat{A}$ by applying the above mentioned local computation rules),

3. and finally, check that P is zero on the required sub-cube.

We can now describe the steps which save on randomness.

Reduction to Algebraic Problem: The first blow-up in the proof size is due to the reduction from CIRCUIT SATISFIABILITY to the algebraic problem. For instance, the reduction described in the proof of the PCP Theorem (Section 5.4.1 of Chapter 5) incurs a n^6 blowup in the proof size. We cannot afford such an expensive reduction. To obtain nearly linear-sized PCPs, we require reductions that blow up the instance size by a constant factor or at most a polylogarithmic factor. For this purpose, all constructions of short PCPs, beginning with the work of Polishchuk and Spielman [PS94], reduce CKTSAT to a coloring problem on de-Bruijn graph, which has a short-sized algebraic description. We also adopt this approach, with slight modifications to satisfy our requirements (Sections 7.2 and 7.3).

Randomness for composition: Another step where randomness was incurred in earlier PCP constructions is in the composition of the main construct with itself. As explained in Section 3.1.1, composition was performed earlier by "parallelizing" the outer PCP verifier. We cannot afford the cost of any known parallelization procedure because at the very least these procedures increase the length of the proof by a factor related to the answer size of the outer PCP verifier. Our composition completely avoids this parallelization procedure (see Section 3.2). Composition of robust PCPPs is not only simpler than earlier compositions but also helps in saving randomness during the composition step. In fact, no extra randomness is required for the new composition. The randomness of the composed verifier is precisely the sum of the randomness of the outer and inner verifiers.

Bundling without adding a variable: The bird's eyeview description given earlier is an over-simplification of most PCP constructions. The proof typically involves the evaluations of several low-degree polynomials, not just the two polynomials $\widehat{A}$ and P mentioned. In fact, most proofs contain a super-constant number of such low-degree polynomials. Suppose there are k such functions $p_1, \dots, p_k$. Step. 1 of the verifier checks that each of these functions is close to being low-degree. However, we cannot perform these low-degree checks for each of the functions individually since this would result in a drop in the robustness by a factor of k. For this purpose, we bundle the several functions $p_1, \dots, p_k$ together. Most earlier PCP constructions performed this bundling by introducing a new variable as follows: Let $p_1, \dots, p_k : F^m \to F$ be the k functions on m variables. A new function $q : F^{m+1} \to F$ on $m+1$ variables is then defined which bundles together all the k functions as follows: For $i = 1, \dots, k$, $f(x_1, \dots, x_m, i) = p_i(x_1, \dots, x_m)$. Thus, checking if the functions $p_1, \dots, p_k$ are low-degree is equivalent to checking if q is low-degree. The new variable y behaves as index to the functions $p_1, \dots, p_k$. Note, we are only interested in the values of q when y is in the set $\{1, \dots, k\}$. However, introducing the new variable y defines the function q for all values of y in the field F. Thus, this bundling procedure increases the proof-size by a factor of $|F|/k$. This blowup is too expensive for the parameters we seek. So we instead perform a "combinatorial bundling" instead of the standard algebraic bundling indicated above (by adding a variable). Our combinatorial bundling does not increase the proof-size at all (See Sections 5.4.1 and 9.3 for actual details of the combinatorial bundling procedure).

To understand how randomness is saved in the next two steps, we will focus on the first of the three checks performed by the verifier – the task of low-degree testing. For this purpose, we will go into the details of this test. The task of low-degree testing is typically performed by what is popularly called the LINE–POINT–TEST test. The LINE–POINT–TEST test works as follows (for more details, see Appendix A): The LINE–POINT–TEST checks if a given function $f : F^m \to F$ is close to some polynomial of degree at most d with the help of another oracle h. The oracle h returns for each line in F^m a univariate polynomial of degree d. For line $\mathcal{L}$, the polynomial $h(\mathcal{L})$ is supposedly the restriction of the function f to the line $\mathcal{L}$.

LINE–POINT–TEST

Input: A function $f : F^m \to F$ and lines-oracle $h : \{\text{lines in } F^m\} \to \{\text{univariate degree } d \text{ polynomials}\}$ where F is a finite field.

Test: Choose a random line $\mathcal{L}$ in F^m and a random point x on the line $\mathcal{L}$. Query the function f at the point x and the lines-oracle for the line $\mathcal{L}$. Accept if the polynomial $h(\mathcal{L})$ agrees with $f(x)$.

There are other variations of the LINE–POINT–TEST, e.g., using planes instead of lines (plane-point test of Raz and Safra [RS96]), random axis-parallel lines instead of random lines (low-degree tests of [BFLS91, AS98, PS94])

Derandomized Low-Degree Tests: The LINE–POINT–TEST in its above form is expensive in randomness for the following reason: Choosing a random line in F^m requires choosing two random points in F^m and this costs $2m \log |F|$ bits of randomness. On the other hand, the size of the input function f is only F^m. Hence, the LINE–POINT–TEST results in a quadratic blowup in the proof-size. In fact, the PCPs of Harsha and Sudan [HS00] were of nearly cubic size since they employed the plane-point test. However, luckily for us, there have been several improvements in the LINE–POINT–TEST. Ben-Sasson *et al.* [BSVW03] following the work of Goldreich and Sudan [GS02] observed that it is sufficient if the lines are chosen from a derandomized set of size approximately $F^{m(1+o(1)}$ (For further details, see Appendix A). This derandomized low-degree tests uses only $m(1 + o(1)) \log |F|$ bits of randomness compared to the $2m \log |F|$ bits needed earlier. This incurs only a nearly linear blowup in the proof-size. Both our PCPs and the short PCPs of [BSVW03] are constructed using these derandomized low-degree tests.

Overcoming the $2^{\sqrt{\log n}}$ barrier via PCPs of Proximity: This step is our main improvement over all earlier constructions of short PCPs. We show below that all PCP constructions (including that of [BSVW03]) which use the above-mentined LINE–POINT–TEST (either in its original form or in its derandomized form) incur a blowup factor of at least $2^{\sqrt{\log n}}$ in the proof-size. We then explain how this barrier of $2^{\sqrt{\log n}}$ can be overcome if we use PCPs of proximity instead of PCPs.

We first note that even before the LINE–POINT–TEST can be applied, we need to encode the input into a polynomial of the form $f : F^m \to F$ using a Reed-Muller encoding. A Reed-Muller encoding involves a blow-up factor of at least m^m in the size of the proof. Thus, if the original input is of size n, then the size of the new encoding (which is of size $|F|^m$, since it is a table of evaluations) is at least $m^m \cdot n$ (i.e., $|F|^m \geq m^m \cdot n$). Now consider the LINE–POINT–TEST. It first chooses a random line and then a random point on the line. Thus, the total randomness required for LINE–POINT–TEST is at least $R = \log(\text{Number of random lines in } F^m) + \log |F|$. The number of random lines in F^m (even for the derandomized LINE–POINT–TEST) is at least $|F|^m$. Hence, the total randomness required by the LINE–POINT–TEST is at least $R = (m+1)\log|F|$. Hence, the blowup in the proof size incurred by the LINE–POINT–TEST is at least $2^R/|F|^m$ since the old-proof size is $|F|^m$ and the new proof is at least 2^R. Observe that $2^R/|F|^m = |F|$ which is at least $m\cdot n^{1/m}$ (since $|F|^m \approx m^m \cdot n$). Hence, the overall blowup in proof-size incurred by the Reed-Muller encoding followed by the LINE–POINT–TEST is at least $m^m \times m \cdot n^{1/m} \geq m^m \cdot n^{1/m}$. However, $m^m \cdot n^{1/m} \geq 2^{O(\sqrt{\log n})}$ for all values of m ($m^m \cdot n^{1/m}$ is minimized when $m \approx \sqrt{\log n}$ in which case $m^m \cdot n^{1/m} \approx 2^{O(\sqrt{\log n})}$). Hence, any use of the LINE–POINT–TEST (derandomized or otherwise) incurs a blowup of at least $2^{\sqrt{\log n}}$ in the proof-size. It is to be noted that the PCPs of [BSVW03] exactly match this lower-bound.

We now demonstrate how we can overcome this barrier of $2^{\sqrt{\log n}}$ using PCPs of proximity instead of PCPs. Recall that PCPs of proximity are designed to check that a given string satisfies a property by merely probing the string and an additional proof at a very few locations. The main purpose of the LINE–POINT–TEST is to check that the restriction of the function $f : F^m \to F$ to most random lines is a univariate polynomial of degree at most d. We can now do this check directly using PCPs of proximity without the help of any additional lines oracle as follows:

LINE–POINT–TEST--VIA-PCPPS

Input: A function $f : F^m \to F$ where F is a finite field.

Test: Choose a random line $\mathcal{L}$ in F^m. Query the function f at each point x on the line $\mathcal{L}$ and check that the restriction of f to the line $\mathcal{L}$ is a univariate polynomial of degree d.

It is not immediate where PCPPs play a role in the above test. In fact, PCPPs do not occur in the description of the LINE–POINT–TEST--VIA-PCPPS at all. They are used when composing the above verifier as follows: If the LINE–POINT–TEST--VIA-PCPPS is used as an outer verifier, then one can compose it with a inner PCPP verifier that checks if the string $f|_{\mathcal{L}}$ is close to a univariate polynomial of degree d. Note this cannot be performed if we are composing using PCPs instead of PCPs of proximity.

Let us now analyse the blowup in proof-size. As before, the Reed-Muller enoding involves a blowup factor of at least m^m. For the LINE–POINT–TEST--VIA-PCPPS the only randomness used is to choose a random line, which if we are using the derandomized lines of [BSVW03], is at most $R' = m(1+o(1))\log|F|$. Thus, we do not incur an extra $\log|F|$ term in the randomness as before. Hence, the blowup due to the LINE–POINT–TEST--VIA-PCPPS is at most $2^{R'}/|F|^m = |F|^{o(m)}$. Hence, the overall blowup in proof-size is $m^m \times |F|^{o(m)} \approx m^m \cdot n^{o(1)} \approx m^m$. Unlike in the earlier case, there is no inherent lower bound (m^m can be as small as possible as opposed to $m^m \cdot n^{1/m}$ which is lower bounded by $2^{\sqrt{\log n}}$). In fact, we do construct PCPs with blowup factors of at most $2^{(\log n)^{\varepsilon}}$ (for constant query complexity) and $2^{o(\log\log n)^2}$ (for $o(\log\log n)$ query complexity) both of which are significantly smaller than $2^{\sqrt{\log n}}$. Though there is no inherent lower-bound on the proof-size as in the earlier case, we are not able to construct PCPs with blowup less than $2^{(\log n)^{\varepsilon}}$ (for constant query complexity) and $2^{o(\log \log n)^2}$ (for $o(\log \log n)$ query complexity) due to certain technical limitations of our construction.

6.3 *Overview of Main Construct*

Following is an overview of the proof of Theorem 6.1.1; the actual proof of this theorem is given in the subsequent three chapters.

Theorem 6.1.1 is proved by modifying a construction that establishes Theorem 1.3.1. We follow [HS00] and modify their construction. (An alternative approach would be to start from [PS94], but that construction does not seem amenable to achieving robust soundness.) The construction of [HS00] may be abstracted as follows: To verify the satisfiability of a circuit of size n, a verifier expects oracles $\mathrm{F}_i : F^m \to F$, $i \in \{1 \dots, t = \mathrm{poly}\log n\}$, where F is a field and m is a parameter such that $F^m \approx m^m \cdot n$. The verifier then needs to test that (1) each of the F_i's is close to a m-variate polynomial of $\underline{\text{low degree}}$ and (2) the polynomials satisfy some $\underline{\text{consistency}}$ properties which verify that F_i is locally consistent with F_{i-1}.[2] (These consistency checks include tests which depend on the input circuit and verify that F_i's actually encode a satisfying assignment to the circuit.)

We work within this framework — namely our verifier will also try to access oracles for F_i's and test low-degreeness and consistency. Our key modification to this construction is a randomness-reduction in the low-degree test obtained by using the small collection of (small-biased) lines of [BSVW03], while using only the "canonical" representations of these lines (and avoiding any complication that was introduced towards "proof composition"). In particular, unlike in [HS00, GS02, BSVW03], we cannot afford to pack the polynomials $\mathrm{F}_1, \dots, \mathrm{F}_t$ into a single polynomial (by using an auxiliary variable that blows-up the proof length by a factor of the size of the field in use). Instead, we just maintain all these t polynomials separately and test them separately to obtain Theorem 1.3.1. (In the traditional framework of parallelized PCPs, this would give an unaffordable increase in the number of (non-Boolean) queries. However, we will later ameliorate this loss by a "bundling technique" that will yield robust-soundness.)

The resulting PCP is converted into a PCP of proximity by comparing the input-oracle (i.e. supposed satisfying assignment to the circuit) to the proof-oracle (which is supposed to include an encoding of the said assignment). That is, we read a random location of the input and the corresponding location of the proof oracle, and test for equality. Actually, these locations of the proof-oracle must be accessed via a self-correction mechanism (rather than merely probing at the

[2]Strictly speaking, the consistency checks are a little more complicated, with the functions really being indexed by two subscripts and consistency tests being between $\mathrm{F}_{i,j}$ and $\mathrm{F}_{i,j-1}$, as well as between $\mathrm{F}_{i,0}$ and $\mathrm{F}_{i+1,0}$. However, these differences don't alter our task significantly — we ignore them in this section to simplify our notation.

desired points of comparison), since they constitute only a small part of the proof oracle (and thus corruptions there may not be detected). (This technique was already suggested in [BFLS91].)

The most complex and subtle part of the proof of Theorem 6.1.1 is establishing the robust-soundness property. We sketch how we do this below, first dealing with the low-degree test and the consistency tests separately, and then showing how to reconcile the two "different" fixes.

Low-degree tests of $\mathrm{F}_1, \ldots, \mathrm{F}_t$***:*** Selecting a random line $\ell : F \to F^m$ (from the aforementioned sample space), we can check that (for each i) the restriction of F_i to the line ℓ (i.e., the function $f_i(j) \triangleq \mathrm{F}_i(\ell(j))$) is a low-degree (univariate) polynomial. Each of these tests is *individually robust*; that is, if F_i is far from being a low-degree polynomial then with high probability the restriction of F_i to a random line ℓ (in the sample space) is far from being a low-degree polynomial. The problem is that the conjunction of the t tests is not sufficiently robust; that is, if one of the F_i's is δ-far from being a low-degree polynomial then it is only guaranteed that the sequence of t restrictions (i.e., the sequence of the f_i's) is (δ/t)-far from being a sequence of t low-degree (univariate) polynomials. Thus robustness decreases by a factor of t, which we cannot afford for non constant t.h

Our solution is to observe that we can "bundle" the t functions together into a function $\overline{\mathrm{F}} : F^m \to F^t$ such that if one of the F_i's is far from being a low-degree polynomial then the restriction of $\overline{F}$ to a random line will be far from being a bundling of t low-degree univariate polynomials. Specifically, for every $x \in F^m$, define $\overline{\mathrm{F}}(x) \triangleq (\mathrm{F}_1(x), ..., \mathrm{F}_t(x))$. To test that $\overline{\mathrm{F}}$ is a bundling of low-degree polynomials, select a random line ℓ (as above), and check that $\overline{f}_\ell(j) = \overline{\mathrm{F}}(\ell(j))$ is a bundling of low-degree univariate polynomials. Thus, we establish *robustness at the bundle level*; that is, if one of the F_i's is far from being low degree then, with high probability, one must modify $\overline{f}_\ell$ on a constant fraction of values in order to make the test accept. The point is that this robustness refers to Hamming distance over the alphabet F^t, rather than alphabet F as before. We can afford this increase in alphabet size, as we later encode the values of $\overline{\mathrm{F}}$ using an error-correcting code in order to derive *robustness at the bit level*.

We wish to highlight a key point that makes the above approach work: when we look at the values of $\overline{\mathrm{F}}$ restricted to a random line, we get the values of the individual F_i's restricted to some random line, which is exactly what a low-degree test of each F_i needs. This fact is not very surprising, given that we are subjecting all F_i's to the same test. *But what happens when we need to make two different types of tests?* This question is not academic and does come up in the consistency tests.

Consistency tests: To bundle the t consistency tests between F_i and F_{i+1} we need to look into the structure of these tests. We note that for every i, a random test essentially refers to the values of F_i and F_{i+1} on (random) i-th axis-parallel lines. That is, for every i, and a random $x' = (x_1, ..., x_{i-1}) \in F^{i-1}$ and $x'' = (x_{i+1}, ..., x_m) \in F^{m-i}$, we need to check some relation between

$\mathrm{F}_i(x', \cdot, x'')$ and $\mathrm{F}_{i+1}(x', \cdot, x'')$.[3] Clearly, querying $\overline{\mathrm{F}}$ as above on the ith axis-parallel line, we can obtain the relevant values from $\overline{\mathrm{F}}(x', \cdot, x'')$, but this works only for one specific value of i, and other values of i will require us to make other queries. The end result would be that we'll gain nothing from the bundling (i.e., from $\overline{\mathrm{F}}$) over using the individual F_i's, which yields a factor of t loss in the robustness.[4] Fortunately, a different bundling works in this case.

Consider $\overline{\mathrm{F}}'$ such that $\overline{\mathrm{F}}'(x) \triangleq (\mathrm{F}_1(x), \mathrm{F}_(2)(S(x)), ..., \mathrm{F}_t(S^{t-1}(x)))$, for every $x \in F^m$, where S denotes a (right) cyclic-shift (i.e., $S(x_1, ..., x_m) = (x_m, x_1 \ldots, x_{m-1})$ and $S^i(x_1, ..., x_m) = (x_{m-(i-1)}, \ldots, x_m, x_1, x_2, \ldots, x_{m-i})$). Now, if we ask for the value of $\overline{\mathrm{F}}'$ on the first and last axis-parallel lines (i.e., on $(\cdot, x_2, ..., x_m)$ and $(x_2, ..., x_m, \cdot) = S^{-1}(\cdot, x_2, ..., x_m)$), then we get all we need for all the m tests. Specifically, for every i, the i-th component in the bundled function $\overline{\mathrm{F}}'(\cdot, x_2, ..., x_m)$ is $\mathrm{F}_i(S^{i-1}(\cdot, x_2, ..., x_m)) = \mathrm{F}_i(x_{m-i+2}, ..., x_m, \cdot, x_2, ..., x_{m-i+1})$, whereas the $(i+1)$-st component in $\overline{\mathrm{F}}'(S^{-1}(\cdot, x_2, ..., x_m))$ is $\mathrm{F}_{i+1}(S^i(S^{-1}(\cdot, x_2 \ldots, x_m))) = \mathrm{F}_{i+1}(x_{m-i+2}, ..., x_m, \cdot, x_2, ..., x_{m-i+1})$. Thus, we need only to query two bundles (rather than t), and robustness only drops by a constant factor.

Reconciling the two bundlings: But what happens with the low-degree tests that we need to do (which were "served" nicely by the original bundling $\overline{\mathrm{F}}$)? Note that we cannot use both $\overline{\mathrm{F}}$ and $\overline{\mathrm{F}}'$, because this will requires testing consistency between them, which will introduce new problems as well as a cost in randomness that we cannot afford. Fortunately, the new bundling (i.e., $\overline{\mathrm{F}}'$), designed to serve the axis-parallel line comparisons, can also serve the low-degree tests. Indeed, the various F_i's will not be inspected on the same lines, but this does not matter, because the property of being a low-degree polynomial is preserved when "shifted" (under S).

Tightening the gap between robustness and proximity: The above description suffices for deriving a weaker version of Theorem 6.1.1 in which the robustness is only (say) $\delta/3$ rather than $(1-\gamma)\delta$ for a parameter γ that may be set as low as $1/\mathrm{poly}(\log n)$. Such a weaker result yields a weaker version of Theorem 10.2.1 in which the query complexity is exponentially larger (e.g., for proof-length $\exp(o(\log\log n)^2) \cdot n$, we would have obtained query complexity $\exp(o(\log\log n)) = \log^{o(1)} n$ rather than $o(\log\log n)$); see comment at the end of Section 10. To obtain the stronger bound on the robustness parameter, we take a closer look at the conjunction of the standard PCP test and the proximity test. The PCP test can be shown to have constant robustness $c > 0$, whereas the proximity test can be shown to have robustness $\delta' \triangleq (1-\gamma))\delta$. When combining the two tests, we obtain robustness equal to $\min(\alpha c, (1-\alpha)\delta')$, where α is the relative length of queries used in the PCP test (as a fraction of the total number of queries). A natural choice, which yields the weaker

[3]Again, this is an oversimplification, but suffices to convey the main idea of our solution.

[4]It turns out that for constant m (e.g., $m = 2$) this does not pose a problem. However, a constant m would suffice only for proving a slightly weaker version of Theorem 1.3.2 (where $o(\log\log n)$ is replaced by $\log\log n$). but not for proving Theorem 1.3.3, which requires setting $m = \log^{\varepsilon} n$, for constant $\varepsilon > 0$.

result, is to weight the queries (or replicate the smaller part) so that $\alpha = 1/2$. (This yields robustness of approximately $\min(c, \delta')/2$.) In order to obtain the stronger bound, we assign weights such that $\alpha = \gamma$, and obtain robustness $\min(\gamma c, (1-\gamma)\delta') > \min(\Omega(\gamma), (1-2\gamma)\delta)$, which simplifies to $(1-2\gamma)\delta$ for $\delta < \gamma/O(1)$. (The above description avoids the fact that the PCP test has constant soundness error, but the soundness error can be decreased to γ by using sequential repetitions while paying a minor cost in randomness and *while approximately preserving the robustness*. We comment that the proximity test, as is, has soundness error γ.)

CHAPTER 7

A randomness-efficient PCP

7.1 *Introduction*

In this chapter, we present a vanilla version (Theorem 7.1.1) of Theorem 6.1.1. More specifically, we construct a regular PCP for CIRCUIT SATISFIABILITY (i.e., a robust PCP of proximity without either the robustness or proximity properties). This construction favors over earlier PCP constructions in the fact that it is very efficient in randomness. As mentioned earlier, this theorem suffices to prove Theorem 1.3.1.

Theorem 7.1.1 *There exists a universal constant $0 < \varepsilon < 1$ such that the following holds. Suppose $m \in \mathbb{Z}^+$ satisfies $m \leq \log n/\text{loglog}n$ Then there exists a PCP for* CIRCUIT SATISFIABILITY *(for circuits of size n) with the following parameters*

- *randomness $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n)$,*
- *query complexity $q = O(m^2 n^{1/m} \log^2 n)$ and decision complexity $\widetilde{O}(q)$,*
- *perfect completeness,*
- *and soundness error $1 - \varepsilon$.*

The construction of the PCP for CIRCUIT SATISFIABILITY proceeds in three steps. First, we transform the input circuit φ to a well-structured circuit φ' along the lines of Polishchuk and Spielman [PS94, Spi95] (Section 7.2). φ' is only slightly larger than φ, but has an algebraic structure that will be crucial to our verification process. Any legal assignment to the gates of φ (i.e. one that preserves the functionality of the gates of φ) can be transformed to a legal assignment to φ'.

The important property of φ' is the following: If we encode an assignment to the gates of φ' using a specific sequence of Reed-Muller codewords (i.e. low degree polynomials), then the legality of the assignment can be locally verified (by reading a small random portion of the encoding). The encoding via low degree polynomials (and resulting local tests) is as in Harsha and Sudan [HS00] and is described in Section 7.3. Thus, our PCP verifier will essentially test *(i)* that the encoding of the purported satisfying assignment to φ' is formed of low degree polynomials, (this part will be done using the randomness-efficient low degree test of Ben Sasson *et al.* [BSVW03]); and *(ii)* that the assignment is legal. Section 7.4 describes the construction of the PCP verifier and Section 7.5 analyzes its properties. Most of the above results are implicit in the literature, but carefully abstracting the results and putting them together helps us in significantly reducing the randomness of the PCP verifier.

7.2 Well-structured Boolean circuits

The main problem with designing a randomness-efficient PCP verifier directly for CIRCUIT SATISFIABILITY is that we need to encode the assignment to all gates of the input circuit using certain Reed-Muller based codes, in such a way that will allow us to locally verify the legality of all gates of the circuit, using only the encoded assignment. In order to do this, we require the circuit to have a well-behaved structure (amenable to our specific encoding and verification demands). Of course, an arbitrary circuit does not necessarily have this structure, but luckily we have the technology to overcome this. More to the point, we can restructure any circuit into a well-behaved circuit that will suit our needs. The natural encoding (used e.g. in the Hadamard based PCP, Section 4) incurs a quadratic blowup in size. To get over this problem, Polishchuk and Spielman [PS94, Spi95] introduced a different, more efficient restructuring process that embeds the input circuit into well-structured graphs known as de Bruijn graphs. Indeed, the blowup in circuit size using these circuits is merely by a logarithmic multiplicative factor, and their usefulness for the local verification of legal assignments will become evident later (in Section 7.3). As in Polishchuk and Spielman [PS94, Spi95], we embed the input circuit into wrapped de Bruijn graphs (see Definition 7.2.1). We use a slightly different definition of de Bruijn graphs, more convenient for our purposes, than that used in [PS94, Spi95]. However it can easily be checked that these two definitions yield isomorphic graphs. The main advantage with the de Bruijn graphs is that the neighborhood relations can be expressed very easily using simple bit-operations like cyclic-shifts and bit-flips. In [PS94, Spi95] the vertex set of these graphs is *identified* with a vector space. We instead work with a strict embedding of these graphs in a vector space where the vertices are a *strict* subset of the vector space. The benefit of both approaches is that the neighborhood functions can be expressed as affine functions (see Section 7.3 for more details). The reason for our approach will be explained at the end of Section 7.3.

Definition 7.2.1 *The wrapped de Bruijn graph $\mathcal{G}_{N,l}$ is a directed graph with l layers each with 2^N nodes which are represented by N-bit strings. The layers are numbered $0, 1, \ldots, l-1$. The node represented by $v = (b_0, \ldots, b_{i^*}, \ldots, b_{N-1})$ in layer i has edges pointing to the nodes represented by $\Gamma_{i,0}(v) = (b_0, \ldots, b_{i^*}, \ldots, b_{N-1})$ and $\Gamma_{i,1}(v) = (b_0, \ldots, b_{i^*} \oplus 1, \ldots, b_{N-1})$ in layer $(i+1)$ modulo l, where i^* is i modulo N and $a \oplus b$ denotes the sum of a and b modulo 2.*

See Figure 7-1 for an example.

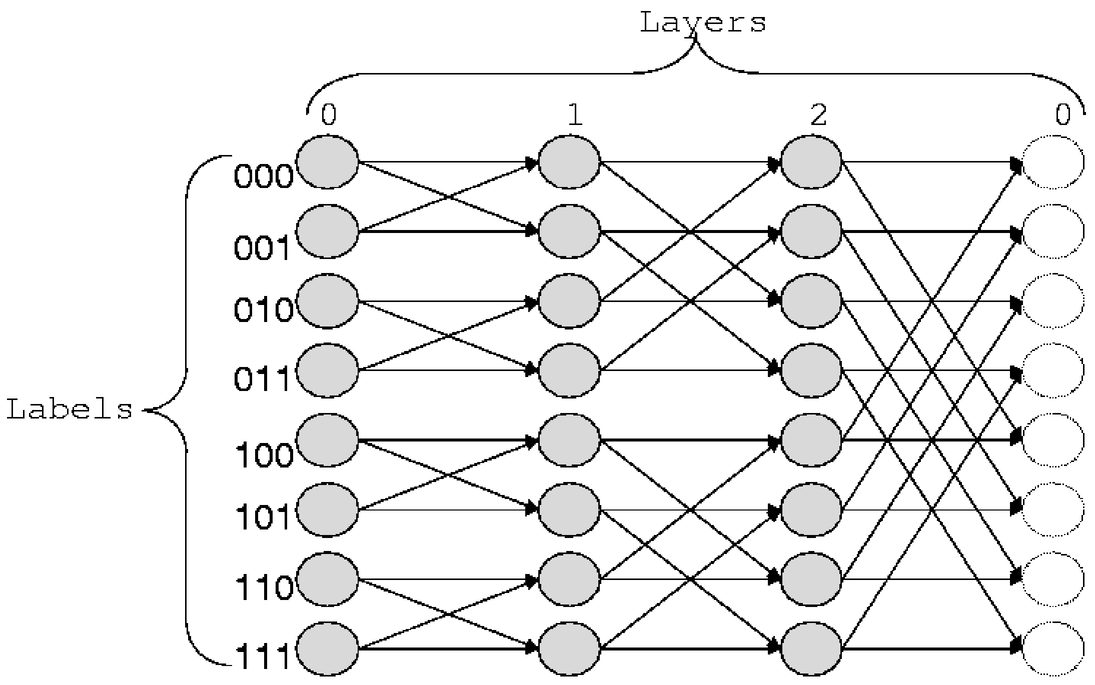

Figure 7-1: The wrapped de Bruijn graph $\mathcal{G}_{3,3}$
Notice the first and last layer are the same.

We now describe how to embed a circuit into a wrapped de Bruijn graph (see Figure 7-2 for a simple example). Given a circuit C with n gates (including both input and output gates), we associate with it a wrapped de Bruijn graph $\mathcal{G}_{N,l}$ where $N = \log n$ and $l = 5N = 5 \log n$. We then associate the nodes in layer 0 with the gates of the circuit. Now, we wish to map each wire in the circuit to a path in $\mathcal{G}_{N,l}$ between the corresponding nodes of layer 0. Standard packet-routing techniques (see [Lei92]) can be used to show that if the number of layers l is at least $5N$ then such a routing can be done with edge-disjoint paths. (Recall that we work with circuits whose fan-in and fan-out are 2.)

Thus, we can find "switches" for each of the nodes in layers $1, \ldots, l-1$ of the graph such that the output of each gate (i.e., node in layer 0) is routed to the input of the gates that require it. Each node has two inputs and two outputs, and thus there is a constant number of switches routing incoming edges to outgoing ones (See Figure 7-3). For nodes in layer 0, instead of specifying a

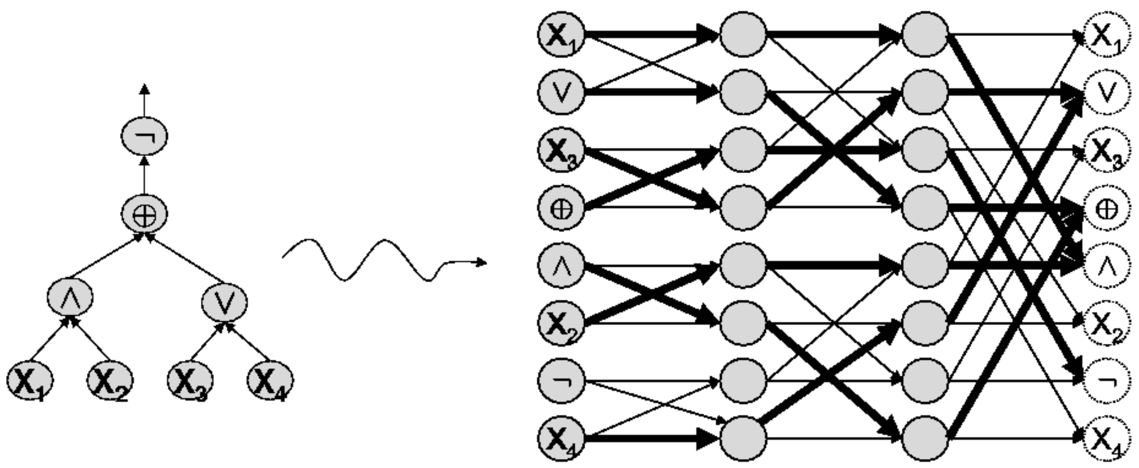

Figure 7-2: Embedding of a circuit into $\mathcal{G}_{3,3}$
In this example all paths between nodes at 0 layer are vertex disjoint. For general circuits we merely need edge disjoint paths.

switch, we specify the functionality of the Boolean gate associated to that node in the circuit (e.g. AND, OR, PARITY, NOT, INPUT, OUTPUT). Actually unary gates (such as NOT and OUTPUT) have two forms (NOT, NOT', OUTPUT, OUTPUT') in order to specify which of the two incoming edges in the de Bruijn graph to use.

This specifies the embedding of the input circuit into a well-structured circuit (based on a de Bruijn graph). More precisely, let $\mathcal{C} = \{\text{Type of switching actions}\} \cup \{\text{Type of Boolean gates}\}$ be the set of allowable gates of the well-structured circuit (See Figure 7-3) . Given a circuit on n gates, we can construct, in polynomial time, a wrapped de Bruijn graph $\mathcal{G}_{N,l}$ (where $N = \log n$ and $l = 5 \log N$) and l functions $T_0, T_1, \ldots, T_{l-1} : \{0,1\}^N \to \mathcal{C}$ where each function T_i is a specification of the gates of layer i (i.e. a specification of the switching action or Boolean functionality).

We now demonstrate how to translate a proof that a circuit is satisfiable into an assignment that satisfies the embedded circuit. A proof that a circuit is satisfiable consists of an assignment of 0's and 1's to the inputs and the gates of the circuit such that each gate's output is consistent with its inputs and the output gate evaluates to 1. The corresponding assignment to the embedded circuit consists of an assignment of 0's and 1's to the edges entering and leaving the nodes of the wrapped de Bruijn graph that is consistent with the functionality of the gates (in layer 0) and the

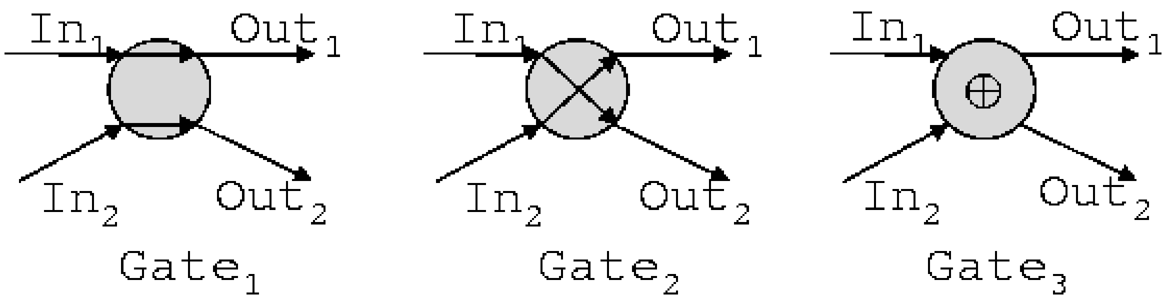

Figure 7-3: Some gates of a well-structured circuit
Gates 1–2 are switching gates, and gate 3 sits in layer 0 and computes the parity (xor) function.

switching actions of the nodes (in the other layers). Since we are assigning values to nodes of the embedded graph (and not their edges), the assignment actually associates a 4-tuple of 0's and 1's to each of the nodes in the graph indicating the value carried by the four edges incident at that node (two incoming and two outgoing). More formally, the embedded assignment is given by a set of l functions $A_0, A_1, \ldots, A_{l-1}$ where each function $A_i : \{0,1\}^N \to \{0,1\}^4$ specifies the values carried by the 4 edges incident at that vertex.

We now list the constraints on the embedded circuit that assure us that the only legal assignments are ones that correspond to legal satisfying assignments of the original circuit, i.e. assignments that correctly propagate along the edges of the circuit, correctly compute the value of every gate and produce a 1 at the output gate.

Definition 7.2.2 *The* assignment constraints *for each node of the well-structured circuit require:*

- *the two outgoing values at the node are propagated correctly to the incoming values of its neighbors at the next level,*
- *for nodes at layers $\neq 0$, the two outgoing values have the unique values dictated by the incoming values and the switching action,*
- *for non-OUTPUT nodes in layer 0, both outgoing values equal the unique value dictated by the gate functionality and the incoming values (the INPUT functionality merely requires that the two outgoing values are equal to each other)*
- *for nodes in layer 0 with an OUTPUT functionality, the appropriate incoming value equals 1*

Let $\psi : \mathcal{C} \times (\{0,1\}^4)^3 \to \{0,1\}$ be the boolean function such that $\psi(t, a, a_0, a_1) = 0$ iff a node whose T-gate is t, A-assignment is a, and whose neighbors in the next layer have A-assignments a_0 and a_1 respectively, satisfies the aforementioned assignment constraints.

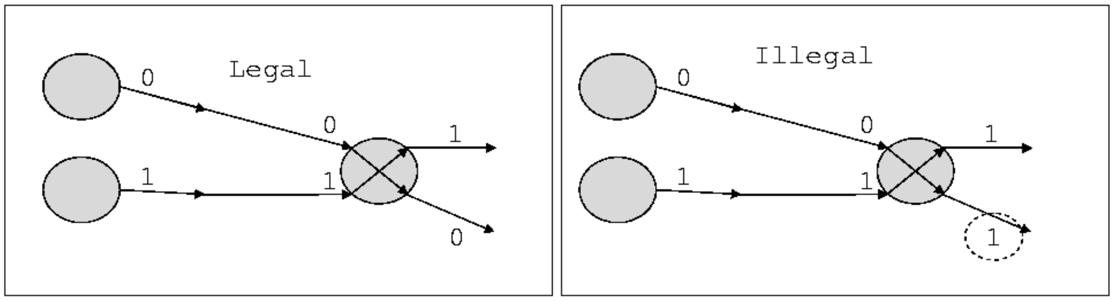

Figure 7-4: Example of legal and illegal assignments
The two vertices on the left are the inputs (at layer $i-1$) to a gate at layer i. Recall that assignments evaluate each incoming and outgoing edge of a gate.

Observe that the definition of ψ is independent of N, the assignments A_i and gates T_i. By definition, the assignment $A = (A_0, \ldots, A_{l-1})$ is legal for an embedded circuit defined by $T_0, \ldots, T_{l-1}$ if and only if for every layer i and every node v in layer i,

$$\psi\Big(T(v), A(v), A\big(\Gamma_{i,0}(v)\big), A\big(\Gamma_{i,1}(v)\big)\Big) = 0.$$

We are now ready to formally define the well-structured circuit satisfiability problem.

Definition 7.2.3 *The problem* STRUCTURED-CKTSAT *has as its instances $\langle \mathcal{G}_{N,l}, \{T_0, T_1, \ldots, T_{l-1}\}\rangle$ where $\mathcal{G}_{N,l}$ is a wrapped de Bruijn graph with l layers and $T_i : \{0,1\}^N \to \mathcal{C}$ is a specification of the node types of layer i of the graph (T_i's are specified by a table of values).*

$\langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle \in$ STRUCTURED-CKTSAT *if there exists a set of assignments $A_0, A_1, \ldots, A_{l-1}$ where $A_i : \{0,1\}^N \to \{0,1\}^4$ is an assignment to the nodes of layer i of $\mathcal{G}_N$ such that for all layers i and all nodes v in layer i,*

$$\psi\Big(T(v), A(v), A\big(\Gamma_{i,0}(v)\big), A\big(\Gamma_{i,1}(v)\big)\Big) = 0.$$

The above discussion also demonstrates the existence of a reduction from CKTSAT to STRUCTURED-CKTSAT which does not blow up the length of the target instance by more than a logarithmic multiplicative factor.

Proposition 7.2.4 *There exists a polynomial time reduction $\mathcal{R}$ from* CKTSAT *to* STRUCTURED-CKTSAT *such that for any circuit C, it holds that $C \in$* CKTSAT *if and only if $\mathcal{R}(C) \in$* STRUCTURED-CKTSAT. *Moreover, if C is a circuit of size n, then $\mathcal{R}(C) = \langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle$ where $N = \lceil \log n \rceil$ and $l = 5N$.*

Remark 7.2.5 *The above reduction though known to take polynomial time (via routing techniques) is not known to be of almost linear time.*

Remark 7.2.6 *We observe that if C is a satisfiable circuit, then any set of assignments $A_0, \ldots, A_l$ proving that the reduced instance $\mathcal{R}(C) = \langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle$ is a YES instance of* STRUCTURED-CKTSAT *contains within it a satisfying assignment to the circuit C. Specifically, let I be the set of nodes in layer 0 that have gate functionality INPUT associated with them. Then the assignment A_0 restricted to the set of nodes I (i.e.,$A_0|_I$) contains a satisfying assignment. More precisely, the satisfying assignment is obtained by concatenating the third bit (i.e., first outgoing bit) of $A_0|_i \in \{0,1\}^4$ for all $i \in I$. Conversely, every satisfying assignment w to C can be extended to $A_0, \ldots, A_{l-1}$ such that $A_0|_I$ contains w. This is done by computing the values of all gates in the computation of $C(w)$, setting the outgoing bits of A_0 according to these values, and routing them throughout $\mathcal{G}_{N,l}$ according to the switching actions to obtain $A_1, \ldots, A_{l-1}$ and the incoming bits of A_0. This observation will be vital while constructing PCPs of proximity(see Section 8).*

Remark 7.2.7 *Suppose the input circuit C is a* linear *circuit, in the sense that all gates are INPUT, OUTPUT, or PARITY gates, and the OUTPUT gates test for 0 rather 1 (See Definition 9.5.1). Then it can be verified that the transformation mapping satisfying assignments w of C to legal assignments $A_0, \ldots, A_{l-1}$ of $\mathcal{R}(C)$ is* GF(2)*-linear. The reason is that each gate in the computation of $C(w)$ is a* GF(2)*-linear function of w. These remarks will be used in the coding applications, to obtain linear codes (see Section 9.5 for more information).*

7.3 Arithmetization

In this section, we construct an algebraic version of STRUCTURED-CKTSAT by arithmetizing it along the lines of Harsha and Sudan [HS00]. The broad overview of the arithmetization is as follows: We embed the nodes in each layer of the wrapped de Bruijn graph $\mathcal{G}_{N,l}$ in a vector space and extend the gate specifications and assignments to low-degree polynomials over this space. Finally, we express the assignment constraints (Definition 7.2.2) as a pair of polynomial identities satisfied by these polynomials.

First for some notation. Let m be a parameter. Set h such that $h = N/m$ where 2^N is the number of nodes in each layer of the de Bruijn graph. Choose a finite extension field F of GF(2) of size roughly $c_F m^2 2^h = c_F m^2 2^{N/m}$ where c_F is a suitably large constant to be specified later. Specifically, take $F = \mathrm{GF}(2)^f$ for $f = h + 2\log m + \log c_F$. Let $\{e_0, e_1, \ldots, e_{f-1}\}$ be a basis of F over GF(2). Set H to be a subspace of $\mathrm{GF}(2)^f$ (and hence a subset of F) spanned by $\{e_0, \ldots, e_{h-1}\}$. Note that H^m is a subset of the space F^m. Furthermore, $|H^m| = 2^N$. Hence, we can embed each layer of the graph $\mathcal{G}_{N,l}$ in F^m by identifying the node $v = (b_0, \ldots, b_{N-1}) \in \{0,1\}^N$ with the element $(b_0e_0 + \cdots + b_{h-1}e_{h-1}, b_he_0 + \cdots + b_{2h-1}e_{h-1}, \ldots, b_{(m-1)h}e_0 + \cdots + b_{mh-1}e_{h-1})$ of H^m. Henceforth, we use both representations (N-bit string and element of H^m) interchangeably. The representation will be clear from the context.

Any assignment $S : H^m \to F$ can be interpolated to obtain a polynomial $\tilde{S} : F^m \to F$ of degree at most $|H|$ in each variable (and hence a total degree of at most $m|H|$) such that $\tilde{S}|_{H^m} = S$ (i.e., the restriction of $\tilde{S}$ to H^m coincides with the function S). Conversely, any polynomial $\tilde{S} : F^m \to F$ can be interpreted as an assignment from H^m to F by considering the function restricted to the sub-domain H^m.

Recall that $\mathcal{C}$ and $\{0,1\}^4$ are the set of allowable gates and assignments given by the gate functions T_i and assignments A_i in the STRUCTURED-CKTSAT problem. We identify $\mathcal{C}$ with some fixed subset of F and we identify $\{0,1\}^4$ with the set of elements spanned by $\{e_0, e_1, e_2, e_3\}$ over GF(2). With this identification, we can view the assignments A_i and gates T_i as functions $A_i : H^m \to F$ and $T_i : H^m \to F$ respectively. Furthermore, we can interpolate these functions, as mentioned above, to obtain polynomials $\tilde{A}_i : F^m \to F$ and $\tilde{T}_i : F^m \to F$ of degree at most $m|H|$ over F.

We now express the neighborhood functions of the graph in terms of affine functions over F^m. This is where the nice structure of the wrapped de Bruijn graph will be useful. For any positive integer i, define affine transformations $\tilde{\Gamma}_{i,0}, \tilde{\Gamma}_{i,1} : F^m \to F^m$ as follows: $\tilde{\Gamma}_{i,0}$ is the identity function. For $\tilde{\Gamma}_{i,1}$, first let $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$. Then $\tilde{\Gamma}_{i,1}(z_0, \ldots, z_{m-1}) = (z_0, \ldots, z_{t-1}, z_t + e_u, z_{t+1}, \ldots, z_{m-1})$.[1] It can checked from the above definition that for any layer i and node x in layer i (which we view as a point in H^m), we have $\tilde{\Gamma}_{i,j}(x) = \Gamma_{i,j}(x)$ for $j = 0, 1$. In other words, $\tilde{\Gamma}$ is an extension of the neighborhood relations of the graph $\mathcal{G}_{N,l}$ over F^m.

Finally, we now express the assignment constraints (Definition 7.2.2) as polynomial identities. The first of these identities checks that the assignments given by the assignment polynomial $\tilde{A}_i$ are actually elements of $\{0,1\}^4$ for points in H^m. For this purpose, let $\psi_0 : F \to F$ be the univariate polynomial of degree 2^4 given by

$$\psi_0(z) = \prod_{\alpha \in \{0,1\}^4} (z - \alpha) \tag{7.1}$$

This polynomial satisfies $\psi_0(z) = 0$ iff $z \in \{0,1\}^4$ (recall we identified $\{0,1\}^4$ with a subset of F spanned by $e_0, \ldots, e_3$). We check that $\psi_0(\tilde{A}_i(x)) = 0$ for all $x \in H^m$ and all layers i. We then arithmetize the rule ψ (from Definition 7.2.2) to obtain a polynomial $\psi_1 : F^4 \to F$. In other words, $\psi_1 : F^4 \to F$ is a polynomial such that $\psi_1(t, a, a_0, a_1) = \psi(t, a, a_0, a_1)$ for all $(t, a, a_0, a_1) \in \mathcal{C} \times (\{0,1\}^4)^3$. The degree of ψ_1 can be made constant, because $|\mathcal{C}|$ and $|\{0,1\}^4|$ are constant.[2] The two polynomial identities we would like to check are $\psi_0(\tilde{A}_i(x)) = 0$ and $\psi_1(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))) = 0$ for all $x \in H^m$. For notational convenience, we express these two conditions together as a pair

[1]An alternate description of $\tilde{\Gamma}_{i,1}$ is as follows: Since $F = \mathrm{GF}(2)^f$, we can view F^m as mf-dimensional space over GF(2). Hence, any vector $(z_0, \ldots, z_{m-1})$ can be written as $(b_{0,0}, \ldots, b_{0,f-1}, b_{1,0}, \ldots, b_{1,f-1}, \ldots, b_{m-1,0}, \ldots, b_{m-1,f-1})$. Furthermore, we note that for any vector $(z_0, \ldots, z_{m-1})$ in H^m, $b_{r,s} = 0$ for all $s \geq h$ and all r. It can now be checked that $\tilde{\Gamma}_{i,1}$ is the affine transformation that flips the bit $b_{t,u}$ where $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$..

[2]Notice that we do not specify ψ_1 uniquely at this stage. Any choice of a constant-degree polynomial will work in this section, but to enforce linearity, we will use a somewhat non-standard choice in Section 9.5. Specifically, we argue that if C is a linear circuit, then ψ_1 can be picked to be GF(2)-linear transformations over $GF(2)$, and we point out that ψ_0 is a $GF(2)$-linear transformation. For more details see Section 9.5.

of polynomials $\psi' = (\psi_0, \psi_1) : F^4 \to F^2$ such that $\psi'(x_1, x_2, x_2, x_4) = (\psi_0(x_2), \psi_1(x_1, x_2, x_3, x_4))$. Let κ be the maximum of the degree of these two polynomials. In order to make these polynomial identities sufficiently redundant,, we set c_F to be a sufficiently large constant (say 100) such that $\kappa m^2 2^h/|F|$ is low.

We have thus reduced STRUCTURED-CKTSAT to an algebraic consistency problem, which we shall call the AS-CKTSAT problem (short for ALGEBRAIC-STRUCTURED-CKTSAT)[3].

Definition 7.3.1 *The promise problem* AS-CKTSAT = (AS-CKTSAT$^{\mathrm{YES}}$, AS-CKTSAT$^{\mathrm{NO}}$) *has as its instances* $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ *where* F *is an finite extension field of* $\mathrm{GF}(2)$ *(i.e.,* $F = \mathrm{GF}(2)^f$ *for some* f*),* H *a* $\mathrm{GF}(2)$*-linear subspace of* F *and* $\tilde{T}_i : F^m \to F$*, for* $i = 0, \ldots, l-1$*, a sequence of polynomials of degree* d*, where* $|H| = n^{1/m}$*,* $d = m \cdot |H|$*, and* $F = c_F \cdot md$*. The field* F *is specified by an irreducible polynomial* $p(x)$ *of degree* f *over* $\mathrm{GF}(2)$*,* H *is taken to be spanned by the first* $h = \log |H|$ *canonical basis elements, and each of the polynomials* $\tilde{T}_i$ *is specified by a list of coefficients.*

- $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle \in$ AS-CKTSAT$^{\mathrm{YES}}$ *if there exist a sequence of degree* d *polynomials* $\tilde{A}_i : F^m \to F, i = 0, \ldots, l-1$ *such that for all* $i = 0, \ldots, l-1$ *and all* $x \in H^m$*,*

$$\psi'\Big(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\Big) = (0,0)$$

- $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle \in$ AS-CKTSAT$^{\mathrm{NO}}$ *if for all functions* $\tilde{A}_i : F^m \to F, i = 0, \ldots, l-1$ *there exists an* $i = 0, \ldots, l-1$ *and* $x \in H^m$ *such that,*

$$\psi'\Big(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\Big) \neq (0,0)$$

where $\tilde{\Gamma}_{i,j}$*'s and* ψ' *are as defined earlier. (Recall that the* $\tilde{\Gamma}$*'s are linear function while* ψ' *represents a pair of polynomials of degree at most* κ*.)*

From the above discussion we have the following reduction from STRUCTURED-CKTSAT to AS-CKTSAT.

Proposition 7.3.2 *There exists a polynomial-time computable function* $\mathcal{R}$ *mapping any instance* $\mathcal{I} = \langle \mathcal{G}_{N,l}, \{T_0, T_1, \ldots, T_{l-1}\}\rangle$ *of* STRUCTURED-CKTSAT *and parameter* $m \leq \log n/\mathrm{loglog} n$ *(where* $n = |\mathcal{I}|$*) to an instance* $\mathcal{R}(\mathcal{I}, 1^m)$ *of* AS-CKTSAT *such that*

$$\begin{aligned} \mathcal{I} \in \text{STRUCTURED-CKTSAT} &\implies \mathcal{R}(\mathcal{I}, 1^m) \in \text{AS-CKTSAT}^{\mathrm{YES}} \\ \mathcal{I} \notin \text{STRUCTURED-CKTSAT} &\implies \mathcal{R}(\mathcal{I}, 1^m) \in \text{AS-CKTSAT}^{\mathrm{NO}} \end{aligned}$$

Moreover, if $\mathcal{R}(\mathcal{I}, 1^m) = \langle 1^{n'}, 1^{m'}, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l'-1}\}\rangle$*, then* $n' = 2^N$ *(the number of nodes in each layer of the de Bruijn graph* $\mathcal{G}_{N,l}$*),* $m' = m$*, and* $l' = l$ *(the number of layers in the de Bruijn graph).*

[3] AS-CKTSAT is actually a promise problem.

Combining Propositions 7.2.4 and 7.3.2, we have the following.

Proposition 7.3.3 *There exists a polynomial-time computable function $\mathcal{R}$ mapping any circuit C and parameter $m \leq \log n/\mathrm{loglog} n$ (where $n = |C|$) to an instance $\mathcal{R}(C, 1^m)$ of* AS-CKTSAT *such that $C \in$* CKTSAT $\iff \mathcal{R}(C, 1^m) \in$ AS-CKTSAT.

Moreover, if C is a circuit of size n then $\mathcal{R}(C, 1^m) = \langle 1^{n'}, 1^{m'}, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l'-1}\}\rangle$, where $n' = \Theta(n)$, $m' = m$, and $l' \leq 5 \log n'$. Thus, $|\mathcal{R}(C, 1^m)| = O((c_F m^2)^m \log n) \cdot |C|$.

Remark 7.3.4 *Following Remark 7.2.6, if C is a satisfiable circuit, then any set of polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ proving that the reduced instance $\mathcal{R}(C, 1^m) = \langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ is a YES instance of* AS-CKTSAT *contain within it a satisfying assignment to the circuit C. Specifically, the set I (of layer-0 nodes with INPUT functionality in $\mathcal{G}_{N,l}$) from Remark 7.2.6 can now be viewed as a subset $I \subseteq H^m$. Then the polynomial $\tilde{A}_0 : F^m \to F$ restricted to the set I (i.e.,$\tilde{A}_0|_I$) contains a satisfying assignment (again as a concatenation of third-bits). Conversely, every satisfying assignment w to C can be extended to a set of polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ such that $\tilde{A}_0|_I$ contains w. This is done by taking low-degree extensions of the functions $A_0, \ldots, A_{l-1}$ from Remark 7.2.6.*

Remark 7.3.5 *Following Remark 7.2.7, if C is a linear circuit, then the mapping of satisfying assignments w of C to polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ satisfying $\mathcal{R}(C)$ is* GF(2)*-linear. This is due to Remark 7.2.7, the association of $\{0,1\}^4$ with the linear space spanned by $\{e_0, e_1, e_2, e_3\}$ in F, and from the fact that the interpolation from A_i to $\tilde{A}_i$ is F-linear and hence* GF(2)*-linear. For more information see Section 9.5.*

Comment: Recall that the arithmetization was obtained by considering low-degree extensions over F^m of functions from H^m to H. If H were a subfield of the field F this step would have caused a quadratic blow-up, and we avoid this problem by not insisting that H be a field. In [PS94, Spi95], H is a field and $F = H^2$ is an extension of it, but the PCP system refers only to a $O(|H|)$-sized subset of F. We cannot take this approach because we will be using a total low-degree test, which needs to refer to the entire vector space F^m. In contrast, in [PS94, Spi95] an individual low-degree test is used, which can work with a subset of F^m.

7.4 The PCP verifier

We design a PCP verifier for CKTSAT via the reduction to AS-CKTSAT based on the randomness-efficient low-degree tests of Ben-Sasson *et al.* [BSVW03]. Given a circuit C, the verifier reduces it to an instance of the problem AS-CKTSAT (Proposition 7.3.3). The proof consists of a sequence of oracles $\tilde{A}_i : F^m \to F$ for $i = 0, \ldots, l-1$ and an auxiliary sequence of oracles $P_{i,j} : F^m \to F^2$ for $i = 0, \ldots, l-1$ and $j = 0, \ldots, m$. For each i and j, we view the auxiliary oracle $P_{i,j} : F^m \to F^2$ as a pair of functions $P_{i,j}^{(0)} : F^m \to F$ and $P_{i,j}^{(1)} : F^m \to F$ (i.e., $P_{i,j}(x) = (P_{i,j}^{(0)}(x), P_{i,j}^{(1)}(x))$). This auxiliary sequence of oracles helps the verifier to check that the functions $\tilde{A}_i$ satisfy condition ψ' (see Definition 7.3.1).

The verifier expects the first subsequence of auxiliary oracles $P_{i,0}(\cdot)$ for $i = 0, \ldots, l-1$, to satisfy the following relation:

$$P_{i,0}(x) = \psi'\Big(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\Big) \quad \forall x \in F^m \tag{7.2}$$

Furthermore, it expects $P_{i,0}(x) = 0$ for every $x \in H^m$. Indeed, by Definition 7.3.1, we have:

Lemma 7.4.1 *1. If $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ is a YES instance of* AS-CKTSAT*, satisfied by polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$, and $P_{0,0}, \ldots, P_{l-1,0}$ are defined according to Equation 7.2, then $P_{i,0}(x) = (0,0)$ for all $x \in H^m$.*

2. If $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ is a NO instance of AS-CKTSAT*, then for any sequences of functions $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$, $P_{0,0}, \ldots, P_{l-1,0}$, either Equation 7.2 fails to hold for some i or $P_{i,0}(x) \neq (0,0)$ for some i and some $x \in H^m$.*

Recalling that the degree of the constraint ψ' (see Definition 7.3.1) is at most κ and that the $\tilde{A}_i$'s are of degree at most $d = m \cdot |H|$, we observe that the $P_{i,0}$'s can be taken to be of degree at most κd in Part 1.

As mentioned above, the verifier now needs to check that the functions $P_{i,0}$ vanish on the set H^m. For this we use a "zero-propagation test", based on the sum-check protocol of Lund *et al.* [LFKN92][4] . Specifically, the verifier expects the remaining set of auxiliary oracles $P_{i,j} = (P^{(0)}_{i,j}, P^{(1)}_{i,j})$ ($i = 0, \ldots, l-1$ and $j = 1, \ldots, m$) to satisfy the following relations: Let $H = \{h_0, \ldots, h_{|H|-1}\}$ be some fixed enumeration of the elements in H. For all $b \in \{0,1\}$,

$$P^{(b)}_{i,j}\Big(\underbrace{x_1, \ldots, x_{j-1}}, x_j, \underbrace{x_{j+1}, \ldots, x_m}\Big) = \sum_{k=0}^{|H|-1} P^{(b)}_{i,j-1}\Big(\underbrace{x_1, \ldots, x_{j-1}}, h_k, \underbrace{x_{j+1}, \ldots, x_m}\Big) x_j^k, \tag{7.3}$$
$$\forall (x_1, \ldots, x_m) \in F^m$$

These relations ensure that for all i and $j \geq 1$, $P_{i,j}(\cdot)$ vanishes on $F^j \times H^{m-j}$ iff the function $P_{i,j-1}(\cdot)$ vanishes on $F^{j-1} \times H^{m-j+1}$. In other words:

Lemma 7.4.2 $P^{(b)}_{i,j}|_{F^j \times H^{m-j}} \equiv 0 \iff P^{(b)}_{i,j-1}|_{F^{j-1} \times H^{m-j+1}} \equiv 0.$

Thus, for all i, $P_{i,m}$ vanishes on the entire space F^m iff $P_{i,0}$ vanishes on H^m. Also, as $P^{(b)}_{i,0}$ has degree at most κd in each variable, so does $P^{(b)}_{i,j}$ for each i and j. Hence, the degree of $P^{(b)}_{i,j}$ is at most κd.

Thus, the verifier needs to make the following checks

[4]For checking that the functions $P_{i,0}$ vanish on the set H^m, we could use the (simpler) "zero-on-subcube" test described in Section 5.3.2, however the "zero-on-subcube" protocol was developed as a result of the latter work of Ben-Sasson and Sudan [BS04]. So, for historical reasons, we retain the (slighly more complicated) zero-propogation test in this section.

- LOW-DEGREE TEST
  For $i = 0, \ldots, l-1$ and $j = 0, \ldots, m$, the sequence of functions $\tilde{A}_i$ are polynomials of degree at most $d = m \cdot |H|$ and the sequence of functions $P_{i,j}$ are pairs of polynomials of degree at most κd,

- EDGE-CONSISTENCY TEST
  For $i = 0, \ldots, l-1$, the functions $P_{i,0}$ obey Equation (7.2),

- ZERO PROPAGATION TEST
  For $i = 0, \ldots, l-1$ and $j = 1, \ldots, m$, the functions $P_{i,j}$ satisfy Equation (7.3),

- IDENTITY TEST
  For $i = 0, \ldots, l-1$, the functions $P_{i,m}$ are identically zero on the entire domain F^m.

The Low-Degree test in most earlier construction of PCP verifiers is performed using the "line-point" test. The "line-point" low degree test first chooses a random line, a random point on this line and checks if the restriction of the function to the line (given by a univariate polynomial) agrees with the value of the function at the point. A random line l is typically chosen by choosing two random points $x, y \in F^m$ and setting $l = l_{x,y} = \{x + ty | t \in F\}$. However, this requires $2m \log |F|$ bits of randomness which is too expensive for our purposes. We save on randomness by using the low-degree test of Ben-Sasson *et al.* [BSVW03] based on small-biased spaces (see Section A.2 for more details). The low-degree test of [BSVW03] uses pseudorandom lines instead of totally random lines in the following sense: The pseudorandom line $l = l_{x,y}$ is chosen by choosing the first point x at random from F^m, while the second point y is chosen from a λ-biased subset S_λ of F^m. This needs only $\log |S_\lambda| + \log |F|^m$ bits of randomness. We further save on randomness by the use of *canonical lines*[5]. Consider any pseudorandom line $l = l_{x,y}$ where $x \in F^m$ and $y \in S_\lambda$. We observe that for every $x' \in l$, we have $l_{x',y} = l_{x,y}$. In other words, $|F|$ different choices of random bits leads to the same line $l_{x,y}$. We prevent this redundancy by representing each line in a canonical manner. A canonical line is chosen by first choosing a random point y from the λ-biased set S_λ. We view this y as specifying the direction (i.e., slope) of the line. This direction partitions the space F^m into $|F|^{m-1}$ parallel lines (each with direction y). We enumerate these lines arbitrarily and select one of them uniformly at random. Thus, choosing a random canonical line costs only $\log |S_\lambda| + \log |F|^{m-1}$ bits of randomness. A further point to be noted is that we perform a "line" test instead of the regular "line-point" test: The test queries the function for all points along the canonical line $l_{x,y}$ and verifies that the restriction of the function to this line is a low-degree polynomial.

[5]It is to be noted that the canonical representation of lines has been used either implicitly or explicitly in the soundness analysis of all earlier uses of the Low-Degree Test. However, this is the first time that the canonical representation is used to save on the number of random bits.

Having performed the low-degree test (i.e., verified that the polynomials $\tilde{A}_i$'s and $P_{i,j}$'s are close to low-degree polynomials), the verifier then performs each of the NODE-CONSISTENCY TEST, ZERO PROPAGATION TEST, and IDENTITY TESTs by choosing a suitable small-sized sample in the entire space and checking if the corresponding condition is satisfied on that sample. For the ZERO PROPAGATION TEST indeed the natural sample is an axis-parallel line. For the EDGE-CONSISTENCY TEST and IDENTITY TEST, the sample we use is any set of $|F|$ points selected from a partition of F^m into $|F|^{m-1}$ equal sets.

We are now ready to formally describe the PCP verifier for CKTSAT. We parameterize the PCP verifier in terms of m, the number of dimensions in our intermediate problem AS-CKTSAT, and λ, the parameter of the λ-biased sets of F^m required for the low-degree tests of Ben-Sasson *et al.* [BSVW03]. We rely on the fact that λ-biased subsets of F^m of size at most $\mathrm{poly}(\log|F|^m, 1/\lambda)$ can be constructed efficiently [NN90, AGHP92].

PCP–VERIFIER$^{\tilde{A}_i, P_{i,j}; i=0,\ldots,l-1; j=0,\ldots,m}_{m,\lambda}(C)$.

1. Use Proposition 7.3.3 to reduce the instance C of CKTSAT, using parameter m, to an instance $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CKTSAT, and set $d = m \cdot |H|$.

   Notation: We let $S_\lambda \subset F^m$ be a λ-biased set of size at most $\left(\frac{\log|F|^m}{\lambda}\right)^2$ [AGHP92]. Let $F^m = \biguplus_{\eta=1}^{|F|^{m-1}} U_\eta$ and $F^m = \biguplus_{\eta=1}^{|F|^{m-1}} V_\eta$ be two arbitrary partitions of the space F^m into $|F|$-sized sets each.

2. Choose a random string R of length $\log(|S_\lambda| \cdot |F|^{m-1})$. [Note: We reuse R in all tests, but only the LOW-DEGREE TEST utilizes the full length of R.]

3. LOW-DEGREE TEST

   Use random string R to determine a random canonical line $\mathcal{L}$ in F^m using the λ-biased set S_λ.

   For $i = 0, \ldots, l-1$,

      Query oracle $\tilde{A}_i$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_i$ to $\mathcal{L}$ is not a (univariate) polynomial of degree at most d.

   For $i = 0, \ldots, l-1$, $j = 0, \ldots, m$, and $b \in \{0, 1\}$,

      Query oracle $P^{(b)}_{i,j}$ on all points along the line $\mathcal{L}$ and reject if the restriction of $P^{(b)}_{i,j}$ to $\mathcal{L}$ is not a (univariate) polynomial of degree at most κd.

4. EDGE-CONSISTENCY TEST

   Use the random string R to determine a random set U_η of the partition $F^m = \biguplus_{\eta=1}^{|F|^{m-1}} U_\eta$.

   For $i = 0, \ldots, l-1$,

      For all $x \in U_\eta$, query $P_{i,0}(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x))$ and $\tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))$ and reject if Equation (7.2) is not satisfied.

5. ZERO PROPAGATION TEST

   For $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0, 1\}$,

   > Use random string R to determine a random jth axis-parallel line in F^m of the form $\mathcal{L} = \{(a_1, \ldots, a_{j-1}, X, a_{j+1}, \ldots, a_m) : X \in F\}$. Query $P^{(b)}_{i,j-1}$ and $P^{(b)}_{i,j}$ along all the points in $\mathcal{L}$. Reject if either the restriction of $P^{(b)}_{i,j-1}$ or $P^{(b)}_{i,j}$ to $\mathcal{L}$ is not a univariate polynomial of degree at most κd or if any of the points on the line $\mathcal{L}$ violate Equation (7.3).

6. IDENTITY TEST

   Use the random string R to determine a random set V_η of the partition $F^m = \biguplus_{\eta=1}^{|F|^{m-1}} V_\eta$. For $i = 0, \ldots, l-1$,

   > For all $x \in V_\eta$, query $P_{i,m}(x)$. Reject if any of these $P_{i,m}(x)$ is not $(0, 0)$.

   Accept if none of the above tests reject.

Remark 7.4.3

1. *The* LOW-DEGREE TEST *requires* $\log(|S_\lambda| \cdot |F|^{m-1})$ *random bits to generate a canonical line in* F^m *using the* λ*-biased set, while each of the other tests require at most* $\log(|F|^{m-1})$ *bits of randomness. Hence, the string* R *suffices for each of the tests. For the settings of parameters we use,* $\log(|S_\lambda| \cdot |F|^{m-1})$ *is typically significantly smaller than* $\log(|F|^m)$*, which we would not be able to afford.*

2. *The* EDGE-CONSISTENCY TEST *and* IDENTITY TEST *in the "standard" sense are usually performed by selecting a random point in the space* F^m *and checking whether the corresponding condition is satisfied. However, we state these tests in a "non-standard" manner using partitions of the space* F^m *into* $|F|$ *sized tests so that these tests can easily be adapted when we construct the robust PCP verifier (see Chapter 9). The non-standard tests are performed in the following manner: Choose a random set in the partition and perform the standard test for each point in the set. At present, we can work with any partition of* F^m*, however we will later need specific partitions to get "robustness".*

7.5 Analysis of the PCP verifier

We now analyze the PCP verifier above. The analysis below assumes that the parameters satisfy $m \leq \log n/\mathrm{loglog} n$ and $\lambda \leq 1/c \log n$ for a sufficiently large constant c. Theorem 7.1.1 can be deduced by setting $\lambda = 1/c \log n$.

Complexity: The PCP VERIFIER makes $O(lm|F|) = O(m^3 n^{1/m} \log n)$ queries each of which expects as an answer an element of F or F^2 (i.e., a string of length $O(\log |F|)$). Hence, the total (bit) query complexity is $O(lm|F| \log |F|) = O(lm \cdot c_F m^2 n^{1/m} \log(c_F m^2 n^{1/m}))$. Recalling that $l = 5 \log n$, this quantity is at most $O(m^2 n^{1/m} \log^2 n)$ for $m \leq \log n$. For the decision complexity, we note that the main computations required are (a) testing whether a function is a low-degree univariate polynomial over F (for LOW-DEGREE TEST and ZERO PROPAGATION TEST), (b) evaluating ψ' on $|F|$ quadruples of points (for EDGE-CONSISTENCY TEST), and (c) univariate polynomial interpolation and evaluation (for testing (7.3) in ZERO PROPAGATION TEST). We now argue that each of these can be done with a nearly linear ($\tilde{O}(|F|)$) number of operations over F, yielding a nearly linear ($\tilde{O}(q)$) decision complexity overall. Each evaluation of ψ' can be done with a constant number of F-operations because ψ' is of constant degree. Polynomial interpolation and evaluation can be done with a nearly linear number of F-operations by [SS71, Sch77], and testing whether a function is of low degree reduces to polynomial interpolation (interpolate to represent as a polynomial of degree $|F|-1$ and check that the high-degree coefficients are zero). Each F-operations can be done with $\tilde{O}(\log |F|)$ bit-operations, using the polynomial multiplication algorithm of [SS71, Sch77] (over GF(2)).

The number of random bits used by the verifier is exactly $\log(|S_\lambda| \cdot |F|^{m-1})$. Let $n' = |F|^m$. Then $\log(|S_\lambda| \cdot |F|^{m-1}) = \left(1 - \frac{1}{m}\right) \log n' + \log\left(\text{poly}\left(\frac{\log n'}{\lambda}\right)\right) = \left(1 - \frac{1}{m}\right)\log n' + O(\log\log n') + O\left(\log\left(\frac{1}{\lambda}\right)\right)$. Now, $n' = (c_F m^2)^m n$. Hence, $\log n' = \log n + 2m \log m + O(m)$ and $\log\log n' = \log\log n + O(\log m)$. Thus, the total randomness is at most $\left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O\left(\log\left(\frac{1}{\lambda}\right)\right)$.

We summarize the above observations in the following proposition for future reference.

Proposition 7.5.1 *The randomness, query and decision complexities of the* PCP–VERIFIER *are* $r = \left(1 - \frac{1}{m}\right)\log n + O(m\log m) + O(\log\log n) + O\left(\log\left(\frac{1}{\lambda}\right)\right)$, $q = O(m^2 n^{1/m} \log^2 n)$ *and* $d = \tilde{O}(q)$ *respectively.*

Completeness: If C is satisfiable, then the reduction reduces it to an YES instance of AS-CKTSAT. Then by definition there exist polynomials $\tilde{A}_i$ that satisfy constraint ψ'. Setting $P_{i,j}$ according to Equations (7.2) and (7.3), we notice that the verifier accepts with probability one.

Soundness: To prove the soundness, we need to prove that if C is not satisfiable then the verifier accepts with probability bounded away from 1. We will prove a stronger statement. Recall from Remark 7.3.4 that the function $\tilde{A}_0 : F^m \to F$ supposedly has the satisfying assignment embedded within it. Let $I \subset F^m$ be the set of locations in F^m that contains the assignment (i.e., $\tilde{A}_0|_I$ is supposedly the assignment).

Lemma 7.5.2 *There exists a constant c and a constant $0 < \varepsilon_0 < 1$ such that for all ε, m, λ satisfying $\varepsilon \leq \varepsilon_0$, $m \leq \log n/\text{loglog}n$ and $\lambda \leq 1/c \log n$, the following holds. Suppose the proof oracle $\tilde{A}_0 : F^m \to F$ is 4ε-far from any polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$, then the verifier accepts proof oracles $\{\tilde{A}_i\}$ and $\{P_{i,j}\}$ with probability at most $1 - \varepsilon$.*

Proof: Let α be the universal constant from Theorem A.2.4. Set $\varepsilon_0 = \min\{\alpha, \frac{1}{22}\}$. Let $d = m2^h$, and choose c_F to be a large enough constant such that $\kappa md/|F| = \kappa/c_F \leq \varepsilon_0$. Suppose each of the functions $\tilde{A}_i$ are 4ε-close to some polynomial of degree md and each of the functions $P_{i,j}^{(b)}$ is 4ε-close to some polynomial of κmd. If this were not the case, then by Theorem A.2.4 the LOW-DEGREE TEST accepts with probability at most $1 - \varepsilon$ for the polynomial that is 4ε-far. It can be verified that the parameters satisfy the requirements of Theorem A.2.4, for sufficiently large choices of the constants c_F and c and sufficiently small ε.

For each $i = 0, \ldots, l-1$, let $\widehat{A}_i : F^m \to F$ be the polynomial of degree at most md that is 4ε-close to $\tilde{A}_i$. Similarly, for each $i = 0, \ldots, l-1, j = 0, \ldots, m$ and $b \in \{0, 1\}$, let $\widehat{P}_{i,j}^{(b)}$ be the polynomial of degree at most κmd that is 4ε-close to $P_{i,j}^{(b)}$. Such polynomials are uniquely defined since every two polynomials of degree κmd disagree in at least a $1 - \frac{\kappa md}{|F|} \geq 1 - \varepsilon_0 > 8\varepsilon$ fraction of points. As in the case of the $P_{i,j}$'s, let $\widehat{P}_{i,j} : F^m \to F^2$ be the function given by $\widehat{P}_{i,j}(x) = (\widehat{P}_{i,j}^{(0)}(x), \widehat{P}_{i,j}^{(1)}(x))$.

By hypothesis, $\widehat{A}_0|_I$ does not satisfy C. Then, by Lemmas 7.4.1 and 7.4.2, at least one of the following must hold.

(a) There exists $i = 0, \ldots, l-1$ and $b \in \{0, 1\}$ such that $\widehat{P}_{i,m}^{(b)} \not\equiv 0$.

Then for this i, the IDENTITY TEST fails unless a random set V_η is chosen such that for all $x \in V_\eta$, $P_{i,m}^{(b)}(x) = 0$. Hence, it must be the case that for all $x \in V_\eta$, either $P_{i,m}^{(b)}(x) \neq \widehat{P}_{i,m}^{(b)}(x)$ or $\widehat{P}_{i,m}^{(b)}(x) = 0$. Since the $V_\eta's$ form a partition of F^m, the probability of this occurring is upper-bounded by the probability that a random $x \in F^m$ satisfies either $P_{i,m}^{(b)}(x) \neq \widehat{P}_{i,m}^{(b}(x)$ or $\widehat{P}_{i,m}^{(b)}(x) = 0$. This probability is at most $4\varepsilon + \frac{\kappa md}{|F|} = 4\varepsilon + \frac{\kappa}{c_F} \leq 5\varepsilon_0$, where we use the fact that $\widehat{P}_{i,m}^{(b)}$ is 4ε-close to $P_{i,m}^{(b)}$ and that a nonzero polynomial of degree κmd vanishes on at most a $\kappa md/|F|$ fraction of points.

(b) There exists $i = 0, \ldots, l-1$ such that $\widehat{P}_{i,0}$, $\widehat{A}_i$, and $\widehat{A}_{i+1}$ do not obey Equation (7.2).

In other words, $\widehat{P}_{i,0}(x) \not\equiv \psi'((\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$. Then for this i, the EDGE-CONSISTENCY TEST fails unless a random partition U_η is chosen such that for all $x \in U_\eta$, $P_{i,0}(x) = \psi'((\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$. Hence, it must be the case that for every $x \in U_\eta$, that one of the following holds:

$$P_{i,0}^{(0)}(x) \neq \widehat{P}_{i,0}^{(0)}(x); \qquad P_{i,0}^{(1)}(x) \neq \widehat{P}_{i,0}^{(1)}(x); \qquad \tilde{A}_i(x) \neq \widehat{A}_i(x); \qquad \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)) \neq \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x));$$
$$\tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)) \neq \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)); \qquad \widehat{P}_{i,0}(x) = \psi'((\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))).$$

The probability of this happening is at most the probability that a random $x \in F^m$ satisfies these conditions, which is at most $5 \cdot 4\varepsilon + \frac{\kappa md}{|F|} \leq 21\varepsilon_0$.

(c) For some $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0, 1\}$, $\widehat{P}_{i,j}^{(b)}$ does not obey Equation (7.3).

In other words, $\widehat{P}_{i,j}^{(b)}(\ldots, x_j, \ldots) \not\equiv \sum_{k=1}^{|H|} \widehat{P}_{i,j-1}^{(b)}(\ldots, h_j, \ldots) x_i^k$. Then, for this i, j, the ZERO PROPAGATION TEST rejects unless a random axis parallel line $\mathcal{L}$ is chosen such that both

$P^{(b)}_{i,j}|_{\mathcal{L}}$ and $P^{(b)}_{i,j-1}|_{\mathcal{L}}$ are polynomials of degree at most κd and for every $x \in \mathcal{L}$, $P^{(b)}_{i,j}(\ldots, x, \ldots) = \sum_{k=0}^{|H|-1} P^{(b)}_{i,j-1}(\ldots, h_k, \ldots)x^k$. Suppose we have that for all $x \in \mathcal{L}$, $P^{(b)}_{i,j}(x) = \widehat{P}^{(b)}_{i,j}(x)$ and $P^{(b)}_{i,j-1}(x) = \widehat{P}^{(b)}_{i,j-1}(x)$. Then, it must be the case that for all $x \in \mathcal{L}$, $\widehat{P}^{(b)}_{i,j}(\ldots, x, \ldots) = \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\ldots, h_k, \ldots)x^k$. Since the axis-parallel lines cover F^m uniformly, the probability of this occurring is at most the probability of a random $x \in F^m$ satisfying this condition which is at most $\frac{\kappa md}{c_F} \leq \varepsilon$. The probability that that both $P^{(b)}_{i,j}|_{\mathcal{L}}$ and $P^{(b)}_{i,j-1}|_{\mathcal{L}}$ are polynomials of degree κd and either $P^{(b)}_{i,j}|_{\mathcal{L}} \neq \widehat{P}^{(b)}_{i,j}|_{\mathcal{L}}$ or $P^{(b)}_{i,j-1}|_{\mathcal{L}} \neq P^{(b)}_{i,j-1}|_{\mathcal{L}}$ is at $2 \cdot 4\varepsilon/(1-\varepsilon_0) \leq 9\varepsilon_0$, since $P^{(b)}_{i,j}$ and $P^{(b)}_{i,j-1}$ are 4ε-close to $\widehat{P}^{(b)}_{i,j}$ and $\widehat{P}^{(b)}_{i,j-1}$ respectively, and any two distinct polynomials of degree κmd disagree on at least a $1 - \kappa md/|F| \geq 1 - \varepsilon_0$ fraction of points.Hence, the ZERO PROPAGATION TEST accepts with probability at most $10\varepsilon_0$.

We thus have that the verifier accepts with probability at most $\max\{1-\varepsilon, 5\varepsilon_0, 21\varepsilon_0, 10\varepsilon_0\} = 1-\varepsilon$. ∎

Proof (of Theorem 7.1.1): Theorem 7.1.1 is proved using the PCP–VERIFIER defined in this section setting $\lambda = 1/c \log n$. Step 1 of the verifier reduces the instance C of CKTSAT to an instance $\langle 1^{n'}, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CKTSAT. We have from Proposition 7.3.3 that $n' = \Theta(n)$ and $l = O(\log n)$ where n is the size of the input circuit C. Setting $n = n'$ in Proposition 7.5.1, we have that the randomness, query and decision complexity of the verifier are as claimed in Theorem 7.1.1. The soundness of the verifier follows from Lemma 7.5.2. ∎

CHAPTER 8

A randomness-efficient PCP of proximity

8.1 Introduction

In this chapter, we modify the PCP for CIRCUIT SATISFIABILITY (constructed in Chapter 7, Theorem 7.1.1) and construct a PCP of proximity for CIRCUIT VALUE while maintaining all the complexities. (Recall that the latter is stronger than the former, via Proposition 2.2.2.) We do so by adding a proximity test to the PCP–VERIFIER defined in Section 7.4. This new proximity test, as the name suggests, checks the closeness of the input to the satisfying assignment that is supposed to be encoded in the proof oracle (see Remark 7.3.4). This check is done by locally decoding a bit (or several bits) of the input from its encoding and comparing it with the actual input oracle.

Theorem 8.1.1 *There exists universal constants c and $0 < \varepsilon < 1$ such that the following holds for all $n, m \in \mathbb{Z}^+$ and $0 < \delta < 1$ such that $m \leq \log n/\mathrm{loglog} n$ and $n^{1/m} \geq m^{cm}/\delta^3$. There exists a PCP of proximity for* CIRCUIT VALUE *(for circuits of size n) with the following parameters*

- *randomness $\left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O(\log(1/\delta))$,*
- *query complexity $q = O(m^2 n^{1/m} \log^2 n)$ and decision complexity $d = \widetilde{O}(q)$,*
- *perfect completeness,*
- *the verifier has soundness error $1 - \varepsilon$ for proximity parameter δ.*

8.2 Proof of Theorem 8.1.1

Recall that the PCPP–VERIFIER is supposed to work as follows: The verifier is given explicit access to a circuit C with n gates on k input bits and oracle access to the input w in the form of an input oracle $W : [k] \to \{0, 1\}$. The verifier should accept W with probability 1 if it is a satisfying assignment and accept it with probability at most $1 - \varepsilon$ if it is δ-far from any satisfying assignment.

For starters, we assume that $k \geq n/5$. In other words, the size of the input w is linear in the size of the circuit C. The reason we need this assumption is that we wish to verify the proximity of w to a satisfying assignment, but our proofs encode the assignment to all n gates of the circuit, thus it better be the case that w is a non-negligible fraction of the circuit. This assumption is not a major restriction, because if this is not the case we work with the circuit C' and input w' which are as follows: For $t = \lceil n/k \rceil$, C' is a circuit with $n' = n + 3tk$ gates on $k' = tk$ input bits such that $C'(w') = 1$ iff $w' = w^t$ for some w such that $C(w) = 1$; that is, C' checks if its input is t copies of some satisfying assignment of C. (It can be verified that C' can indeed be implemented on a circuit of size $n + 3tk$ over the full binary basis.) We choose t such that $k' \geq n'/10$. However, note that the input oracle W cannot be altered. So the verifier emulates the input w' using the original input oracle $W : [k] \to \{0, 1\}$ in the straight-forward manner. Whenever it wants to read the i-th bit of the w', it queries the $(((i - 1) \bmod k) + 1)$-th bit of w.

Remark 8.2.1 *The above transformation from (C, w) to (C', w') is a generic one that increases the length of the input oracle compared to the proof oracle. The circuit C' checks that w' is a repetition codeword in order to maintain the distance features of C; that is, if w is δ-far from the set of satisfying assignments of C then $w' = w^t$ is also δ-far from the satisfying assignments of C'.*

As in the case of the PCP–VERIFIER described in Section 7.4, the PCPP–VERIFIER is constructed by reducing the input circuit C, an instance of CKTSAT, using parameter m, to an instance $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CKTSAT. The proof oracle for the PCPP–VERIFIER is the same as that of the PCP–VERIFIER (i.e., the proof oracle consists of a sequence of functions $\tilde{A}_i : F^m \to F, i = 0, \ldots, l-1$ and $P_{i,j} : F^m \to F^2, i = 0, \ldots, l-1, j = 0, \ldots, m$ where $l = 5 \log n$).

Recall that the function $\tilde{A}_0 : F^m \to F$ is supposed to contain within it an assignment (See Remarks 7.2.6,7.3.4). Let $I \subseteq H^m \subset F^m$ be the set of locations in F^m that contain the assignment. The PCPP–VERIFIER in addition to the tests of the PCP–VERIFIER performs the following PROXIMITY TEST to check if the assignment given by $\tilde{A}_0|_I$ matches with the input oracle W. Specifically:

PCPP–VERIFIER$^{W;\ \tilde{A}_i, P_{i,j}; i=0,\ldots,l-1; j=0,\ldots,m}_{m,\lambda,\delta}(C)$.

1. Run PCP–VERIFIER$^{W;\ \tilde{A}_i, P_{i,j}}_{m,\lambda}(C)$ and reject if it rejects.

   Let R be the random string generated during the execution of this step.

2. PROXIMITY TEST

Use random string R to determine a random canonical line $\mathcal{L}$ in F^m using the λ-biased set S_λ. Query oracle $\tilde{A}_0$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most $d = m \cdot |H|$. Query the input oracle W on all locations corresponding to those in $I \cap \mathcal{L}$ and reject if W disagrees with $\tilde{A}_0$ on any of the locations in $I \cap \mathcal{L}$.

By inspection, the proximity test increases the query and decision complexity by (even less than) a constant factor. For the randomness complexity, we note that the randomness is used only to generate a random canonical line (as in the PCP verifier). So the randomness complexity is $\log(|F|^{m-1} \cdot |S_\lambda|)$ as before. However, in order to prove soundness, we will need to assume not only that $\lambda \leq 1/c \log n$ for some constant c (as before), but also that $\lambda \leq \delta^3/m^{cm}$.[1] Thus, setting $\lambda = \min\{1/c \log n, \delta^3/m^{cm}\}$, the randomness complexity increases by at most $O(m \log m) + O(\log(1/\delta))$, as claimed in Theorem 8.1.1. Summarizing the above observations for future reference, we have the following proposition.

Proposition 8.2.2 *The randomness, query and decision complexities of the* PCPP–VERIFIER *are* $r = \left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O\left(\log\left(1/\delta\right)\right)$, $q = O(m^2 n^{1/m} \log^2 n)$ *and* $d = \tilde{O}(q)$ *respectively.*

It is straightforward to check perfect completeness of this verifier. To prove soundness, we observe that if the input W is δ-far from satisfying the circuit, then one of the following must happen: (1) the verifier detects an inconsistency in the proof oracle or (2) the input oracle does not match with the encoding of the input in the proof oracle. In case of the former, we prove soundness by invoking Lemma 7.5.2 while in the latter case, we prove soundess by analyzing the proximity test. These ideas are explained in detail in the following lemma which proves the soundness of the verifier.

Lemma 8.2.3 *There exists a constant c and a constant $\varepsilon > 0$ such that for all m, λ, δ satisfying $n \geq 8000|F|^{m-1}/\delta^3$, $\lambda \leq 1/c \log n$, and $\lambda \leq \delta/m^{cm}$, the following holds. If the input w given by the input oracle $W : [k] \to \{0, 1\}$ is δ-far from satisfying the circuit, then for any proof oracle the verifier rejects W with probability at least ε.*

Proof: Set ε to be the constant ε_0 in Lemma 7.5.2.

Case (i): $\tilde{A}_0$ is not 4ε-close to any polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Then by Lemma 7.5.2, we conclude that the verifier rejects with probability at least ε.

[1]Actually, for the proximity test we only need $\lambda \leq \delta/m^{cm}$, however to prove robustness of the proximity test (see Section 9.2) we require $\lambda \leq \delta^3/m^{cm}$.

Case (ii): $\tilde{A}_0$ is 4ε-close to some polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Since w is δ-far from any satisfying assignment, the assignment given by $\widehat{A}_0|_I$ must be at least δ-far from w. Let $B \subset F^m$ denote the set of locations in I where the assignment given by $\widehat{A}_0$ disagrees with w (i.e., $B = \{x \in I | \widehat{A}_0(x)$ disagrees with w at $x\}$). Hence, $|B|/|I| \geq \delta$. Since $|I| = k \geq n/5$, we have $|B| \geq \delta n/5$. Consider the following 2 events.

[Event I]: $\tilde{A}_0|_{\mathcal{L}}$ is 5ε-far from $\widehat{A}_0|_{\mathcal{L}}$.

By the Sampling Lemma (Lemma A.2.3) with $\mu = 4\varepsilon$ and $\zeta = \varepsilon$, this event occurs with probability at most $\left(\frac{1}{|F|} + \lambda\right) \cdot \frac{4\varepsilon}{\varepsilon^2} \leq \frac{1}{4}$ since $|F|, \frac{1}{\lambda} \geq 32/\varepsilon$.

[Event II]: $B \cap \mathcal{L} = \emptyset$.

Again by the Sampling Lemma (Lemma A.2.3) with $\mu = \zeta = \frac{|B|}{|F^m|}$, this event occurs with probability at most $\left(\frac{1}{|F|} + \lambda\right) \cdot \frac{|F^m|}{|B|} = \left(\frac{1}{|F|} + \lambda\right) \cdot \frac{5|F^m|}{\delta n} \leq \frac{1}{4}$, where the last inequality follows because $n \geq 8000|F|^{m-1}/\delta^3 \geq 40|F|^{m-1}/\delta$ and $\lambda \leq \delta/(40(c_F m^2)^m)$.

Suppose Event I does not occur. Then, if $\widehat{A}_0|_{\mathcal{L}} \neq \tilde{A}_0|_{\mathcal{L}}$, the PROXIMITY TEST rejects since then $\tilde{A}_0|_{\mathcal{L}}$ cannot be a polynomial of degree at most d as it is 5ε-close to the polynomial $\widehat{A}_0$ and hence cannot be closer to any other polynomial (as $5\varepsilon \leq \frac{1}{2}(1 - \frac{d}{|F|}) = \frac{1}{2}(1 - \frac{1}{c_F})$. Now if $\widehat{A}_0|_{\mathcal{L}} = \tilde{A}_0|_{\mathcal{L}}$ and Event II does not occur, then the PROXIMITY TEST detects a mismatch between the input oracle W and $\tilde{A}_0|_{\mathcal{L}}$. Hence, if both Event I and Event II do not occur, then the test rejects. Thus, the probability of the test accepting in this case is at most the probability of either Event I or Event II occurring which is at most $1/2$.

Thus, the probability that the verifier accepts is at most $\max\left\{1 - \varepsilon, \frac{1}{2}\right\} = 1 - \varepsilon$. This completes the proof of the lemma. ∎

Proof (of Theorem 8.1.1): Theorem 8.1.1 is proved using the PCPP–VERIFIER defined in this section setting $\lambda = \min\{1/c \log n, \delta^3/m^{cm}\}$. The randomness and decision (query) complexity follow from Proposition 8.2.2. The only fact to be verified is the soundness of the verifier. Suppose $n^{1/m} \geq m^{cm}/\delta^3$ for a suitably large constant c (i.e., as given in the hypothesis of Theorem 8.1.1). Then, $n^{1/m} \geq 8000(c_F m^2)^{m-1}/\delta^3$ or equivalently $n \geq 8000|F|^{m-1}/\delta^3$). Hence, Lemma 8.2.3 applies and we have that the verifier has soundness error $1 - \varepsilon$ for proximity parameter δ. This proves Theorem 8.1.1. ∎

CHAPTER 9

A randomness-efficient robust PCP of proximity

9.1 Introduction

In this chapter, we modify the PCP of proximity for CIRCUIT VALUE constructed in Chapter 8 to design a robust PCP of proximity, while essentially maintaining all complexities. Recall the definition of robustness: If the input oracle W is δ-far from a satisfying assignment, a "regular" PCPP verifier for most choices of its random coins rejects the input; that is, it observes an inconsistency in the input. A robust PCPP verifier, on the other hand, for most choices of its random coins not only notices an inconsistency in the input but also observes that a considerable portion of the input read by it has to be modified to remove this inconsistency.

Theorem 9.1.1 *There exists a universal constant c such that the following holds for all $n, m \in \mathbb{Z}^+$, $\delta, \gamma > 0$ satisfying $m \leq \log n/\text{loglog}n$ and $n^{1/m} \geq m^{cm}/\delta^3$: There exists a robust PCP of proximity for* CIRCUIT VALUE *(for circuits of size n) with the following parameters*

- *randomness $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$,*
- *query complexity $q = O((m^2 n^{1/m} \log^2 n)/\gamma)$ and decision complexity $d = \widetilde{O}(q)$,*
- *perfect completeness, and*
- *for every $\delta' > \delta$, the verifier has expected robustness*

$$\min\{\Omega(\gamma), (1 - \gamma) \cdot (\delta' - (\delta/2))\}$$

for proximity parameter δ'.

Note that the expected robustness can be very close to the proximity parameter δ' (provided $\delta \ll \delta' \ll \gamma \ll 1$), Thus, this robust PCPP is suitable for a large number of proof composition operations. In contrast, the robust PCPP described in Theorem 5.1.2 the robustness is always smaller than the proximity by a constant factor. Indeed, this is how the two theorems 6.1.1 (which is obtained from Theorem 9.1.1) and 5.1.2 are used in the proof of Theorem 10.2.1.

How Theorem 9.1.1 implies Theorem 6.1.1 Our main construct (i.e., Theorem 6.1.1) follows Theorem 9.1.1 by using the error-reduction lemma (Lemma 2.4.4). Specifically, replacing δ by $\delta'\gamma$ and using $\delta' < \gamma/c$, yields expected robustness of $\min\{\Omega(\gamma), (1-\gamma)\cdot(\delta' - \delta'\gamma/2)\}$, which is lower-bounded by $\rho' \triangleq (1-\gamma)^2 \cdot \delta'$. Applying Lemma 2.4.4 with a slackness parameter of $\gamma' \triangleq \gamma\rho'$ and $s = \gamma$, yields robust-soundness error of γ with robustness parameter of $\rho' - \gamma' = (1-\gamma)^3 \cdot \delta'$ for proximity parameter δ'. Using $\gamma \leq 1/2$, note that the randomness increases by an additive term of $O(\log(1/\gamma')) + O(\log(1/\delta')) = O(\log(1/\delta'))$, and the decision complexity increases by a multiplicative factor of $1/(\gamma \cdot (\gamma\rho')^2) = \mathrm{poly}(1/\delta')$. These modifications do not affect the general form of the corresponding complexities claimed in Theorem 6.1.1, and the latter follows (when substituting δ' for δ and γ by $\gamma/3$).

Overview of the proof of Theorem 9.1.1: We "robustify" our PCPP–VERIFIER in 3 steps. First we observe that a single execution of the verifier actually involves several tests (in fact $lm + 2l$ LOW-DEGREE TESTS, l EDGE-CONSISTENCY TESTS, lm ZERO PROPAGATION TESTS, l IDENTITY TESTS and a single PROXIMITY TEST). In the first step (Section 9.2), we observe that each of these tests is robust individually. In the second step (Section 9.3), we perform a "bundling" of the queries so that a certain set of queries can always be asked together. This bundling achieves robustness, albeit over a much a larger alphabet. In the final step (Section 9.4), we use a good error-correcting code to transform the "bundles" into regular bit-queries such that robustness over the binary alphabet is achieved.

9.2 Robustness of individual tests

For each possible random string R, the PCPP–VERIFIER performs several tests. More precisely, it performs $l(m+2)$ LOW-DEGREE TESTS, l EDGE-CONSISTENCY TESTS, lm ZERO PROPAGATION TESTS, l IDENTITY TESTS and a single PROXIMITY TEST. In this section, we prove that each of these test are robust individually. In other words, we show that when one of these tests fail, it fails in a "robust" manner; that is, a considerable portion of the input read by the test has to be modified for the test to pass.

First, some notation. We view functions $g, g' : F^m \to F$ as strings of length $|F|^m$ over the alphabet F, so their relative Hamming distance $\Delta(g, g')$ is simply $\Pr_x[g(x) \neq g'(x)]$. As before, let $I \subseteq H^m \subset F^m$ be the set of locations in F^m that contains the assignment.

Let $0 < \varepsilon < 1$ be a small constant to be specified later. As before, for $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0, 1\}$, let $\widehat{A}_i$ (resp., $\widehat{P}^{(b)}_{i,j}$) be the closest polynomials of degree md (resp., κmd) to $\tilde{A}_i$ and $P_{i,j}$ respectively. (If there is more than one polynomial, choose one arbitrarily.) The proof of the soundness of the PCPP–VERIFIER (see Sections 7 and 8) was along the following lines: If the input oracle $W : [k] \to \{0, 1\}$ is δ-far from satisfying the circuit, then one of the following must happen (changing ε by a factor of 2).

1. There exists a $i = 0, \ldots, l-1$ such that $\tilde{A}_i$ is 8ε-far from every degree md polynomial or there exists a $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0, 1\}$ such that $P^{(b)}_{i,j}$ is 8ε-far from every degree κmd polynomial. In this case, the LOW-DEGREE TEST detects the error with probability at least 2ε.

2. There exists $i = 0, \ldots, l-1$ and $b \in \{0, 1\}$, such that $\Delta(P^{(b)}_{i,m}, \widehat{P}^{(b)}_{i,m}) \leq 8\varepsilon$ and $\widehat{P}_{i,m} \not\equiv 0$. In this case, the IDENTITY TEST detects the error with probability at least $1 - 10\varepsilon$.

3. There exists $i = 0, \ldots, l-1$, $j = 1, \ldots, m$ and $b \in \{0, 1\}$ such that $\Delta(P_{i,j}, \widehat{P}_{i,j}) \leq 8\varepsilon$, $\Delta(P_{i,j-1}, \widehat{P}_{i,j-1}) \leq 8\varepsilon$, and $\widehat{P}_{i,j}(\ldots, x_j, \ldots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}(\ldots, h_k, \ldots) x_j^k$. In this case, the ZERO PROPAGATION TEST detects the error with probability at least $1 - 20\varepsilon$.

4. There exists a $i = 0, \ldots, l-1$ such that $\Delta(P^{(0)}_{i,0}, \widehat{P}^{(0)}_{i,0}) \leq 8\varepsilon$, $\Delta(P^{(1)}_{i,0}, \widehat{P}^{(1)}_{i,0}) \leq 8\varepsilon$, $\Delta(\tilde{A}_i, \widehat{A}_i) \leq 8\varepsilon$, $\Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \leq 8\varepsilon$, and $\widehat{P}_{i,0}(x) \not\equiv \psi'((\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$. In this case, the EDGE-CONSISTENCY TEST detects the error with probability at least $1 - 42\varepsilon$.

5. $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 8\varepsilon$ but W and $\widehat{A}_0|_I$ disagree on at least δ fraction of the points. In this case, the PROXIMITY TEST detects the error with probability at least $1/2$ (in Case I).

Claims 9.2.1 to 9.2.5 below strengthen the above analysis and show that one of the tests not only detects the error, but a significant portion of the input read by that test needs to be modified in order to make the test accept. More formally, recall that each of our tests T (randomly) generates a pair (I, D) where I is a set of queries to make to its oracle and D is the predicate to apply to the answers. For such a pair $(I, D) \leftarrow T$ and an oracle π, we define the *distance of* $\pi|_I$ *to* T to be the relative Hamming distance between $\pi|_I$ and the nearest satisfying assignment of D. Similarly, we say that π has *expected distance* ρ *from satisfying* T if the expectation of the distance of $\pi|_I$ to T over $(I, D) \overset{\text{R}}{\leftarrow} T$ equals ρ.

We then have the following claims about the robustness of the individual tests.

The robustness of the LOW-DEGREE TEST can be easily be infered from the analysis of the λ-biased low-degree test due to Ben-Sasson *et al.* [BSVW03] as shown below.

Claim 9.2.1 *The following holds for all sufficiently small $\varepsilon > 0$. If $A : F^m \to F$ (resp., $P : F^m \to F$) is 8ε-far from every polynomial of degree md (resp., degree κmd), then then the expected distance of A (resp. P) from satisfying the* LOW-DEGREE TEST *with degree parameter d (resp., κd) is at least 2ε.*

Proof: Recall that the LOW-DEGREE TEST chooses a random canonical line $\mathcal{L}$ and checks if $A|_{\mathcal{L}}$ is a univariate polynomial of degree d. For each canonical line $\mathcal{L}$, define $A_{\text{lines}}(\mathcal{L})$ to be the degree d univariate polynomial mapping $\mathcal{L} \to F$ having maximum agreement with A on $\mathcal{L}$, breaking ties arbitrarily. The distance of $A|_{\mathcal{L}}$ to satisfying LOW-DEGREE TEST is precisely $\Delta(A|_{\mathcal{L}}, A_{\text{lines}}(\mathcal{L}))$.

The low-degree test LINE–POINT–TEST$_{S_\lambda}$ of Ben-Sasson *et al.* [BSVW03] works as follows (see Section A.2 for more details): The test LINE–POINT–TEST$_{S_\lambda}$ has oracle access to a points-oracle $f : F^m \to F$ and a lines oracle g. It chooses a random canonical line $\mathcal{L}$ using the λ-biased set, queries the lines-oracle g on the line $\mathcal{L}$ and the points-oracle f on a random point x on $\mathcal{L}$. It accepts iff $g(\mathcal{L})$ agrees with f at x.

By inspection, the probability that LINE–POINT–TEST$_{S_\lambda}^{A,A_{\text{lines}}}$ rejects the points-oracle A and lines-oracle A_{lines} as defined above equals $\mathrm{E}_{\mathcal{L}}[\Delta(A|_{\mathcal{L}}, A_{\text{lines}}(\mathcal{L}))]$. By Theorem A.2.4, if A is 8ε-far from every degree md polynomial, then LINE–POINT–TEST$_{S_\lambda}^{A,A_{\text{lines}}}$ rejects with probability at least 2ε (for sufficiently small ε). (Recall that our parameters satisfy the conditions of Theorem A.2.4 for sufficiently large choices of the constants c and c_F.) Thus, A has expected distance 2ε from satisfying our LOW-DEGREE TEST, as desired.

The intuition behind the proofs of robustness of IDENTITY TEST, ZERO PROPAGATION TEST, and EDGE-CONSISTENCY TEST is as follows. The key point to be noted is that the checks made by each of these tests are in the form of polynomial identities. Hence, if the functions read by these tests are close to being polynomials, then it follows from the Schwartz-Zippel Lemma that the inputs read by these tests either satisfy these polynomial identities or are in fact far from satisfying them. We formalize this intuition and prove the robustness of IDENTITY TEST, EDGE-CONSISTENCY TEST, and ZERO PROPAGATION TEST in Claims 9.2.2, 9.2.3, and 9.2.4 respectively.

Claim 9.2.2 *The following holds for all sufficiently small $\varepsilon > 0$. If for some $i = 0, \ldots, l-1$ and $b \in \{0,1\}$, $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \le 8\varepsilon$ and $\widehat{P}_{i,m}^{(b)}(\cdot) \not\equiv 0$, then $P_{i,m}$ has expected distance at least $1 - 9\varepsilon$ from satisfying the* IDENTITY TEST.

Proof: The expected distance of $P_{i,m}$ from satisfying the IDENTITY TEST equals

$$\begin{aligned}
\mathrm{E}_{V_\eta}[\Delta(P_{i,m}|_{V_\eta}, 0)] &= \Delta(P_{i,m}, 0) \text{(since the } \{V_\eta\} \text{ are a partition)} \\
&\ge \Delta(\widehat{P}_{i,m}, 0) - \Delta(P_{i,m}, \widehat{P}_{i,m}) \\
&\ge \left(1 - \frac{\kappa md}{|F|}\right) - 8\varepsilon \text{(by Schwartz-Zippel and hypothesis)} \\
&\ge 1 - 9\varepsilon
\end{aligned}$$

Claim 9.2.3 *The following holds for all sufficiently small* $\varepsilon > 0$. *Suppose for some* $i = 0, \dots, l-1$, *we have* $\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \leq 8\varepsilon$, $\Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \leq 8\varepsilon$, $\Delta(\tilde{A}_i, \widehat{A}_i) \leq 8\varepsilon$, $\Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \leq 8\varepsilon$, *and* $\widehat{P}_{i,0}(\cdot) \not\equiv \psi'(\tilde{T}_i(\cdot), \widehat{A}_i(\cdot), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))$. *Then* $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}$ *has expected distance at least* $(1-41\varepsilon)/5$ *from satisfying the* EDGE-CONSISTENCY TEST.

Proof: Note that the distance of $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}|_{U_\eta}$ from satisfying the EDGE-CONSISTENCY TEST is at least $1/5$ times the the distance of $P_{i,0}(\cdot)|_{U_\eta}$ to the function $\psi'(\tilde{T}_i(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))|_{U_\eta}$ (Since for each point $x \in U_\eta$ where the latter two functions disagree, at least one of $P_{i,0}, A_i, A_{i+1} \circ \tilde{\Gamma}_{i,0}, A_{i+1} \circ \tilde{\Gamma}_{i,1}$ needs to be changed at x to make the test accept). As in the proof of Claim 9.2.2, we have:

$$\mathrm{E}_{U_\eta}[\Delta(P_{i,0}(\cdot)|_{U_\eta}, \psi'(\tilde{T}_i(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))|_{U_\eta})] \quad \geq \quad \left(1 - \frac{\kappa m d}{|F|}\right) - 5 \cdot 8\varepsilon \geq 1 - 41\varepsilon,$$

where the $(1 - \kappa m d/|F|)$ term corresponds to the distance if we replace all five functions with their corrected polynomials (e.g., $\widehat{P}_{i,0}$, $\widehat{A}_i$, $\widehat{A}_{i+1} \circ \tilde{\Gamma}_{i,0}$, $\widehat{A}_{i+1} \circ \tilde{\Gamma}_{i,1}$) and the $-5 \cdot 8\varepsilon$ accounts for the distance between each of the five functions and their corrected polynomials. Thus, the overall expected distance to satisfying the EDGE-CONSISTENCY TEST is at least $(1-41\varepsilon)/5$.

Claim 9.2.4 *The following holds for all sufficiently small* $\varepsilon > 0$. *Suppose for some* $i = 0, \dots, l-1$, $j = 1, \dots, m$, *and* $b \in \{0,1\}$, *we have* $\Delta(P_{i,j}^{(b)}, \widehat{P}_{i,j}^{(b)}) \leq 8\varepsilon$, $\Delta(P_{i,j-1}^{(b)}, \widehat{P}_{i,j-1}^{(b)}) \leq 8\varepsilon$, *and* $\widehat{P}_{i,j}^{(b)}(\dots, x_j, \dots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}^{(b)}(\dots, h_k, \dots) x_j^k$. *Then* $(P_{i,j}^{(b)}, P_{i,j-1}^{(b)})$ *has expected distance at least* $(1-19\varepsilon)/2$ *from satisfying the* ZERO PROPAGATION TEST.

Proof: Suppose that $\mathcal{L}$ is a jth axis-parallel line such that

$$\widehat{P}_{i,j}^{(b)}(\dots, x_j, \dots)|_{\mathcal{L}} \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}^{(b)}(\dots, h_k, \dots) x_j^k|_{\mathcal{L}},$$

Then in order for the ZERO PROPAGATION TEST to accept, either $P_{i,j}^{(b)}|_{\mathcal{L}}$ must be modified to equal a degree κd polynomial other than $\widehat{P}_{i,j-1}^{(b)}(\dots, x_j, \dots)|_{\mathcal{L}}$ or $P_{i,j-1}^{(b)}|_{\mathcal{L}}$ must be modified to equal a degree κd polynomial other than $\widehat{P}_{i,j-1}^{(b)}(\dots, x_j, \dots)|_{\mathcal{L}}$. (Recall that the ZERO PROPAGATION TEST checks that the said restrictions are in fact polynomials of degree κd.) This would require modifying $P_{i,j}^{(b)}|_{\mathcal{L}}$ (resp., $P_{i,j-1}^{(b)}|_{\mathcal{L}}$) in at least a $1 - \kappa d/|F| - \Delta(P_{i,j}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j}^{(b)}|_{\mathcal{L}})$ fraction (resp., $1 - \kappa d/|F| - \Delta(P_{i,j-1}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j-1}^{(b)}|_{\mathcal{L}})$ fraction) of points. This implies that the pair $(P_{i,j}^{(b)}|_{\mathcal{L}}, P_{i,j-1}^{(b)}|_{\mathcal{L}})$ would have to be modified in at least a

$$\frac{1}{2} \cdot \left(1 - \frac{\kappa d}{|F|} - \Delta(P_{i,j}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j}^{(b)}|_{\mathcal{L}}) - \Delta(P_{i,j-1}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j-1}^{(b)}|_{\mathcal{L}})\right)$$

fraction of points.

Thus the expected distance of $(P_{i,j}^{(b)}, P_{i,j-1}^{(b)})$ to satisfying the ZERO PROPAGATION TEST is at least

$$\begin{aligned}
&\frac{1}{2}\cdot \mathrm{E}_{\mathcal{L}}\left[1-\frac{\kappa d}{|F|}-\Delta(P_{i,j}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j}^{(b)}|_{\mathcal{L}})-\Delta(P_{i,j-1}^{(b)}|_{\mathcal{L}}, \widehat{P}_{i,j-1}^{(b)}|_{\mathcal{L}})\right] \\
&-\Pr_{\mathcal{L}}\left[\widehat{P}_{i,j}^{(b)}(\ldots, x_j, \ldots)|_{\mathcal{L}} \equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}^{(b)}(\ldots, h_k, \ldots) x_j^k|_{\mathcal{L}}\right] \\
\geq\ & \frac{1}{2}(1-\varepsilon-8\varepsilon-8\varepsilon)-\frac{\kappa d}{|F|} \\
\geq\ & \frac{1}{2}(1-19\varepsilon).
\end{aligned}$$

We are now left with analyzing the robustness the PROXIMITY TEST. Note that the input of the PROXIMITY TEST comes in two parts: (a) the restriction of A_0 to the line $\mathcal{L}$ and (b) the input W restricted to the line $\mathcal{L}$. Unlike earlier tests, we do not collate the robustness of these two parts of the input but express them separately. The robustness of the PROXIMITY TEST is proved by repeated applications of the Sampling Lemma (Lemma A.2.3).

Let $B \subset F^m$ denote the set of locations in I where the assignment given by $\widehat{A}_0$ disagrees with W (i.e., $B = \{x \in I | \widehat{A}_0(x)$ disagrees with W at $x\}$). Recall that $|I| = k \geq n/5$.

Claim 9.2.5 *There exists a constant c and a constant $\varepsilon > 0$ such that for all $m, \lambda, \delta, \delta'$ satisfying $n \geq 8000|F|^{m-1}/\delta^3$, $\lambda \leq 1/c\log n$, $\lambda \leq \delta^3/m^{cm}$, $\delta' > \delta$, the following holds. Suppose $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 1/4$ and the input oracle W is δ'-far from $\widehat{A}_0|_I$ (i.e., $|B|/|L| \geq \delta'$), then with probability at least $1-\delta/4$ (over the choice of the canonical line $\mathcal{L}$) either at least a ε-fraction of $A_0|_{\mathcal{L}}$ or at least a $(\delta'-\delta/4)$-fraction of $W|_{\mathcal{L}}$ needs to be changed to make the* PROXIMITY TEST *accept.*

This claim is the robust analogue of Lemma 8.2.3. Observe that the robustness of the verifier is expressed separately for the proof and input oracles. As expected, the robustness of the input oracle depends on the proximity parameter δ while that of the proof oracle is independent of δ.

Proof: Consider the following three events.

Event 1: $\Delta(\tilde{A}_0|_{\mathcal{L}}, \widehat{A}_0|_{\mathcal{L}}) \geq 1/3$.

By the Sampling Lemma (Lemma A.2.3) with $\mu = 1/4$ and $\zeta = 1/12$, this event occurs with probability at most $\left(\frac{1}{|F|}+\lambda\right)\cdot\frac{1/4}{(1/12)^2} \leq \frac{\delta}{12}$ since $|F| \geq (8000|F|^m)/(\delta^3 n) > (12^3/2)/\delta$ and $\lambda < 2\delta/12^3$.

Event 2: $\frac{|I\cap\mathcal{L}|}{|\mathcal{L}|} > \left(1+\frac{\delta}{8}\right)\cdot\frac{|I|}{|F^m|}$.

Again by the Sampling Lemma (Lemma A.2.3) with $\mu = |I|/|F^m| \geq \frac{n}{5|F|^m}$ and $\zeta = \frac{\delta\mu}{8}$, this event occurs with probability at most

$$\left(\frac{1}{|F|}+\lambda\right)\cdot\frac{8^2}{\delta^2\mu}=\left(\frac{1}{|F|}+\lambda\right)\cdot\frac{320|F|^m}{\delta^2 n}\leq\frac{\delta}{12},$$

where the last inequality follows from the fact that $n \geq 24\cdot 320\cdot|F|^{m-1}/\delta^3$ and $\lambda \leq \delta^3/(24\cdot 320(c_F m^2)^m)$.

Event 3: $\frac{|B\cap\mathcal{L}|}{|\mathcal{L}|} < \frac{|B|}{|F^m|} - \frac{\delta}{8}\cdot\frac{|I|}{|F^m|}$.

Again by the Sampling Lemma (Lemma A.2.3) with $\mu = |B|/|F^m| = \frac{\delta' n}{5|F|^m}$ and $\zeta = \frac{\delta n}{40|F|^m}$, this event occurs with probability at most

$$\left(\frac{1}{|F|}+\lambda\right)\cdot\frac{\mu}{\zeta^2}\leq\left(\frac{1}{|F|}+\lambda\right)\cdot\frac{320|F|^m}{\delta^2 n}\leq\frac{\delta}{12}.$$

Hence, the probability that at least one of the three events occurs is at most $\delta/4$.

Now, suppose none of the three events occur. We then get that

$$\frac{|B\cap\mathcal{L}|}{|I\cap\mathcal{L}|}\geq\frac{|B|-\delta|I|/8}{(1+\delta/8)|I|}=\frac{\delta'-\delta/8}{1+\delta/8}\geq\delta'-\delta/4.$$

Now for the PROXIMITY TEST to accept the pair $(\tilde{A}_0|_{\mathcal{L}}, W\cap\mathcal{L})$, either we must change $\tilde{A}_0|_{\mathcal{L}}$ to a polynomial other than $\widehat{A}_0|_{\mathcal{L}}$ or correct the input for all $x \in B\cap\mathcal{L}$. The former requires us to change at least $(1-\frac{d}{|F|}-1/3)\geq 1/2$ fraction of the points of $A_0|_{\mathcal{L}}$ while the latter requires us to change at least $\delta'-\delta/4$-fraction of the input read (i.e., the input oracle W restricted to the line $\mathcal{L}$). This proves the claim. ∎

9.3 *Bundling*

In Section 9.2, we showed that each of the tests performed by the PCPP verifier is individually robust. However, we need to show that the conjunction of all these tests is also robust. This is not true for the PCPP verifier in its present form for the following reason: Suppose the input oracle W is δ-far from satisfying the circuit. We then know that one of the tests detects this fact with non-negligible probability. Moreover as seen in Section 9.2, this test is robust. However, since this test is only one of the $O(lm)$ tests being performed by the verifier, the oracle bits read by this test comprise a small fraction of the total query complexity of the verifier. For instance, the number of bits read by a single LOW-DEGREE TEST is about $1/lm$ times the query complexity. This causes the robustness of the verifier to drop by a factor of at least lm. Note that the issue here is not the fact that the verifier performs different types of tests (i.e., LOW-DEGREE TEST, IDENTITY TEST, ZERO PROPAGATION TEST, etc) but rather that it performs many instances of each test and that the soundness analysis only guarantees that one of these test instances rejects (robustly). This is not sufficient to make the verifier robust.

For this purpose, we "bundle" the various functions in the proof oracle so that the inputs required for the several test instances can be read together. This maintains the robustness of the individual tests, albeit over a larger alphabet. To understand this "bundling", let us assume for the present that the only type of tests that the verifier performs is the LOW-DEGREE TEST. There exists a natural bundling in this case. Instead of $l(m+2)$ different oracles $\{\tilde{A}_i\}$ and $\{P_{i,j}\}$, we have one oracle Π which bundles together the data of all these oracles. The oracle $\Pi : F^m \to F^{l\cdot(2m+3)}$ is supposed to satisfy $\Pi(x) = (\tilde{A}_0(x), \ldots, \tilde{A}_{l-1}(x), P_{0,0}(x), \ldots, P_{l-1,m}(x))$ for all $x \in F^m$. It can now be easily checked that over this proof oracle, the conjunction of all the LOW-DEGREE TESTs is robust (over alphabet $F^{l\cdot(2m+3)}$) with the same soundness and robustness parameters as a single LOW-DEGREE TEST(over alphabet F). However, this natural bundling does not lend itself to the other tests performed by the PCPP verifier (namely, ZERO PROPAGATION TEST, and EDGE-CONSISTENCY TEST). Next, we provide an alternate bundling and massage our verifier slightly to work with this bundling.

First for some notation. As mentioned earlier, we will be able to prove robustness of the verifier via bundling, however over a larger alphabet. This large alphabet will be $\Sigma = F^{l+2l\cdot(m+1)}$. Unlike before, the proof oracle for the robust PCPP verifier will consist of only one function $\Pi : F^m \to \Sigma$. The robust PCPP verifier simulates the PCPP verifier as follows: To answer the queries of the PCPP verifier, the robust verifier bundles several queries together, queries the new proof oracle Π and then unbundles the answer to obtain the answers of the queries of the original PCPP verifier. For convenience, we index the $l + 2l \cdot (m+1)$ coordinates of $\Sigma = F^{l+2l\cdot(m+1)}$ as follows: The first l coordinates are indexed by a single index i ranging from 0 to $l-1$, while the remaining $2l \cdot (m+1)$ are indexed by a triplet of indices (i, j, b) where i ranges over $0, \ldots, l-1$, j ranges over $0, \ldots, m$ and $b \in \{0, 1\}$. Let $S : F^m \to F^m$ denote the (linear) transformation that performs one cyclic shift to the right; that is, $S(x_0, \ldots, x_{m-1}) = (x_{m-1}, x_0, \ldots, x_{m-2})$. The bundling of the proof oracles $\tilde{A}_i$'s and $P_{i,j}$'s by the modified proof oracle Π is as follows:

$$\forall x \in F^m, \qquad \begin{cases} \Pi(x)_i = \tilde{A}_i\left(S^{\lfloor \frac{i}{h} \rfloor}(x)\right) & i = 0, \ldots, l-1 \\ \Pi(x)_{(i,j,b)} = P_{i,j}^{(b)}\left(S^{j+\lfloor \frac{i}{h} \rfloor}(x)\right) & i = 0, \ldots, l-1;\ j = 0, \ldots, m \text{ and } b \in \{0,1\} \end{cases} \tag{9.1}$$

where $h = \log|H| = \log n/m$. Note that the size of the new proof oracle Π is exactly equal to the sum of the size of the oracles $\tilde{A}_i$'s and $P_{i,j}$'s.

We now state how the robust verifier performs the unbundling and the individual tests. We consider each step of the PCPP verifier and present their robust counterparts.

The first steps of the PCP–VERIFIER (and PCPP–VERIFIER) are independent of the proof oracle and are performed as before. That is, the robust verifier, as before, reduces the CKTSAT instance to an instance $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CKTSAT, sets $d = m \cdot |H|$, and generates a random string R of length $\log(|S_\lambda| \cdot |F|^{m-1})$. The remaining steps are proof-oracle dependent and we will discuss each of them in detail.

Proximity Test. For the proximity test, the only portion of the proof oracle that we require is the portion containing $\tilde{A}_0$. Since $\Pi(x)_0$ is $\tilde{A}_0 \circ S^{\lfloor \frac{0}{h} \rfloor}(x) = \tilde{A}_0(x)$, the ROBUST PROXIMITY TEST can easily be describes as follows:

ROBUST PROXIMITY TEST$^{W;\,\Pi}(R)$

> Use random string R to determine a random canonical line $\mathcal{L}$ in F^m using the λ-biased set S_λ. Query oracle Π on all points along the line $\mathcal{L}$. Unbundle $\Pi(\mathcal{L})$ to obtain the values of $\tilde{A}_0$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most d. Query the input oracle W on all locations corresponding to those in $I \cap \mathcal{L}$ and reject if W disagrees with $\tilde{A}_0$ on any of the locations in $I \cap \mathcal{L}$.

Low-Degree Test. We note that the distance of the polynomial $\tilde{A}_i : F^m \to F$ to being degree k (for any $k \in \mathbb{Z}^+$) is exactly the same as that of $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor} : F^m \to F$ since $S^{\lfloor \frac{i}{h} \rfloor}$ is an invertible linear transformation. Hence, it is sufficient if we check that $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ is low-degree. The case with the $P_{i,j}^{(b)}$'s is similar. Thus, the new ROBUST LOW-DEGREE TEST can be described as follows:

ROBUST LOW-DEGREE TEST$^{\Pi}(R)$

```
Use random string $R$ to determine a random canonical line $\mathcal{L}$ in $F^m$ using the $\lambda$-biased
set $S_\lambda$.
Query the oracle $\Pi$ on all points along the line $\mathcal{L}$.
For $i = 0, \ldots, l-1$,
    Unbundle $\Pi(\mathcal{L})$ to obtain the values of $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$ and
    reject if the restriction $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ to $\mathcal{L}$ is not a polynomial of degree at most $d$.
For $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0,1\}$,
    Unbundle $\Pi(\mathcal{L})$ to obtain the values of $P_{i,j}^{(b)} \circ S^{j+\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$
    and reject if the restriction of $P_{i,j}^{(b)} \circ S^{j+\lfloor \frac{i}{h} \rfloor}$ to $\mathcal{L}$ is not a polynomial of degree at most
    $\kappa d$.
```

Thus, effectively we are testing $\tilde{A}_i$ (respectively $P_{i,j}$) using the line space $S^{\lfloor \frac{i}{h} \rfloor} \circ S_\lambda$ (respectively $S^{j+\lfloor \frac{i}{h} \rfloor} \circ S_\lambda$).

Identity Test. In the case of the IDENTITY TEST, we observe that $P_{i,m}^{(b)}$ vanishes on F^m iff $P_{i,m}^{(b)} \circ S^{m+\lfloor \frac{i}{h} \rfloor}$ vanishes on F^m. Recall that we were allowed to use arbitrary partitions of the space F^m. The set of random 1st axis-parallel lines is one such partition and we use this partition.

```
ROBUST IDENTITY TEST$^\Pi(R)$

    Use random string $R$ to determine a random 1st axis-parallel line in $F^m$ of the form $\mathcal{L} = (X, a_1, \ldots, a_{m-1})$. Query the oracle $\Pi$ on all points along the line $\mathcal{L}$.
    For $i = 0, \ldots, l-1$ and $b \in \{0,1\}$,
        Unbundle $\Pi(\mathcal{L})$ to obtain the values of $P^{(b)}_{i,m} \circ S^{m+\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$ and reject if any of these is non-zero.
```

Edge Consistency Test. For any $x \in F^m$, we say that $P_{i,0}$ is *well-formed* at x if the Equation (7.2) is satisfied for this x. The EDGE-CONSISTENCY TEST verifies that $P_{i,0}$ is well-formed for all $x \in U_\eta$ and $i = 0, \ldots, l-1$. This was done earlier by reading the values of $P_{i,0}, \tilde{A}_i, \tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,0} = \tilde{A}_{i+1}$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}$ for all $x \in U_\eta$.

Let $\mathcal{L}$ be a random 1st axis-parallel line. The robust version of this test checks that $P_{i,0}$ is well-formed for all points on $S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$. Consider any $x = (x_0, \ldots, x_{m-1}) \in \mathcal{L}$. To verify that $P_{i,0}$ is well-formed at $S^{\lfloor \frac{i}{h} \rfloor}(x)$, the verifier needs the values $P_{i,0}(S^{\lfloor \frac{i}{h} \rfloor}(x))$, $\tilde{A}_i(S^{\lfloor \frac{i}{h} \rfloor}(x))$, $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$. We will show that all these values can be obtained from unbundling the value of Π on $\mathcal{L}$ and $S^{-1}(\mathcal{L})$. Clearly, the values $P_{i,0}(S^{\lfloor \frac{i}{h} \rfloor}(x))$ and $\tilde{A}_i(S^{\lfloor \frac{i}{h} \rfloor}(x))$ can be obtained from unbundling the value of Π at x. The other two values that we require are $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$. We first show that $\tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = S^{\lfloor \frac{i}{h} \rfloor}(x')$ for $x' = (x_0 + e_{(i \bmod h)}, x_1, \ldots, x_{m-1}) \in \mathcal{L}$ (recall that $\{e_0, \ldots, e_{f-1}\}$ are a basis for F over GF(2) and $\{e_0, \ldots, e_{h-1}\}$ span $H \subset F$). For this purpose, we first recall the definition of $\tilde{\Gamma}_{i,1}$: $\tilde{\Gamma}_{i,1}(z_0, \ldots, z_{m-1}) = (z_0, \ldots, z_{t-1}, z_t + e_u, z_{t+1}, \ldots, z_{m-1})$ where $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$. Furthermore, since S^m is the identity map, we have that $S^{\lfloor \frac{i}{h} \rfloor \bmod m} = S^{\lfloor \frac{i}{h} \rfloor}$. With these observations, we have the following:

$$\begin{aligned}
\tilde{\Gamma}_{i,1}\left(S^{\lfloor \frac{i}{h} \rfloor}(x)\right) &= \tilde{\Gamma}_{i,1}\left(S^{\lfloor i/h \rfloor \bmod m}(x)\right) \\
&= \tilde{\Gamma}_{i,1}\left(S^{\lfloor i/h \rfloor \bmod m}(x_0, \ldots, x_{m-1})\right) \\
&= S^{\lfloor i/h \rfloor \bmod m}\left(x_0 + e_{(i \bmod h)}, x_1, \ldots, x_{m-1}\right) \\
&= S^{\lfloor \frac{i}{h} \rfloor}(x')
\end{aligned}$$

Now, $S^{\lfloor \frac{i+1}{h} \rfloor}$ is either $S^{\lfloor \frac{i}{h} \rfloor}$ or $S^{\lfloor \frac{i}{h} \rfloor + 1}$ depending on the value of i. Suppose $S^{\lfloor \frac{i+1}{h} \rfloor} = S^{\lfloor \frac{i}{h} \rfloor}$. We then have that $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(x))$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = \tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x')) = \tilde{A}_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(x'))$. Both these values can be obtained by unbundling the value of Π on $\mathcal{L}$ (since both x and x' lie on $\mathcal{L}$). In the other case, where $S^{\lfloor \frac{i+1}{h} \rfloor} = S^{\lfloor \frac{i}{h} \rfloor + 1}$, we have $A_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(S^{-1}x))$ and $A_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x')) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(S^{-1}x'))$. These values can be obtained by unbundling the value of Π on $S^{-1}(\mathcal{L})$. Thus, to check that $P_{i,0}$ is well-formed for all points on $S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$, it suffices if the verifier queries Π on all points on $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

ROBUST EDGE-CONSISTENCY TEST$^{\Pi}(R)$

Use the random string R to determine a random 1st axis-parallel line in F^m of the form $\mathcal{L} = (X, a_2, \ldots, a_m)$. Query the oracle Π along all points in the lines $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

For $i = 0, \ldots, l-1$,

For all $x \in S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$, reject if $P_{i,0}$ is not well-formed at x. [Note that all the values required for this verification can be obtained by unbundling $\Pi(\mathcal{L})$ and $\Pi(S^{-1}(\mathcal{L}))$.]

Zero Propagation Test. For each $i = 0, \ldots, l-1$ and $b \in \{0,1\}$, the ZERO PROPAGATION TEST checks that $P_{i,0}^{(b)}$ vanishes on H^m by verifying that Equation (7.3) is satisfied for all $j = 1, \ldots, m-1$ (we also need to check that $P_{i,m}^{(b)} \equiv 0$, however this is taken care by the IDENTITY TEST). Since $S(H^m) = H^m$, checking if $P_{i,0}^{(b)}$ vanishes on H^m is equivalent to checking if $P_{i,0}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}$ vanishes on H^m. Hence, we can perform the zero propagation on the polynomials $P_{i,0}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}; i = 0, \ldots, l-1, b \in \{0,1\}$ instead of the polynomials $P_{i,0}^{(b)}; i = 0, \ldots, l-1, b \in \{0,1\}$. In other words, we need to verify the following equation instead of Equation (7.3).

$$P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor} \left(\underbrace{x_1, \ldots, x_{j-1}}, x_j, \underbrace{x_{j+1}, \ldots, x_m} \right) = \sum_{k=0}^{|H|-1} P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor} \left(\underbrace{x_1, \ldots, x_{j-1}}, h_k, \underbrace{x_{j+1}, \ldots, x_m} \right) x_j^k, \quad \forall (x_1, \ldots, x_m) \in F^m \tag{9.2}$$

This equation can be further rewritten in terms of the cyclic shift S as follows:

$$P_{i,j}^{(b)} \left(S^{\lfloor \frac{i}{h} \rfloor + j - 1}(x_1, x_2, \ldots, x_m) \right) = \sum_{k=0}^{|H|-1} P_{i,j-1}^{(b)} \left(S^{\lfloor \frac{i}{h} \rfloor + j - 1}(h_k, x_2, \ldots, x_m) \right) x_1^k, \qquad \forall (x_1, \ldots, x_m) \in F^m \tag{9.3}$$

This helps us to rewrite the ZERO PROPAGATION TEST with bundling as follows:

ROBUST ZERO PROPAGATION TEST$^{\Pi(R)}$

Use random string R to determine a random 1st axis-parallel line in F^m of the form $\mathcal{L} = (X, a_2, \ldots, a_m)$. Query the oracle Π along all points in the lines $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

For $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0,1\}$

Unbundle $\Pi(\mathcal{L})$ to obtain the value of $P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ on all points along the line $\mathcal{L}$. Similarly, unbundle $\Pi(S^{-1}(\mathcal{L}))$ to obtain the value of $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j}$ on all points along the line $S^{-1}(\mathcal{L})$ (Equivalently, this is the value of $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ on all points along the line $\mathcal{L}$). Reject either if the restriction of $P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ or $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ to $\mathcal{L}$ is not a polynomial of degree at most κd or if any of the points on the line $\mathcal{L}$ violate Equation (9.3).

The integrated robust verifier. Having presented the robust version of each of the tests, the integrated robust verifier is as follows.

ROBUST-PCPP–VERIFIER$^{W;\,\Pi}_{m,\lambda,\delta}(C)$.

1. Using Proposition 7.3.3, reduce the instance C of CKTSAT, using parameter m, to an instance $\langle 1^n, 1^m, F, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CKTSAT, and set $d = m \cdot |H|$.
   We let $S_\lambda \subset F^m$ be a λ-biased set of size at most $\left(\frac{\log|F|^m}{\lambda}\right)^2$ [AGHP92].
2. Choose a random string R of length $\log(|S_\lambda| \cdot |F|^{m-1})$. [Note: We reuse R in all tests, but only the LOW-DEGREE TEST utilizes the full length of R.]
3. Run ROBUST LOW-DEGREE TEST$^{\Pi}(R)$.
4. Run ROBUST EDGE-CONSISTENCY TEST$^{\Pi}(R)$.
5. Run ROBUST ZERO PROPAGATION TEST$^{\Pi}(R)$.
6. Run ROBUST IDENTITY TEST$^{\Pi}(R)$.
7. Run ROBUST PROXIMITY TEST$^{W;\Pi}(R)$.

Reject if any of the above tests reject.

The randomness of the ROBUST-PCPP–VERIFIER is exactly the same as before whereas the query complexity and decision complexity increase by a constant factor[1].

Proposition 9.3.1 *The randomness, query and decision complexities of the* ROBUST-PCPP–VERIFIER *are* $r = \left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O\left(\log\left(1/\delta\right)\right)$, $q = O(m^2 n^{1/m} \log^2 n)$ *and* $d = \tilde{O}(q)$ *respectively.*

It is straightforward to check perfect completeness of this verifier.

Robustness analysis of the integrated verifier. To state the robust soundness, it is useful for us to separate the robustness wrt the input oracle and wrt the proof oracle. Let $W : [k] \to \{0,1\}$ be the input oracle and Π the proof oracle. For every sequence of coin tosses R (and a given setting of parameters), let $\Delta^{W,\Pi}_{\text{inp}}(R)$ (resp., $\Delta^{W,\Pi}_{\text{pf}}(R)$) denote the fraction of the bits read from W (resp. Π) that would need to be changed to make the ROBUST-PCPP–VERIFIER accept on coin tosses R. Then the following lemma states the robustness property of our verifier.

Lemma 9.3.2 *There are constants* $c \in \mathbb{Z}^+$ *and* $\rho > 0$ *such the following holds for every* $n, m \in \mathbb{Z}^+$, $\delta, \delta' > 0$ *satisfying* $m \leq \log n/\text{loglog}n$, $n^{1/m} \geq m^{cm}/\delta^3$, $\lambda \leq \min\{1/c\log n, \delta^3/m^{cm}\}$, $\delta' > \delta$. *If* W *is* δ'*-far from satisfying the circuit, then for any proof oracle* $\Pi : F^m \to \Sigma$, *either* $\mathrm{E}_R[\Delta^{W,\Pi}_{\text{pf}}(R)] \geq \rho$ *or* $\mathrm{E}_R[\Delta^{W,\Pi}_{\text{inp}}(R)] \geq \delta' - \delta/2$.

[1]Though the new proof oracle returns elements of Σ and not bits, we express the query complexity as the number of bits read by the verifier rather than the number of symbols (i.e., elements of $|\Sigma|$) to maintain consistency across calculating the query complexity into the proof and input oracles.

Proof: Unbundle the proof oracle Π to obtain the functions $\tilde{A}_i$ and $P_{i,j}$ using Equation (9.1). Consider the action of the PCPP–VERIFIER (i.e., the non-robust verifier) on the proof oracles $\tilde{A}_i, P_{i,j}$ and input oracle W.

Let ε be a sufficiently small constant such that the Claims 9.2.1–9.2.5 hold. Suppose W is δ'-far from satisfying the circuit. We then know that one of the following holds and that the corresponding test instance of the PCPP–VERIFIER rejects its input robustly (see Claims 9.2.1 to 9.2.5).

1. There exists a $i = 0, \ldots, l-1$ such that $\tilde{A}_i$ is 8ε-far from every degree md polynomial or there exists a $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0,1\}$ such that $P_{i,j}^{(b)}$ is 8ε-far from every degree κmd polynomial. In this case,the expected distance of $\tilde{A}_i$ (or resp. $P_{i,j}^{(b)}$) from satisfying the LOW-DEGREE TEST with degree parameter d (resp., κd) is at least 2ε (Claim 9.2.1).

2. There exists $i = 0, \ldots, l-1$ and $b \in \{0,1\}$, such that $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \leq 8\varepsilon$ and $\widehat{P}_{i,m} \not\equiv 0$. In this case, $P_{i,m}$ has expected distance at least $1-9\varepsilon$ from satisfying the IDENTITY TEST (Claim 9.2.2).

3. There exists a $i = 0, \ldots, l-1$ such that $\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \leq 8\varepsilon$, $\Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \leq 8\varepsilon$, $\Delta(\tilde{A}_i, \widehat{A}_i) \leq 8\varepsilon$, $\Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \leq 8\varepsilon$, and $\widehat{P}_{i,0}(x) \not\equiv \psi'((\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$. In this case, $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}$ has expected distance at least $(1-41\varepsilon)/5$ from satisfying the EDGE-CONSISTENCY TEST (Claim 9.2.3).

4. There exists $i = 0, \ldots, l-1$, $j = 1, \ldots, m$ and $b \in \{0,1\}$ such that $\Delta(P_{i,j}, \widehat{P}_{i,j}) \leq 8\varepsilon$, $\Delta(P_{i,j-1}, \widehat{P}_{i,j-1}) \leq 8\varepsilon$, and $\widehat{P}_{i,j}(\ldots, x_j, \ldots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}(\ldots, h_k, \ldots) x_j^k$. In this case, $(P_{i,j}^{(b)}, P_{i,j-1}^{(b)})$ has expected distance at least $(1-19\varepsilon)/2$ from satisfying the ZERO PROPAGATION TEST (Claim 9.2.4).

5. $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 8\varepsilon$ but W and $\widehat{A}_0|_I$ disagree on at least δ fraction of the points. In this case, with probability at least $1-\delta/4$ (over the choice of the canonical line $\mathcal{L}$) either at least a ε-fraction of $A_0|_{\mathcal{L}}$ or at least a $(\delta' - \delta/4)$-fraction of $W|_{\mathcal{L}}$ needs to be changed to make the PROXIMITY TEST accept (Claim 9.2.5).

   This implies that either A_0 has expected distance $(1-\delta/4)\varepsilon \geq \varepsilon/2$ or W has expected distance $(1-\delta/4)(\delta' - \delta/4) \geq (\delta' - \delta/2)$ from satisfying the PROXIMITY TEST.

For instance, lets us assume $\tilde{A}_0$ is 8ε-far from being low degree so the LOW-DEGREE TEST rejects it robustly; that is, for a random canonical line $\mathcal{L}$, the expected distance of $\tilde{A}_0|_{\mathcal{L}}$ from satisfying the LOW-DEGREE TEST is at least 2ε. Recall from Equation (9.1) that $\tilde{A}_0(x)$ is one of the co-ordinates in the bundled $\Pi(x)$. Hence, if $\tilde{A}_0|_{\mathcal{L}}$ is ρ-far from satisfying the LOW-DEGREE TEST, so is $\Pi_{\mathcal{L}}$ from satisfying the ROBUST LOW-DEGREE TEST. Thus, Π has expected distance at least 2ε from satisfying the ROBUST LOW-DEGREE TEST. Now, the oracle positions read by the ROBUST LOW-DEGREE TEST constitute a constant fraction of the oracle positions read by the ROBUST-PCPP–VERIFIER,

so Π has expected distance $\Omega(\varepsilon)$ from satisfying the ROBUST-PCPP–VERIFIER. Thus, the robustness of the individual test instance is transfered to the combined ROBUST LOW-DEGREE TEST by bundling. The case with the other test types is similar. We thus have that $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi}(R)] \geq \Omega(\varepsilon)$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi}(R)] \geq \delta' - \delta/2$. The lemma then follows by setting $\rho = \Omega(\varepsilon)$.
■

9.4 *Robustness over the binary alphabet*

The transformation from a robust verifier over the alphabet Σ to one over the binary alphabet is analogous to converting non-Boolean error correcting codes to Boolean ones via "code catenation". This transformation is identical to the transformation mentioned in Section 5.4.3. As the proof mimics the proof indicated in Section 5.4.3, we merely state the final result.

Lemma 9.4.1 *There are constants $c \in \mathbb{Z}^+$ and $\rho > 0$ such the following holds for every $n, m \in \mathbb{Z}^+$, $\delta, \delta' > 0$ satisfying $m \leq \log n/\text{loglog}n$, $n^{1/m} \geq m^{cm}/\delta^3$, $\lambda \leq \min\{1/c\log n, \delta^3/m^{cm}\}$, $\delta' > \delta$. If W is δ'-far from satisfying the circuit, then then for any proof oracles $\Pi : F^m \to \{0,1\}^{\log|\Sigma|}$, $\Upsilon : F^m \to \{0,1\}^b$, either $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] \geq \rho$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \geq \delta' - \delta/2$.*

It is to be noted that the expected robustness of the proof oracle (ρ) is not the same as similar parameters in Lemmas 9.3.2, but weaker by a constant factor as suggested in Lemma 2.4.5.

Finally, we conclude by proving Theorem 9.1.1.

Proof (of Theorem 9.1.1): Theorem 9.1.1 is proved using the ROBUST-PCPP–VERIFIER defined in this section setting $\lambda = \min\{1/c\log n, \delta^3/m^{cm}\}$. The randomness, query and decision complexity of the ROBUST-PCPP–VERIFIER (i.e., before the transformation to the binary alphabet) are as mentioned in Proposition 9.3.1. As mentioned in the earlier paragraph, the transformation from the alphabet Σ to the binary alphabet maintains the randomness complexity while the query (and decision) complexity increase by at most a constant factor. Hence, the randomness, query and decision complexities of the verifier are as claimed in Theorem 9.1.1

So far, we have considered the proof and input oracle separately. Hence the expected robustness in Lemma 9.4.1 was expressed separately for the proof and input oracles. We can consider them together by giving weights to the two oracle portions in the decision circuits (i.e. repeating queries, see Remark 8.2.1).

We give weight $(1-\gamma)$ to the input oracle and γ to the proof oracle, where γ is as specified in Theorem 9.1.1. Recall that these weights mean that each query to the input oracle is repeated several times such that the relative length of the input-part in the decision circuit is $1-\gamma$. These repeated queries may increase the query (and decision) complexity increase by a factor of at most

$1/\gamma$. Note that weighting does not afect the randomness complexity (or any other parameter such as the proximity parameter δ).

Since $n^{1/m} \geq m^{cm}/\delta^3$, we have $n^{1/m} \geq 8000(c_F m^2)^{m-1}/\delta^3$ or equivalently $n \geq 8000|F|^{m-1}/\delta^3$. Hence, Lemma 9.4.1 applies and we have that either $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] \geq \rho$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \geq \delta' - \delta/2$. Note that the first expression refers to the "expected robustness" of the proof-part whereas the second expression refers to the input-part. The overall expected robustness is obtained by a weighted average of these two expressions, where the weights are with respect to the aforementioned weighting (which assigns weight γ to the input-part). Hence, the expected robustness with respect to the said weighting is

$$\gamma \cdot \mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] + (1-\gamma)\cdot \mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \;\geq\; \min\{\gamma\cdot\rho, (1-\gamma)\cdot(\delta' - \delta/2)\}\,.$$

Thus, the expected robustness is as claimed. Noting that the other parameters (e.g., the randomness and decision complexities) are as claimed, Theorem 9.1.1 follows. ■

9.5 Linearity of encoding

In this section we point out that, for linear circuits (to be defined below), the mapping from an assignment to the corresponding PCP of proximity is linear. Throughout this section, "linear" means $GF(2)$-linear (yet, we will sometimes refer to F-linearity, for an extension field F of $GF(2)$). The main motivation to the current study is to derive linear codes satisfying local-testability and relaxed local-decodability (i.e., Theorems 1.3.4 and 1.3.5, respectively). Specifically, the constructions presented in Section III yield linear codes provided that the corresponding PCP of proximity is linear in the aforementioned sense.

We call a circuit is *linear* if it is a conjunction of linear constraints. However, instead of representing this conjunction via AND gates, it is more convenient for us to work with circuits that have multiple output gates, one for each linear constraint. That is:

Definition 9.5.1 *A multi-output circuit is* linear *if all its* internal *gates are parity gates and an input is accepted by it if and only if all output gates evaluate to zero.*

Proposition 9.5.2 *If C is a linear circuit, then there is a linear transformation T mapping satisfying assignments w of C to proof oracles $T(w)$ such that the PCPP verifier of Theorem 6.1.1 will, on input C, accept oracle $(w, T(w))$ with probability 1. Moreover, all the decision circuits produced by the verifier, on input C, can be made linear (while maintaining the claimed decision complexity).*

In the rest of this section, we provide a proof of Proposition 9.5.2, starting with an assignment w that satisfies the linear circuit. We prove that the mapping from w to a proof-oracle is linear by reviewing our construction of this mapping and ensuring that all steps in this construction are linear transformations.

Phase I* - STRUCTURED-CKTSAT*: In this phase (described in Section 7.2) we write down the values to all gates of the circuit and route them along the wrapped de Bruijn graph. Actually, we make a few minor and straightforward modifications to Definition 7.2.2: we allow multiple output gates (rather than a single output gate) and require that each such gate evaluates to zero (rather than to 1).[2] Also, here we deal with gate types that are linear (e.g., XOR), rather than arbitrary (e.g., AND and OR).

Since all the circuit gates are linear functions of the input, the values on the wires leaving the zero-th layer of the well-structured circuit (i.e., the last two bits of the mapping $A_0 : \{0,1\}^N \to \{0,1\}^4$ in Section 7.2) are linear in the input (i.e., in w). As to A_i, $i > 0$, (and the first two bits of A_0) notice that it is obtained by permuting the values of the previous layer A_{i-1} and setting some wires to zero (if they are not needed in the routing, e.g. gates 3 and 4 in Figure 7-3). These operations are linear, and so all assignment functions are linear in the input.

Phase II - Arithmetization: In this phase (described in Section 7.3) we extend the values given by A_i to an evaluation of a low-degree multivariate polynomial over some finite field F that is an extension field of $GF(2)$ of degree f. Each value of A_i is four bits long (say b_0, b_1, b_2, b_3) and identified with the element $b_0e_0 + b_1e_1 + b_2e_2 + b_3e_3$, where $e_0, \dots, e_{f-1}$ is a basis for F viewed as a vector space over $GF(2)$. We view A_i as a function $A_i : H^m \to F$ and construct a low-degree extension $\tilde{A}_i : F^m \to F$ of A_i by interpolation. on all inputs in H^m and use these values to interpolate and evaluate $\tilde{A}_i$ on all points in F^m. Notice that interpolation is F-linear and hence also $GF(2)$-linear. We conclude that the values of $\tilde{A}_i$ on all points in F^m is a linear transformation of the values of A_i. Since A_i is linear in the input assignment, so is $\tilde{A}_i$.

Clarification: Many parts of our encoding (starting with $\tilde{A}_i$) consist of evaluations of multivariate polynomials $P(x)$ over F^m. The linearity we claim is not linearity in x (the free variables of the polynomial). Rather, we claim the table of values $\{P(a) : a \in F^m\}$ is linear in the initial assignment w, which may be viewed as the information encoded in this table. In contrast, throughout this section, x is merely an index to this table. For example, in Phase II we showed the table $\{\tilde{A}_i(a) : a \in F^m\}$ is obtained by a linear transformation applied to the table $\{A_i(a') : a' \in H^m\}$ (but we certainly do not claim $\tilde{A}_i(a)$ is linear in a). That is, each $\tilde{A}_i(a)$ is a linear combination of the $A_i(a')$'s.

Phase III - The Constraint Polynomials: We now discuss the polynomials $P_{i,0}^{(0)}$ and $P_{i,1}^{(1)}$ defined in Equation (7.2), and show their values are a linear transformation of the values of $\tilde{A}_i$. The first polynomial (i.e., $P_{i,0}^{(0)}$) is obtained by applying the univariate polynomial ψ_0 defined in Equation 7.1 to each value of $\tilde{A}_i$ (i.e., $P_{i,0}^{(0)}(x) = \psi_0(\tilde{A}_i(x))$). By definition, ψ_0 evaluates to zero iff its input, when

[2]Recall that an input is accepted by the linear circuit if and only if all output gates evaluate to zero.

represented as a vector in $GF(2)^f$, belongs to the linear space spanned by $\{e_0, e_1, e_2, e_3\}$. This polynomial defines a linear transformation, as claimed by the following lemma.

Lemma 9.5.3 *Let L be a* GF(2)*-linear subspace of $F = GF(2^f)$ and $\psi_L(t) = \prod_{\alpha \in L}(t - \alpha)$. Then the mapping $\psi_L : F \to F$ is linear.*

Proof: We use the fact that for any integer i, the transformation $t \mapsto t^{2^i}$ is linear; that is, $(t+t')^{2^i} = t^{2^i} + t'^{2^i}$. Our main claim is that the polynomial $\psi_L(t)$ can be written as $\sum_i c_i t^{2^i}$ and hence is linear (being a sum of linear transformations). We prove this claim by induction on the dimension of $L \subseteq GF(2)^f$. Indeed, for $dim(L) = 0$ (i.e., $L = \{0^f\}$), it holds that $\psi_L(t) = t$ and our claim follows. In the induction step, write L as $L = L' \cup \{\alpha + L'\}$ where L' is some linear space of dimension $k-1$ and $\alpha \in L \setminus L'$. Clearly, $\psi_L(t) = \psi_{L'}(t) \cdot \psi_{L'}(t+\alpha)$. Using the inductive hypothesis for L' (and the linearity of $t \mapsto t^{2^j}$), we get

$$\begin{aligned}
\psi_L(t) &= \left(\sum_i c_i \cdot t^{2^i}\right) \cdot \left(\sum_j c_j \cdot (t+\alpha)^{2^j}\right) \\
&= \left(\sum_i c_i \cdot t^{2^i}\right) \cdot \left(\sum_j c_j \cdot \left(t^{2^j} + \alpha^{2^j}\right)\right) \\
&= \sum_{i,j} c_i c_j t^{2^i} t^{2^j} + \sum_{i,j} c_i c_j t^{2^i} \alpha^{2^j} \\
&= \sum_i c_i^2 t^{2^{i+1}} + \sum_i c_i' t^{2^i}
\end{aligned}$$

where $c_i' = \sum_j c_i c_j \alpha^{2^j}$ and $\sum_{i \neq j} c_i c_j t^{2^i} t^{2^j} = 2 \sum_{i<j} c_i c_j t^{2^i} t^{2^j} = 0$ (because F has characteristic 2). This completes the proof of the inductive claim.

We now turn to the second polynomial, $P_{i,0}^{(1)}$. Recall that $P_{i,0}^{(1)}(x) = \psi_1(s, a, a_0, a_1)$, where $s = \tilde{T}_i(x)$, $a = \tilde{A}_i(x)$ and $a_j = \tilde{A}_{i+1}(\tilde{\Gamma}_{i,j}(x))$. It can be verified that $\tilde{T}_i(x)$ (which represents the gate type) is independent of the input w to the circuit, and by our previous discussion a, a_0, a_1 are linear in the input w (to the circuit). Thus, it will suffice to show that ψ' is linear in its last three inputs. When discussing Equation (7.2) we did not go into the specific construction of the polynomial ψ' because only its functionality mattered, and we showed that there exists some constant-degree polynomial that does the job. But for our current purposes (of showing linearity) we need to present some specific polynomial ψ' that is linear (as an operator over $GF(2)^f$) and has the desired properties needed by the verification process. To do this, recall $\mathcal{C}$ is the set of allowable gates in the well-structured circuit, and so we define $\delta_{s_0}(z)$ to be the (minimal degree) uni-variate polynomial of degree $|\mathcal{C}|$ that is 1 if $z = s_0$ and is 0 if $z \in \mathcal{C} \setminus \{s_0\}$, and write ψ' as

$$\psi'(s, a, a_0, a_1) = \sum_{s_0 \in \mathcal{C}} \delta_{s_0}(s) \cdot \psi'_{s_0}(a, a_0, a_1) \qquad (9.4)$$

Claim 9.5.4 *For any $s_0 \in C$ that can occur as a gate in a well-structured circuit constructed from a linear circuit C, the polynomial $\psi'_{s_0}(a, a_0, a_1)$ of Equation 9.4 can be written as a linear transformation (of (a, a_0, a_1)).*

Proof: Recall that the value of $\psi'_{s_0}(a, a_0, a_1)$ is supposed to represent whether or not the four least significant bits of the three inputs (denoted a', a'_0 and a'_1) satisfy some condition. By inspecting Definition 7.2.2, it can be verified that (in our case) this condition is a linear one. That is, $\psi'_{s_0}(a, a_0, a_1) = 0$ if and only if the triplet (a', a'_0, a'_1), viewed as a 12-bit vector over $GF(2)$, belongs to some specific linear space $L_{s_0} \subseteq GF(2)^{12}$.

Recall that we may assume that $a = 0^{f-4}a'$ (and similarly for a_0 and a_1), because this condition is imposed by the constraint polynomial $P^{(0)}_{i,0}$. Thus, we seek a polynomial (over F^3) such that if each of its three inputs belongs to $Span(e_0, \ldots, e_3)$ then it will output 0 iff the inputs reside in the linear space that is analogous to L_{s_0}; that is, the input (a, a_0, a_1) should evaluate to 0 iff $a' \circ a'_0 \circ a'_1 \in L_{s_0}$. To obtain this, we assume the existence of $\alpha \in F$ such that multiplying an element by α corresponds to a left cyclic shift by four positions (e.g., $\alpha \cdot \sigma_0 \cdots \sigma_{f-1} = \sigma_4 \cdots \sigma_{f-1}\sigma_0 \cdots \sigma_3$). Such an element exists for the standard representation of F. Using this element we can write $\psi'_{s_0} : F^3 \to F$ as

$$\psi'_{s_0}(a, a_0, a_1) = \psi_{L_{s_0}}(\alpha^2 a + \alpha a_0 + a_1)$$

where $\psi_{L_{s_0}}$ is the univariate polynomial that is zero iff its input is in L_{s_0}. Note that, for inputs in $Span(e_0, \ldots, e_3)$, indeed $\psi'_{s_0}(a, a_0, a_1) = 0$ iff $a' \circ a'_0 \circ a'_1 \in L_{s_0}$. By Lemma 9.5.3, $\psi_{L_{s_0}}$ is linear. It follows that ψ'_{s_0} is linear, because multiplication by a fixed element of F (i.e., α) is a linear operation. ■

Recall $\delta_{s_0}(s)$ depends only on the circuit and not on its input (i.e., w). Thus, each summand of (9.4) is linear in w and hence the sum is itself linear in w. We conclude that the table of evaluations of the polynomials given by Equation (7.2) is obtained by linear transformations applied to the input to the circuit.

Phase IV - The Sum-check Polynomials: In this phase (described by Equation (7.3)) we apply a sequence of interpolations to previously constructed polynomials $P^{(b)}_{i,j}$. Each such interpolation is an F-linear transformation and hence also a GF(2)-linear one. Thus, the sequence of polynomials $P^{(b)}_{i,j}$ is obtained by a linear transformation applied to the input.

Phase V - Bundling and Encoding: In this phase (described in Sections 9.3 and 9.4) we apply some cyclic shifts to the (values of the) sequence of $l + 2l(m+1)$ polynomials obtained in the previous phases. Then we bundle the polynomials together, obtaining an alphabet of size $|F|^{l+2l(m+1)}$. This bundling does not change the encoding (only the partitioning of the proof into symbols) and hence is also a linear transformation. Finally, we apply an error correcting code to each symbol in order to reduce the alphabet size (from $|F|^{l+2l(m+1)}$) to binary, and this is also a linear transformation as long as the error correcting code is itself linear.

The result of this shifting, bundling and encoding is the actual proof given to the (outer) verifier of Theorem 9.1.1. Notice this transformation from $l + 2l(m + 1)$ polynomials (each evaluated in F) to one proof (over the binary alphabet) is linear, because all three parts of it are linear.

Now we argue that all tests performed by the verifier are linear and the decision complexity claimed in Theorem 9.1.1 can be achieved by using small linear circuits. This can be seen by inspecting the various tests described in Section 7.4, noticing that they all check either linear or F-linear conditions, and applying the general result of Strassen [Str73] showing that any algebraic circuit that computes a linear function (as a formal polynomial) can be converted into a linear circuit with only a constant-factor increase in size. This completes the proof of Proposition 9.5.2.

CHAPTER 10

Putting them together: Very short PCPs with very few queries

10.1 Main Construct - Recalled

In this chapter we prove the main results of this work; that is, we establish Theorem 1.3.2 and 1.3.3. Our starting point is the following Robust PCP of proximity, which is constructed in the second part of this work.

Theorem 6.1.1 (Main Construct - restated): *There exists a universal constant c such for all $n, m \in \mathbb{Z}^+$, $0 < \delta, \gamma < 1/2$ satisfying $n^{1/m} \geq m^{cm}/(\gamma\delta)^3$ and $\delta \leq \gamma/c$,* CIRCUIT VALUE *has a robust PCP of proximity (for circuits of size n) with the following parameters*

- *randomness* $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$,
- *decision complexity* $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$, *which also upper-bounds the query complexity.*[1]
- *perfect completeness, and*
- *for proximity parameter δ, the verifier has robust-soundness error γ with robustness parameter $(1 - \gamma)\delta$.*

We comment that the condition $\delta < \gamma/c$ merely means that we present robust PCPs of proximity only for the more difficult cases (when δ is small), and our robustness parameter does not improve

[1]In fact, we will upper-bound the query complexity by $q = n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ and show that the verifier's decision can be implemented by a circuit of size $\widetilde{O}(q)$, which can also be bounded by $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ with a slightly larger unspecified polynomial.

for larger values of δ. We call the reader's attention to the typically small value of the query and randomness complexities, which yield a proof length that is upper-bounded by $\mathrm{poly}(m^m \log n) \cdot n$ (for δ and γ as small as $1/\mathrm{poly}(m^m, \log n)$), as well as to the small values of the soundness error and the the small deterioration of robustness wrt proximity.

We also need the following robust PCP of proximity which we constructed while proving the PCP Theorem in Chapter 5. This robust has parameters similar to the PCP constructed by Arora *et al.* [ALM$^+$98]. In comparison to the main construct above, this PCPP is not very efficient in randomness. However, as we plan to use this robust PCPP only towards the final stages of composition, we can afford to pay this cost in randomness.

Theorem 5.1.2 (ALMSS-type Robust PCP of proximity - restated): *For all $n \in \mathbb{Z}^+$ and $\delta \in (0,1)$,* CIRCUIT VALUE *has a robust PCP of proximity (for circuits of size n) with the following parameters*

- *randomness $O(\log n)$,*
- *decision complexity $\mathrm{poly}(\log n)$, which also upper-bounds the query complexity.*
- *perfect completeness, and*
- *for proximity parameter δ, the verifier has robust-soundness error $1 - \Omega(\delta)$ with robustness parameter $\Omega(1)$.*

10.2 Composing the Main Construct

Using Theorems 6.1.1 and 5.1.2, we derive the general trade-off (captured by the following Theorem 10.2.1) between the length of PCPs and their query complexity via repeated applications of Composition Theorem (Theorem 3.2.1).

Theorem 10.2.1 (Randomness vs. query complexity trade-off for PCPs of proximity) *For every parameters $n, t \in \mathbb{N}$ such that $3 \leq t \leq \frac{2 \log\log n}{\log\log\log n}$ there exists a PCP of proximity for* CIRCUIT VALUE *(for circuits of size n) with the following parameters*

- *randomness complexity $\log_2 n + A_t(n)$, where*
$$A_t(n) \triangleq O\big(t + (\log n)^{\frac{1}{t}}\big) \log\log n + O((\log n)^{\frac{2}{t}}) \tag{10.1}$$
- *query complexity $O(1)$,*
- *perfect completeness, and*
- *soundness error $1 - \Omega(1/t)$ with respect to proximity parameter $\Omega(1/t)$.*

Alternatively, we can have query complexity $O(t)$ and soundness error $1/2$ maintaining all other parameters the same.

For $t \in [3, ..., \frac{0.99 \log\log n}{\log\log\log n}]$, we have $(\log n)^{\frac{1}{t}} > (\log\log n)^{1/0.99}$ and so $A_t(n) = O((\log n)^{\frac{2}{t}})$. On the other hand, for $t \geq \frac{1.01 \log\log n}{\log\log\log n}$, we have $(\log n)^{\frac{1}{t}} \leq (\log\log n)^{1/1.01}$ and so $A_t(n) = O\left((\log\log n)^2 / \log\log\log n \right) = o(\log\log n)^2$.

Theorem 10.2.1 actually asserts a PCP of proximity (for CIRCUIT VALUE), but a PCP for CIRCUIT SATISFIABILITY and a PCP of proximity for NONDETERMINISTIC CIRCUIT VALUE (of the same complexity) follow; see Propositions 2.2.2 and 2.2.3. Theorems 1.3.2 and 1.3.3 follow by suitable settings of the parameter t. Further detail as well as another corollary appear in Section 10.2.2.

10.2.1 Proof of Theorem 10.2.1

Theorem 10.2.1 is proved by using the robust PCP of proximity described in Theorem 6.1.1. Specifically, this robust PCP of proximity is composed with itself several times (using the Composition Theorem from Section 2). Each such composition drastically reduces the query complexity of the resulting PCP, while only increasing very moderately its randomness complexity. The deterioration of the soundness error and the robustness is also very moderate. After composing the robust PCP of proximity with itself $O(t(n))$ times, we compose the resulting robust PCP with the ALMSS-type robust PCP of proximity thrice to reduce the query complexity to $\text{poly}\log\log\log n$. Finally we compose this resultant robust PCP of proximity with a PCPP of proximity parameter roughly $\Omega(1/t)$ that has query complexity $O(1)$ and exponential length. The latter PCP of proximity can be obtained by a suitable modification of the Hadamard-based PCP of [ALM+98], as shown in Chapter 4.

Proof: We construct the PCP of proximity of Theorem 10.2.1 by composing the robust PCP of proximity described in Theorem 6.1.1 with itself several times. Each such composition reduces the query complexity from n to approximately $n^{1/m}$. Ideally, we would like to do the following: Set $m = (\log n)^{\frac{1}{t}}$ and compose the robust PCPP of Theorem 6.1.1 with parameter m with itself $t-1$ times. This would result in a robust PCPP of query complexity roughly $n^{1/m^t} = n^{1/\log n} = O(1)$ giving us the desired result. However, we cannot continue this repeated composition for all the $t-1$ steps as the requirements of Theorem 6.1.1 (namely, $n^{1/m} \geq m^{cm}/(\delta\gamma)^3$) are violated in the penultimate two steps of the repeated composition. So we instead do the following: In the first stage, we compose the (new and) highly efficient verifier from Theorem 6.1.1 with itself $t-3$ times. This yields a verifier with query complexity roughly $(n^{1/m^t})^{m^2} = 2^{m^2} = \exp(\log^{2/t} n) < n$, while the soundness error is bounded away from 1 and robustness $\Omega(1/t)$. In the second stage, we compose the resultant robust PCPP a constant number of times with the ALMSS-type robust PCPP described in Theorem 5.1.2 to reduce the query complexity to $\text{poly}\log\log\log n$ (and keeping the other parameters essentially the same). The ALMSS-type PCPP is (relatively) poor in terms of randomness, however the input size to the ALMSS-type PCPP is too small to affect the randomness of the resultant PCPP. Finally, we compose with the Hadamard-based verifier of Theorem 4.1.1 to bring the query complexity down to $O(1)$. In all stages, we invoke the Composition Theorem (Theorem 3.2.1).

Throughout the proof, n denotes the size of the circuit that is given as explicit input to the PCPP verifier that we construct. We shall actually construct a sequence of such verifiers. Each verifier in the sequence will be obtained by composing the prior verifier (used as the outer verifier in the composition) with an adequate inner verifier. In the first stage, the inner verifier will be the verifier obtained from Theorem 6.1.1, whereas in the second and third stages it will be the one obtained from Theorem 5.1.2 and Theorem 4.1.1, respectively. Either way, the inner verifier will operate on circuits of much smaller size (than n) and will use a proximity parameter that is upper-bounded by the robustness parameter of the corresponding outer verifier.

Stage I: Let $m = (\log n)^{\frac{1}{t}} \geq 2$ and $\gamma = \frac{1}{t}$. For this choice of m and γ, let V_0 be the verifier obtained from Theorem 6.1.1. We recall the parameters of this verifier: For circuits of size ℓ and any proximity parameter $\delta_0 \in (\gamma/3c, \gamma/c)$, its randomness complexity is $r_0(\ell) \triangleq (1 - \frac{1}{m}) \cdot \log_2 \ell + O(\log\log \ell) + O(m \log m) + O(\log t)$, its decision (and query) complexity is $d_0(\ell) \triangleq \ell^{\frac{1}{m}} \cdot \mathrm{poly}(\log \ell, t)$, its soundness error is $s_0 \triangleq \gamma$ and its robustness is $\rho_0 \geq (1-\gamma)\delta_0$.

We compose V_0 with itself $t-3$ times for the same fixed choice of m and γ to obtain a sequence of verifiers of increasingly smaller query complexity. While doing so, we will use the largest possible proximity parameter for the inner verifier (V_0) in each stage; that is, in the ith composition, we set the proximity parameter of the inner verifier to equal the robustness of the outer verifier, where the latter is the result of $i-1$ compositions of V_0 with itself. We get a sequence of verifiers $V_1, \ldots, V_{t-2}$ such that $V_1 = V_0$ and the verifier V_i is obtained by composing (the outer verifier) V_{i-1} with (the inner verifier) V_0, where the proximity parameter of the latter is set to equal the robustness of the former. Unlike V_0, which is invoked on different circuit sizes and (slightly) different values of the proximity parameter, all the V_i's ($i \in [t-2]$) refer to circuit size n and proximity parameter $\delta \triangleq \gamma/c < 1/t$.

Let r_i, d_i, δ_i, s_i and ρ_i denote the randomness complexity, decision (and query) complexity, proximity parameter, soundness error, and the robustness parameter of the verifier V_i. (Recall that V_i will be composed with the inner-verifier V_0, where in this composition the input size and proximity parameter of the latter will be set to d_i and ρ_i respectively, and so we will need to verify that $d_i^{1/m} \geq m^{cm}/(\gamma\rho_i)^3$ and $\rho_i < \gamma/c$ for $i < t-2$).[2] We first claim that the decision complexity, proximity, soundness-error, robustness, and proof size parameters satisfy the following conditions:

1. Decision complexity: $d_i(n) \leq a(n,m)^2 \cdot n^{1/m^i}$, where $a(\ell, m) \triangleq d_0(\ell)/\ell^{1/m} = \mathrm{poly}(\log \ell, t)$. On

[2]We also need to verify that $n^{1/m} \geq m^{cm}/(\gamma\delta_0)^3$ and $\delta_0 < \gamma/c$ for the initial verifier $V_1 = V_0$ but this is true for our choice of parameters. Furthermore, as ρ_i can only deteriorate with each composition, we have that $\rho_i \leq \rho_0 \leq \gamma/c$. Thus, the only condition that needs to be verified is $d_i^{1/m} \geq m^{cm}/(\gamma\rho_i)^3$ for $i < t-2$.

the other hand, $d_i(n) \geq n^{1/m^i}$.

2. Proximity: $\delta_i = \delta$.

3. Soundness error: $s_i \leq 1 - (1-\gamma)^i$ (In particular, $s_i < i\gamma$).

4. Robustness: $\rho_i \geq (1-\gamma)^i \cdot \delta$. On the other hand, $\rho_i \leq \rho_0 < \gamma/c$.

5. Proof length: $2^{r_i(n)} d_i(n) \leq b(n,m)^i \cdot n$, where $b(\ell, m) \triangleq 2^{r_0(\ell)} \cdot d_0(\ell)/\ell = \text{poly}(m^m, \log \ell, t)$.

We prove this claim by induction on i. For starters, note that the base case (i.e., $i = 1$) follows from the properties of V_0: In particular, $d_1(n) \leq \text{poly}(\log n, t) \cdot n^{1/m}$ and $2^{r_1(n)} d_1(n) \leq \text{poly}(m^m, \log n, t) \cdot n$. Turning to the induction step, assuming that these claims holds for V_i, we prove that they hold also for V_{i+1}. For (1), note that

$$\begin{array}{rcll} d_{i+1}(n) & = & d_0(d_i(n)) & \text{[By the Composition Theorem]} \\ & = & a(d_i(n), m) \cdot d_i(n)^{1/m} & \text{[By the properties of } V_0\text{]} \\ & \leq & a(n,m) \cdot d_i(n)^{1/m} & \text{[By monotonicity of } a(\cdot,\cdot) \text{ and } d_i(n) \leq n\text{]} \\ & \leq & a(n,m) \cdot \left(a(n,m)^2 \cdot n^{1/m^i}\right)^{1/m} & \text{[By induction]} \\ & \leq & a(n,m)^2 \cdot n^{1/m^{i+1}} & \text{[Using } m \geq 2\text{]} \end{array}$$

and $d_{i+1}(n) \geq d_i(n)^{1/m} \geq n^{1/m^{i+1}}$ also holds. Clearly $\delta_i = \delta$ and the bound on s_i is straightforward from the Composition Theorem. Recalling that the proximity parameter for V_0 in this composition is set to ρ_i, the robustness of the composed verifier V_{i+1} is $\rho_{i+1} = (1-\gamma)\rho_i = (1-\gamma)^{i+1}\delta$ as desired. Furthermore, $\rho_i = (1-\gamma)^i \delta \geq (1 - \frac{1}{t})^t \delta \geq e^{-1}\delta = \gamma/O(1)$. We now move to the last condition (essentially bounding the randomness). Notice first that $r_{i+1}(n) = r_i(n) + r_0(d_i(n))$ and thus

$$\begin{array}{rcll} 2^{r_{i+1}(n)} \cdot d_{i+1}(n) & = & 2^{r_i(n)} \cdot 2^{r_0(d_i(n))} \cdot d_0(d_i(n)) & \text{[By the Composition Theorem]} \\ & \leq & 2^{r_i(n)} \cdot d_i(n) \cdot b(d_i(n), m) & \text{[By the properties of } V_0\text{]} \\ & \leq & n \cdot b(n,m)^i \cdot b(n,m) & \text{[By induction and monotonicity of } b(\cdot,\cdot)\text{]} \\ & \leq & n \cdot b(n,m)^{i+1} & \end{array}$$

Thus, Part (5) is verified. Recall that we have to verify that $d_i^{1/m} \geq m^{cm}/(\gamma\rho_i)^3$ for $i < t-2$ as promised before. We have $d_i^{1/m} \geq (n^{1/m^i})^{1/m} = n^{1/m^{i+1}} \geq n^{1/m^{t-2}}$ (since $i < t-2$). Since $m = (\log n)^{\frac{1}{t}}$, we have $n^{1/m^t} = 2$. Hence, $d_i^{1/m} \geq (n^{1/m^t})^{m^2} = 2^{m^2}$. On the other hand, $m^{cm}/(\gamma\rho_i)^3 \leq m^{cm}/(e^{-1}\gamma\delta)^3 = m^{cm} \cdot \text{poly}(t)$. Thus it suffices to verify that $2^{m^2}/m^{cm} \geq \text{poly}(t)$, for $3 \leq t \leq 2 \log\log n / \log\log\log n$, which is straightforward.[3]

[3]Note that as t varies from 3 to $2\log\log n/\log\log\log n$, the value of m varies from $\sqrt{\log n}$ to $\sqrt{\log\log n}$. For $t \in [3, 2\log\log n/\log\log\log n]$, the maximum value of $\text{poly}(t)$ is $\text{poly}(\log\log n/\log\log\log n) = \text{poly}(\log\log n)$. On the other hand, for $m \in [\sqrt{\log\log n}, \sqrt{\log n}]$, the minimum value of $2^{m^2}/m^{cm} > 2^{m^2/2}$ is $2^{\sqrt{\log\log n}^2/2} = \sqrt{\log n} > \text{poly}(\log\log n)$.

Lastly, we consider the running-time of V_i, denoted T_i. A careful use of the Composition Theorem (Theorem 3.2.1) indicates that $T_i(n) = \mathrm{poly}(n) + T_{i-1}(n)$, for every $i = 2, \ldots, t-2$, where $T_1(n) = \mathrm{poly}(n)$ (since $V_1 = V_0$). Alternatively, unraveling the inductive composition, we note that V_i consists of invoking V_0 for i times, where in the first invocation V_0 is invoked on V_i's input and in later invocations V_0 is invoked on an input obtained from the previous invocation. Furthermore, the output of V_i is obtained by a combining the inputs obtained in these $i \leq t-2 < n$ invocations.

We now conclude the first stage by showing that the final verifier $V_c = V_{t-2}$ has the desired properties. By Part (5) above (and the fact that $d_{t-2} \geq 1$), we have $r_c(n) = r_{t-2}(n) \leq \log n + (t-2) \cdot \log b(n,m) \leq \log n + t \log b(n,m)$. By the definition of $b(n,m)$, we have $\log b(n,m) = O(\log\log n) + O(m \log m) + O(\log t) = O(\log\log n + m \log m)$, whereas $m \log m = (\log n)^{\frac{1}{t}} \cdot \frac{1}{t} \log\log n$. Thus $r_c(n) \leq \log_2 n + O(t \cdot \log\log n) + t \cdot O(m \log m) = \log_2 n + O(t + (\log n)^{\frac{1}{t}}) \cdot \log\log n$. The decision complexity of V_c is $d_c(n) = d_{t-2}(n) \leq a(n,m)^2 \cdot n^{1/m^{t-2}} = a(n,m)^2 \cdot 2^{m^2}$, because $n^{1/m^t} = 2$. Using $a(n,m) = \mathrm{poly}(\log n, t)$, it follows that $d_c(n) \leq 2^{m^2} \cdot \mathrm{poly}(\log n)$. The proximity of V_c equals δ, its soundness error is $s_c = s_{t-2} = 1 - (1-\gamma)^{t-2} = 1 - (1 - (1/t))^{t-2} < 1/2$, and its robustness is $\rho_c = \rho_{t-2} \geq (1-\gamma)^{t-2}\delta = \delta/e = \Omega(1/t)$.

Stage II: We now compose the verifier V_c with the ALMSS-type verifier V_a described in Theorem 5.1.2 thrice to obtain the verifiers V', V'', and V''' respectively (i.e., V' equals V_c composed with V_a, V'' equals V' composed with V_a, and V''' equals V'' composed with V_a). We compose as before setting the proximity parameter of the inner verifer equal to the robustness parameter of the outer verifier. Recall from Theorem 5.1.2 that the ALMSS-type verifer V_a has the following parameters: randomness $r_a(\ell, \delta) = O(\log \ell)$, decision complexity $d_a(\ell, \delta) = \mathrm{poly}\log \ell$, soundness error $s_a(\ell, \delta) = 1 - \Omega(\delta)$ and robustness $\rho_a(\ell, \delta) = \Omega(1)$ for input size ℓ and proximity parameter δ. Thus each composition with the inner verifier V_a adds $O(\log q)$ to the randomness while reducing the query complexity to $\mathrm{poly}\log q$ where q is the decision complexity of the outer verifier. Furthermore, the robustness parameter improves to the constant $\Omega(1)$ while the soundness error increases from a constant to $1 - \Omega(\rho)$ where ρ is the robustness of the outer verifier (provided the soundness error of the outer verifier is a constant). Hence, the parameters of the verifiers V', V'' and V''' are as follows:

Parameters of V' (recall that $d_c = 2^{m^2} \cdot \mathrm{poly}(\log n)$ and $\delta = \Omega(1/t)$):

$r' = r_c + O(m^2 + \log\log n)$	$d' = \mathrm{poly}(m, \log\log n)$
$s' = 1 - \Omega(\delta)$	$\rho' = \Omega(1)$

Parameters of V'':

$r'' = r' + O(\log m + \log\log\log n)$	$d'' = \mathrm{poly}(\log m, \log\log\log n)$
$s'' = 1 - \Omega(\delta)$	$\rho'' = \Omega(1)$

Parameters of V''':

$r''' = r'' + O(\log\log m + \log\log\log\log n)$	$d''' = \mathrm{poly}(\log\log m, \log\log\log\log n)$

$s''' = 1 - \Omega(\delta)$ $\qquad\qquad\qquad\qquad$ $\rho''' = \Omega(1)$

while the proximity parameter for all three verifiers is that of V_c (i.e., δ). We have that

$$\begin{aligned} r''' &= \log_2 n + O(t + (\log n)^{1/t}) \cdot \log\log n + O(m^2), \\ q''' < d''' &= \mathrm{poly}(\log\log\log\log n, \log\log m), \end{aligned}$$

whereas $\delta''' = \delta = 1/(ct)$, $s''' = 1 - \Omega(\delta)$ and $\rho''' = \Omega(1)$. Substituting $m = (\log n)^{\frac{1}{t}}$, we get $r''' = \log_2 n + O(t + (\log n)^{1/t}) \cdot \log\log n + O((\log n)^{\frac{2}{t}})$ and $q''' = \mathrm{poly}(\log\log\log n)$.

Stage III: Finally, we compose V''' with the Hadamard-based inner verifier V_h of Theorem 4.1.1 to obtain our final verifier V_f. The query complexity of V_h and hence that of V_f is constant. The randomness complexity of V_f is $r_f(n) \triangleq r'''(n) + r_h(q'''(n)) = r'''(n) + \mathrm{poly}(\log\log\log n)$, because $r_h(\ell) = O(\ell^2)$. Thus, $r_f(n) = \log_2 n + O(t + (\log n)^{\frac{1}{t}}) \cdot \log\log n + O((\log n)^{\frac{2}{t}})$. On proximity parameter δ_h, the soundness error of V_h is $s_h = 1 - \Omega(\delta_h)$. Setting $\delta_h = \rho''' = \Omega(1)$, we conclude that the soundness error of V_f on proximity parameter δ is $1 - \Omega(\delta) = 1 - \Omega(1/t)$ (since the soundness error of V''' is $1 - \Omega(\delta)$).

To obtain soundness error $1/2$, we repeat perform $O(t)$ repetitions of V_h, yielding a query complexity of $O(t)$. This can be done without increasing the randomness complexity by using "recycled randomness" (specifically, the neighbors of a uniformly selected vertex in a Ramanujan expander graph; see [Gol97, Apdx. C.4]). ∎

10.2.2 Corollaries to Theorem 10.2.1

Recall that Theorem 10.2.1 asserts a PCP of proximity with randomness complexity $\log_2 n + A_t(n)$, where $A_t(n) \triangleq O(t + (\log n)^{\frac{1}{t}}) \log\log n + O((\log n)^{\frac{2}{t}})$ and query complexity $O(t)$ (for soundness error $1/2$). For constant $t \geq 3$, we have $A_t(n) = O((\log n)^{\frac{2}{t}})$. On the other hand, for $t \geq \frac{1.01 \log\log n}{\log\log\log n}$, we have $A_t(n) = o(\log\log n)^2$.

Deriving Theorems 1.3.2 and 1.3.3: Two extreme choices of $t(n)$ are when $t(n) = \frac{2}{\varepsilon}$, for some $\varepsilon > 0$ (which maintains a constant query complexity), and $t(n) = \frac{2\log\log n}{\log\log\log n}$ (which minimizes the randomness complexity of the verifier). Setting $t(n) = \frac{2}{\varepsilon}$ yields Theorem 1.3.3 (i.e., constant query complexity $O(1/\varepsilon)$ and randomness $\log_2 n + O(\log^{\varepsilon} n)$), whereas setting $t(n) = \frac{2\log\log n}{\log\log\log n}$ yields Theorem 1.3.2 (i.e., query complexity $O((\log\log n)/\log\log\log n)$ and randomness $\log_2 n + O\big((\log\log n)^2/\log\log\log n\big)$). Thus, both Theorems 1.3.2 and 1.3.3 follow from Theorem 10.2.1.

Deriving a PCP of proximity for NONDETERMINISTIC CIRCUIT VALUE***:*** By Proposition 2.2.3, we conclude that for every $3 \leq t(n) \leq \frac{2\log\log n}{\log\log\log n}$, there exists a PCP of proximity for NONDETERMINISTIC CIRCUIT VALUE of the same complexities (i.e., randomness complexity $\log_2 n + A_t(n)$, query complexity $O(t(n))$, perfect completeness, and soundness error $1/2$ with respect to proximity $\delta = \Omega(1/t(n))$.

Comment: We note that the tight bound on the robustness (as a function of the proximity parameter) in our main construct (Theorem 6.1.1) plays an important role in the proof of Theorem 10.2.1. The reason is that when we compose two robust PCPs of proximity, the proximity parameter of the second must be upper-bounded by the robustness parameter of the first. Thus, when we compose many robust PCPs of proximity, the robustness parameter deteriorates exponentially in the number of composed systems where the base of the exponent is determined by the tightness of the robustness (of the second verifier). That is, let $\tau \triangleq \rho/\delta$, where δ and ρ are the proximity and robustness parameters of the system. Then composing this system t times with itself, means that at the lowest PCP-instance we need to set the proximity parameter to be τ^{t-1} times the initial proximity. This requires the lowest PCP-instance to make at least $1/\tau^{t-1}$ queries (or be composed with a PCP of proximity that can handle proximity parameter τ^t, which again lower-bounds the number of queries). For a constant $\tau < 1$, we get $\exp(t)$ query complexity, whereas for $\tau = 1 - \gamma = (1 - (1/t))$ we get query complexity that is linear in $1/((1-\gamma)^t \cdot \gamma) = O(t)$. Finally, we argue that in the context of such an application, setting $\gamma = 1/t$ is actually the "natural" choice. Such a choice, assigns each proof-oracle encountered in the composition almost equal weight (of $1/t$); that is, such a proof oracle is assigned weight $1/t$ when it appears as the current proof-oracle and maintains its weight when it appears as part of the input-oracle in subsequent compositions.

A more flexible notion of a PCP of proximity: Our definition of a PCP of proximity (see Definition 2.2.1) specifies for each system a unique proximity parameter. In many settings (see, e.g., Section 12), it is better to have the proximity parameter be given as an input to the verifier and have the latter behave accordingly (e.g., make an adeqaute number of queries). We refrain from presenting a formal definition as well as a general transformation of PCPs of proximity to their more relaxed form. Instead, we state the following corollary to Theorem 10.2.1.

Corollary 10.2.2 *For every parameters $n, t, T \in \mathbb{N}$ such that $3 \le t \le T \le \frac{2 \log\log n}{\log\log\log n}$ there exists a PCP of proximity for* Circuit Value *(for circuits of size n) with proof length $2^{A_t(n)} \cdot n$, where $A_t(n)$ is as in* Eq. (10.1)*, query complexity $O(T)$, perfect completeness, and soundness error $1/2$ with respect to proximity parameter $1/T$. Furthermore, when given* (as auxiliary input) *a proximity parameter $\delta \in (T^{-1}, t^{-1})$, the verifier makes only $O(1/\delta)$ queries and rejects any input oracle that is δ-far from satisfying the circuit with probability at least $1/2$.*

Underlying the following proof is a general transformation of PCPs of proximity to the more relaxed form as stated in Corollary 10.2.2.

Proof: The proof oracle consists of a sequence of proofs for the system of Theorem 10.2.1, when invoked with proximity parameter 2^{-i}, for $i = \lfloor \log_2 t \rfloor, ..., \lceil \log_2 T \rceil$. When the new verifier is invoked with proximity parameter δ, it invokes the original verifier with proximity parameter 2^{-i}, where $i = \lceil \log_2 1/\delta \rceil$, and emulates the answers using the i-th portion of its proof oracle. ∎

Part III

Coding Theory Applications

CHAPTER 11

Introduction

In this part, we describe some of the applications of our PCP constructions to coding theory. The flexibility of PCPs of proximity makes them relatively easy to use in obtaining results regarding locally testable and decodable codes. In particular, using a suitable PCP of proximity, we obtain an improvement in the rate of locally testable codes (improving over the results of [GS02, BSVW03]). We also introduce a relaxed notion of locally decodable codes, and show how to construct such codes using any PCP of proximity (and ours in particular). Before defining either locally testable codes or locally decodable codes, we first give a brief introduction to coding theory and sublinear time algorithms. We then show how locally testable codes and locally decodable codes appear naturally in the context of sublinear time algorithms for coding theory. We defer the formal definitions of these codes and their constructions to the next two chapters (Chapters 12 and 13).

11.1 Coding Theory

The fundamental problem in coding theory is to design "good" codes that can be "efficiently" decoded even if a faulty communication channel corrupts a small fraction of bits in the code. Before formalizing this question, we need the following coding theory preliminaries.

Preliminaries: For a string $w \in \{0,1\}^n$ and $i \in [n] \triangleq \{1, 2, ..., n\}$, unless stated differently, w_i denotes the i-th bit of w.

[0]This introductory chapter is based on the following talk by Madhu Sudan:
"Sublinear Time Algorithms in Coding Theory" (Invited Talk), *RANDOM '04* (Cambridge, Massachusetts, 22–24 Aug. 2004)
I am thankful to Madhu Sudan for letting me include some of the material from his talk in this chapter.

We consider codes mapping sequences of k (input) bits into sequences of $n \geq k$ (output) bits. Such a generic code is denoted by $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, and the elements of $\{\mathrm{C}(x) : x \in \{0,1\}^k\} \subseteq \{0,1\}^n$ are called codewords (of C). Throughout this part, *the integers k and n are to be thought of as parameters*, and we are typically interested in the relation of n to k (i.e., how n grows as a function of k). Thus, we actually discuss infinite families of codes (which are associated with infinite sets of possible k's), and whenever we say that some quantity of the code is a constant we mean that this quantity is constant for the entire family (of codes). We denote the ratio k/n as the rate of the code C.

The distance of a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ is the minimum (Hamming) distance between its codewords; that is, $\min_{x \neq y}\{\overline{\Delta}(\mathrm{C}(x), \mathrm{C}(y))\}$, where $\overline{\Delta}(u,v)$ denotes the number of bit-locations on which u and v differ. *Throughout this work, we focus on codes of "linear distance"*; that is, codes $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ of distance $\Omega(n)$. The distance of $w \in \{0,1\}^n$ from a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, denoted $\overline{\Delta}_{\mathrm{C}}(w)$, is the minimum distance between w and the codewords; that is, $\overline{\Delta}_{\mathrm{C}}(w) \triangleq \min_x\{\overline{\Delta}(w, \mathrm{C}(x))\}$. For $\delta \in [0,1]$, the n-bit long strings u and v are said to be δ-far (resp., δ-close) if $\overline{\Delta}(u,v) > \delta \cdot n$ (resp., $\overline{\Delta}(u,v) \leq \delta \cdot n$). Similarly, w is δ-far from C (resp., δ-close to C) if $\overline{\Delta}_{\mathrm{C}}(w) > \delta \cdot n$ (resp., $\overline{\Delta}_{\mathrm{C}}(w) \leq \delta \cdot n$).

Equipped with these preliminaries, we observe that if a communication channel is slightly faulty in the sense that it corrupts at most ε-fraction of the bits, then in order to be uniquely decodable any code C must have relative distance at least $\delta(C) \geq 2\varepsilon$. The fundamental question mentioned above can now be formalized as follows: For any $\varepsilon \in (0,1)$, design a code with relative distance at least 2ε that has maximum rate. In addition to this problem, we will also be interested in the algorithmic complexities of the following problems.

Let $C : \{0,1\}^k \to \{0,1\}^n$ be a fixed code.

Encoding: Encoding is the problem of finding the encoding of a given message. Formally, given any message $m \in \{0,1\}^k$, encoding is the problem of computing $C(m) \in \{0,1\}^n$.

Error Detection (Testing): Testing is the problem of deciding whether a given string is a codeword. Formally, given any string $w \in \{0,1\}^k$, testing is the problem of deciding whether there exists a $m \in \{0,1\}^k$ such that $w = C(m)$. A related (and possibly more interesting problem) is the problem of ε-testing: Given a string $w \in \{0,1\}^n$, does there exist a string $m \in \{0,1\}^k$ such that $\delta(w, C(m)) \leq \varepsilon$.

Error Correction (Decoding): Decoding involves finding the closest codeword to a given word. Formally, given an error parameter ε, decoding is the following problem: Given any string $w \in \{0,1\}^n$ such that $\delta(w, C) \leq \varepsilon$, find $m \in \{0,1\}^k$ that minimizes $\delta(w, C(m))$.

11.2 Sublinear time algorithms

The main question in sublinear time algorithms is the following: Given a computable function $f : \{0,1\}^k \to \{0,1\}^n$, can it be computed in time significantly less than both k and n, i.e., can it be computed in $o(k,n)$ time? On a first glance, the answer seems to be no since we do not have sufficient time to even read the input or for that matter even write the output. However, we can expect to perform such sublinear computations if we make the following modifications to the way the input and output are represented.

- The input is represented implicitly by an oracle. Whenever the sublinear time algorithm wants to access the j^{th} bit of the input string x (for some $j \in [k]$), it queries the input x-oracle for the j^{th} bit and obtains x_j. This (implicit) representation lets the algorithm work with the input even when it cannot read the entire input. Thus, the running time is not bounded below by the length of the input but by the number of queries made by the algorithm to the input oracle. We call the maximum number of queries made by the algorithm, the query complexity of the algorithm. We will be interested in algorithms that have very low query complexity, typically constant.

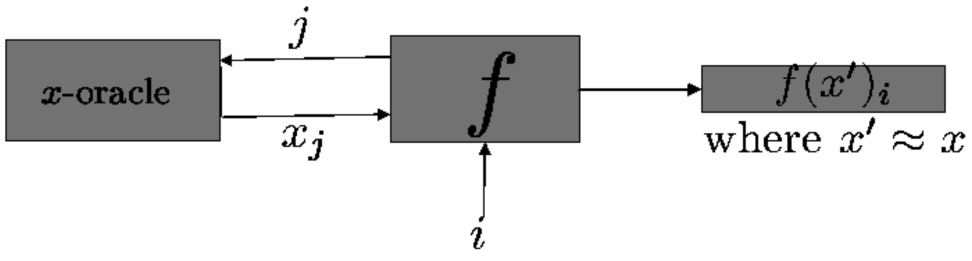

Figure 11-1: Sublinear Time Algorithms

- The output is not explicitly written by the algorithm, instead it is only implicitly given by the algorithm. Formally, on being queried for index i of the output string $f(x)$ (for some $i \in [n]$), the algorithm outputs the bit $f(x)_i$. Thus, the algorithm itself behaves as an oracle for the string $f(x)$, which in turn has oracle access to the input oracle x.
  Thus, sublinear time algorithms are oracle machines with oracle access to the input string. As in the case of PCPs, all oracle machines considered in this part are *non-adaptive*. Representing both the input and output implicitly as suggested above, has the added advantage that these algorithms can now be composed in a very natural fashion.

- Since the algorithm does not read the entire input x, we cannot expect it compute the output

$f(x)$ exactly. We instead relax our guarantee on the output as follows: On input $x \in \{0,1\}^k$, the algorithm must compute $f(x')$ exactly for some $x' \in \{0,1\}^k$ that is ε-close to the actual input x. In other words, the algorithm computes functions on some approximation to the input instead of the input itself.

Figure 11-1 gives a pictorial description of a sublinear time algorithm with the above mentioned relaxations.

Sublinear time algorithms were first studied in the context of program checking/testing and interactive proofs and PCPs. More recently, sublinear time algorithms are widely used in property testing, large graph problems, web-based algorithms, sorting, searching, high-dimensional computational geometry, statistics/entropy computations etc. In the context of coding theory, we will study whether each of the three algorithmic tasks mentioned in the earlier section (Sec. 11.1) can be performed in sublinear time.

Encoding: For a code to have good error-correcting properties, most bits of the codeword needs to depend on most message-bits. Taking this into account, it does not seem reasonable to expect a "good" code to have sublinear time encoding.

Decoding: Several codes have very efficient sublinear decoding procedures. In fact, many of the initial results in decoding actually yield sublinear time decoding procedures. Informally, a code that admits sublinear time decoding is called a locally decodable code.

Testing: Like decoding, sublinear time testing is also an interesting problem and several codes admit sublinear time testing. Informally, a a code that admits sublinear time testing is called a locally testable code.

In the next two sections, we will take a closer look at these types of codes (locally testable codes and locally decodable codes) and explain why they are interesting both from an algorithmic and complexity theoretic perspective. For further applications of coding theory to complexity theory, please refer to the excellent survey by Trevisan [Tre04].

11.3 Locally Testable Codes

Loosely speaking, a codeword test (for a code C) is a randomized (non-adaptive) oracle machine that is given oracle access to a string. The tester may query the oracle at a constant number of bit-locations and is required to (always) accept every codeword and reject with (relatively) high probability every string that is "far" from the code. A code C is said to be locally testable if there exists such a tester for the code C. Observe that according to this definition, the local testability of a code is based on the query complexity of the tester and not on the running time of the tester. If furthermore the tester also runs in sublinear time, we say that the code is *efficiently locally testable*. This notion of local testability was hinted at in the work of Babai *et al.* [BFLS91]. It was later formalized in the works of Rubinfeld and Sudan [RS96], Arora [Aro95], Spielman [Spi95] and Friedl and Sudan [FS95]. The interest in locally testable codes was revived by the recent works of Goldreich and Sudan [GS02] and Ben-Sasson *et al.* [BSVW03].

We do not know of any generic applications of locally testable codes, however they are interesting objects in their own right. Locally testable codes are intimately connected to PCPs. In fact, the notion of locally testable codes was introduced to study the possible limitations of PCP constructions. Surprisingly, this study, on the contrary, has only led to improvements in PCP constructions. Locally testable codes do not have any algorithmic applications at present. One of the reasons for this might be that the present constructions of such codes involve such a huge blowup in the message length (i.e., rate of code is inverse polynomial) that it makes it infeasible for any practical application. However, there does not seem to be any inherent lower-bound on the blowup in message length of locally testable codes. It is possible that there exist locally testable codes with constant rate, good distance and testable with a constant number of queries. Some potential algorithmic applications of such codes are explained below.

- Efficient Spam Filters: Current spam-filters need to read the entire email to check for possible spam. If efficient locally testable codes with constant rate exist, then it is possible to have spam-filters that distinguish between spam-free email and far from spam-free email by merely probing the email at a constant number of locations.

- Efficient Hard-disk Virus Scans: Current Virus Scans for the hard disks consume a lot of time to check if the disk is infected or not since they have to read every byte of information on the disk. However, if constant rate locally testable codes exist, it is possible to encode the contents of the disk according to such a locally testable code so that the resulting encoding can be scanned for possible virus infections by merely probing the disk at a constant number of locations. A complete scan of the disk could then be run at less frequent intervals while a scan using locally testable codes can be performed more frequently without consuming too much time.

We defer the formal definition of locally testable codes to Chapter 12, where we also present our improved constructions of such codes via PCPs of proximity.

11.4 Locally Decodable Codes

Loosely speaking, a code is said to be locally decodable if whenever a few locations are corrupted, the decoder is able to recover each information-bit, with high probability, based on a constant number of queries to the (corrupted) codeword. This notion was formally defined by Katz and Trevisan [KT00]. As in the case of locally testable codes, observe that the notion of local decodability is defined not on the basis of the running time of the decoder but rather on the basis of the query complexity of the decoder. Again as before, if a code admits a local decoder that runs in sublinear time, we say that the code is *efficiently locally decodable*.

Though locally decodable codes were defined by Katz and Trevisan [KT00] only in 2000, their constructions predate this definition. Goldreich and Levin in their work on the hardcore predicates [GL89] showed that the Hadamard code is efficiently locally decodable. Beaver and Feigenbaum [BF90], Lipton [Lip91], Blum, Luby and Rubinfeld [BLR93] and Chor *et al.* [CGKS98] gave several constructions of locally decodable codes in the context of program checking, testing and private information retrieval. Babai *et al.* [BFLS91] in their work on program checking hinted at a possible definition and an interpretation of locally decodable codes in the context of error-correcting codes. The definition of locally decodable codes can be extended to the case when a lot of bits of the codeword are corrupted. In such a case, the corrupted codeword can no longer be uniquely decoded but can instead only be list-decoded. Surprisingly, the notion of local-list-decodability was introduced by Sudan, Trevisan and Vadhan [STV01] before the formal definition of local-decodability due to Katz and Trevisan [KT00]. Locally decodable codes appeared in various forms due to their applicability to worst-case to average-case analysis [Lip91, STV01], information hiding [BF90] and private information retrieval [CGKS98].

Katz and Trevisan in their seminal work on locally decodable codes also proved that if the decoder makes q queries then the length of the code is at least $n = \Omega(k^{1+1/(q-1)})$. This lower bound is far from the best known upper bound, due to Beimal *et al.* [BIKR02] that asserts $n = O(\exp(k^{o(1/q)}))$ which barely improves on the Hadamard code which satisfies $n = 2^k$. Unable to improve either the lower-bound or upper-bound, we instead introduce a new relaxed notion of locally decodable codes, with the hope of obtaining more efficient constructions (i.e., $n = \mathrm{poly}(k)$).

11.4.1 Relaxed Locally Decodable Codes

We relax the definition of locally decodable codes by requiring that, whenever a few locations are corrupted, the decoder should be able to recover most of the individual information-bits (based on a few queries) and for the rest of the locations, the decoder may output either the right message bit or a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but is allowed to say "`don't know`" on a few bit-locations. We show that this relaxed notion of local decodability can be supported by codes that have codewords of length that is almost-linear in the number of information bits (i.e., $n = k^{1+\varepsilon}$ for every $\varepsilon > 0$).

We defer the formal definition of relaxed locally decodable codes to Chapter 13, where we also present our construction of such codes using PCPs of proximity.

CHAPTER 12

Locally Testable Codes

In this chapter, we show that, combined with any good code, any PCP of proximity yields a Locally Testable Code (LTC). Using our PCPs of proximity, we obtain an improvement in the rate of LTCs (improving over the results of [GS02, BSVW03]).

12.1 Definitions

Loosely speaking, by a codeword test (for the code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$) we mean a randomized (non-adaptive) oracle machine, also called a tester, that is given oracle access to $w \in \{0,1\}^n$. The tester may query the oracle at a constant number of bit-locations and is required to (always) accept every codeword and reject with (relatively) high probability every oracle that is "far" from the code. Indeed, since our focus is on positive results, we use a strict formulation in which the tester is required to accept each codeword with probability 1. (This corresponds to "perfect completeness" in the PCP setting.) The first definition below provides a general template (in terms of several parameters) for the rejection condition. Later we will discuss the kinds of asymptotic parameters we would like to achieve.

Definition 12.1.1 (codeword tests): *A randomized* (non-adaptive) *oracle machine M is called a* (δ, s)-codeword test for $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *if it satisfies the following two conditions:*

1. Accepting codewords (aka completeness)*: For every $x \in \{0,1\}^k$, given oracle access to $w = \mathrm{C}(x)$, machine M accepts with probability 1. That is,* $\Pr[M^{\mathrm{C}(x)}=1] = 1$, *for every $x \in \{0,1\}^k$.*
2. Rejection of non-codeword (aka soundness)*: Given oracle access to any $w \in \{0,1\}^n$ that is δ-far from* C, *machine M accepts with probability at most s. That is,* $\Pr[M^w = 1] \leq s$, *for every*

$w \in \{0,1\}^n$ *that is* δ*-far from* C.

The parameter δ *is called the* proximity parameter *and* s *is called the* soundness error. *The* query complexity q *of* M *is the maximum number of queries it makes (taken over all sequences of coin tosses).*

Note that *this definition requires nothing with respect to non-codewords that are relatively close to the code* (i.e., are δ-close to C). In addition to the usual goals in constructing error-correcting codes (e.g., maximizing minimum distance and minimizing the blocklength $n = n(k)$), here we are also interested in simultaneously minimizing the query complexity q, the proximity parameter δ, and the soundness error s. More generally, we are interested in the tradeoff between q, δ, and s. (As usual, the soundness error can be reduced to s^k by increasing the query complexity to $k \cdot q$.) A minimalistic goal is to have a family of codes with q, δ, and s all fixed constants. However, note that this would only be interesting if δ is sufficiently small with respect to the distance parameters of the code, e.g. smaller than half the relative minimum distance. (For example, if δ is larger than the "covering radius" of the code, then there does not exist any string that is δ-far from the code, and the soundness condition becomes vacuous.) A stronger definition requires the tester to work for any *given* proximity parameter $\delta > o(1)$, but allows its query complexity to depend on δ:

Definition 12.1.2 (locally testable codes): *A family of codes* $\{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^n\}_{k \in \mathbb{N}}$ *is* locally testable *if it satisfies*

1. *Linear Distance: There is a constant* $\rho > 0$*, such that for every* k*,* C_k *has minimum distance at least* $\rho \cdot n$*.*
2. *Local Testability: There is a randomized, non-adaptive oracle machine* M *such that for every constant* $\delta > 0$*, there is a constant* $q = q(\delta)$ *such that for all sufficiently large* k*,* $M^w(1^k, \delta)$ *is a* $(\delta, 1/2)$*-codeword test for* C_k *with query complexity* q*.*

The family is called explicit *if both* C_k *and* $M^w(1^k, \delta)$ *can be evaluated with computation time polynomial in* k*.*

We comment that Definition 12.1.2 is somewhat weaker than the definitions used in [GS02].[1]

12.2 *Constructions*

Using an adequate PCP of proximity, we can transform any code to a related code that has a codeword tester. This is done by appending each codeword with a PCP of proximity proving the codeword is indeed the encoding of a message. One technical problem that arises is that the PCP of

[1] In the weaker among the definitions in [GS02], the tester is not given δ as input (and thus has query complexity that is a fixed constant independent of δ) but is required to be a $(\delta, 1 - \Omega(\delta))$-codeword test for every constant $\delta > 0$ and sufficiently large k. That is, strings that are δ-far from the code are rejected with probability $\Omega(\delta)$. Such a tester implies a tester as in Definition 12.1.2, with query complexity $q(\delta) = O(1/\delta)$.

proximity constitutes most of the length of the new encoding. Furthermore, we cannot assume much about the Hamming distance between different proofs of the same statement, thus the distance of the new code may deteriorate. But this is easily fixed by repeating the codeword many times, so that the PCP of proximity constitutes only a small fraction of the total length.[2] Specifically, given a code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$, we consider the code $\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^t, \pi(x))$, where $t = (d(k) - 1) \cdot |\pi(x)|/|\mathrm{C}_0(x)|$ such that (say) $d(k) = \log k$, and $\pi(x)$ is a PCP of proximity that asserts that an m-bit string (given as an input oracle) is a codeword (of C_0).

Construction 12.2.1 *Let d be a free parameter to be determined later, $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ be a code, and V be a PCP of proximity verifier for membership in $S_0 = \{\mathrm{C}_0(x) : x \in \{0,1\}^k\}$. Let $\pi(x)$ be the proof-oracle corresponding to the claim that the input-oracle equals $\mathrm{C}_0(x)$; that is, $\pi(x)$ is the canonical proof obtained by using x as an NP-proof for membership of $\mathrm{C}_0(x)$ in S_0. Consider the code $\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^t, \pi(x))$, where $t = (d-1) \cdot |\pi(x)|/|\mathrm{C}_0(x)|$.*

The codeword test emulates the PCP-verifier in the natural way. Specifically, given oracle access to $w = (w_1, ..., w_t, \pi) \in \{0,1\}^{t \cdot m + \ell}$, the codeword tester selects uniformly $i \in [t]$, and emulates the PCP-verifier providing it with oracle access to the input-oracle w_i and to the proof-oracle π. In addition, the tester checks that the repetitions are valid (by inspecting randomly selected positions in some q_{rep} randomly selected pairs of m-bit long blocks, where q_{rep} is a free parameter to be optimized later). Let us denote this tester by T. That is, T^w proceeds as follows

1. Uniformly selects $i \in [t]$ and invokes $V^{w_i, \pi}$.

2. Repeats the following q_{rep} times: Uniformly selects $i_1, i_2 \in [t]$ and $j \in [m]$ and checks whether $(w_{i_1})_j = (w_{i_2})_j$.

Proposition 12.2.2 *Let d and q_{rep} be the free parameters in the above construction of the code C and tester T. Suppose that the code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ has relative minimum distance ρ_0, and that the PCP of proximity has proof length $\ell > m$, soundness error $1/4$ for proximity parameter δ_{pcpp} and query complexity q_{pcpp}. Then, the code C and tester T have the following properties:*

1. *The blocklength of C is $n \triangleq d \cdot \ell$ and its relative minimum distance is at least $\rho_0 - 1/d$.*

2. *The oracle machine T is a $(\delta, 1/2)$-codeword tester for the C, where $\delta = \delta_{\mathrm{pcpp}} + \frac{4}{q_{\mathrm{rep}}} + \frac{1}{d}$.*

3. *The query complexity of T is $q = q_{\mathrm{pcpp}} + 2q_{\mathrm{rep}}$.*

[2]Throughout this section we will use repetitions to adjust the "weights" of various parts of our codes. An alternative method would be to work with weighted Hamming distance (i.e. where different coordinates of a codeword receive different weights), and indeed these two methods (weighting and repeating) are essentially equivalent. For the sake of explicitness we work only with repetitions.

Proof: The parameters of the code C are obvious from the construction. In particular, C has blocklength $t \cdot m + \ell = d \cdot \ell = n$, and the PCP of proximity $\pi(x)$ constitutes only an $\ell/n = 1/d$ fraction of the length the codeword $\mathrm{C}(x)$. Since the remainder consists of replicated versions of $\mathrm{C}_0(x)$, it follows that the relative minimum distance of C is at least $(n - \ell)\rho_0/n > \rho_0 - 1/d$.

The query complexity of T is obvious from its construction, and so we only need to show that it is a good codeword tester. Completeness follows immediately from the completeness of the PCP of proximity, and so we focus on the soundness condition. We consider an arbitrary $w = (w_1, ..., w_t, \pi) \in \{0,1\}^{t \cdot m + \ell}$ that is δ-far from C, and observe that $w' = (w_1, ..., w_t)$ must be δ'-far from $\mathrm{C}' = \{\mathrm{C}_0(x)^t : x \in \{0,1\}^k\}$, where $\delta' \geq (\delta n - \ell)/n = \delta - (1/d)$. Let $u \in \{0,1\}^m$ be a string that minimizes $\overline{\Delta}(w', u^t) = \sum_{i=1}^{t} \overline{\Delta}(w_i, u)$; that is, u^t is the "repetition sequence" closest to w'. We consider two cases:

Case 1: $\overline{\Delta}(w', u^t) \geq tm/q_{\mathrm{rep}}$. In this case, a single execution of the basic repetition test (comparing two locations) rejects with probability:

$$\begin{aligned} \mathrm{E}_{r,s\in[t]}\left[\overline{\Delta}(w_r, w_s)/m\right] &\geq \mathrm{E}_{r\in[t]}\left[\overline{\Delta}(w_r, u)/m\right] \\ &= \overline{\Delta}(w', u^t)/(t \cdot m) \\ &\geq 1/q_{\mathrm{rep}} \end{aligned}$$

where the last inequality is due to the case hypothesis. It follows that q_{rep} executions of the repetition test would accept with probability at most $(1 - 1/q_{\mathrm{rep}})^{q_{\mathrm{rep}}} < 1/e < 1/2$.

Case 2: $\overline{\Delta}(w', u^t) \leq tm/q_{\mathrm{rep}}$. In this case

$$\frac{\overline{\Delta}_{\mathrm{C}_0}(u)}{m} = \frac{\overline{\Delta}_{\mathrm{C}'}(u^t)}{tm} \geq \frac{\overline{\Delta}_{\mathrm{C}'}(w') - \overline{\Delta}(w', u^t)}{tm} \geq \delta' - \frac{1}{q_{\mathrm{rep}}}$$

where the last inequality is due to the case hypothesis. Also, recalling that on the average (i.e., average i) w_i is $1/q_{\mathrm{rep}}$-close to u, it holds that at least two thirds of the w_i's are $3/q_{\mathrm{rep}}$-close to u. Recalling that u is $(\delta' - (1/q_{\mathrm{rep}}))$-far from C_0 and using $\delta_{\mathrm{pcpp}} = \delta' - (4/q_{\mathrm{rep}})$, it follows at least two thirds of the w_i's are δ_{pcpp}-far from C_0. Thus, by the soundness condition of the PCP of proximity, these w_i will be accepted with probability at most $1/4$. Thus, in the current case, the tester accepts with probability at most $\frac{1}{3} + \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{2}$.

The soundness condition follows. ■

To prove Theorem 1.3.4, we instantiate the above construction as follows. We let $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ come from a family of codes with constant relative minimum distance $\rho_0 > 0$ and nearly linear blocklength $m = \widetilde{O}(k)$, where encoding can be done by circuits of nearly linear size $s_0 = s_0(k) = \widetilde{O}(k)$. We take the PCP of proximity from Corollary 10.2.2, setting $t_1 = O(1/\varepsilon)$ (for an arbitrarily small constant $\varepsilon > 0$) and $t_2 = 2\mathrm{loglog}s_0/\log\log\log s_0 = \omega(1)$. Thus, we obtain proof length $\ell = s_0 \cdot \exp(\log^{\varepsilon/2} s_0)$ and query complexity $q_{\text{pcpp}} = O(\max\{1/\delta_{\text{pcpp}}, t_1\}) = O(1/\delta_{\text{pcpp}})$ for any proximity parameter $\delta_{\text{pcpp}} \geq 1/t_2 = o(1)$. We actually invoke the verifier twice to reduce its soundness error to $1/4$. Setting $d = \log k = \omega(1)$, we obtain final blocklength $n = d{\cdot}\ell < k{\cdot}\exp(\log^{\varepsilon} k)$ and relative distance $\rho_0 - o(1)$. We further specify the test T as follows. Given a proximity parameter $\delta \geq 6/t_2 = o(1)$, the tester T invokes the aforementioned PCPP with $\delta_{\text{pcpp}} = \delta/6$, and performs the repetition test $q_{\text{rep}} = 6/\delta$ times. Observing that $\delta_{\text{pcpp}} + (4/q_{\text{rep}}) + (1/d) < \delta$, we conclude that the resulting test (i.e., $T = T(1^k, \delta_{\text{pcpp}})$) is a $(\delta, 1/2)$-codeword tester of query complexity $O(1/\delta_{\text{pcpp}}) + 2q_{\text{rep}} = O(1/\delta)$. Thus we conclude:

Conclusion (Restating Theorem 1.3.4): *For every constant $\varepsilon > 0$, there exists a a family of locally testable codes $\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^n$, where $n = \exp(\log^{\varepsilon} k) \cdot k$, with query complexity $q(\delta) = O(1/\delta)$.*

CHAPTER 13

Relaxed Locally Decodable codes

In this chapter, we introduce a relaxed notion of Locally Decodable Codes, and show how to construct such codes using any PCP of proximity (and ours in particular).

13.1 Definitions

We first recall the definition of Locally Decodable Codes (LDCs), as formally stated by Katz and Trevisan [KT00]. A code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ is locally decodable if for some constant $\delta > 0$ (which is independent of k) there exists an efficient oracle machine M that, on input any index $i \in [k]$ and access to any oracle $w \in \{0,1\}^n$ such that $\overline{\Delta}(w, \mathrm{C}(x)) \leq \delta n$, recovers the i-th bit of x with probability at least $2/3$ while making a constant number of queries to w. That is, whenever relatively few location are corrupted, the decoder should be able to recover each information-bit, with high probability, based on a constant number of queries to the (corrupted) codeword.

Katz and Trevisan showed that if M makes q queries then $n = \Omega(k^{1+1/(q-1)})$ must hold [KT00].[1] This lower-bound is quite far from the best known upper-bound, due to Beimal *et al.* [BIKR02], that asserts $n = O(\exp(k^{\varepsilon(q)}))$, where $\varepsilon(q) = O((\log\log q)/(q \log q)) = o(1/q)$, which improves (already for $q = 4$) over a previous upper-bound where $\varepsilon(q) = 1/(2q+1)$. It has been conjectured that, for a constant number of queries, n should be exponential in k; that is, for every constant q there exists a constant $\varepsilon > 0$ such that $n > \exp(k^\varepsilon)$ must hold. In view of this state of affairs, it is natural to relax

[1]Their lower-bound refers to non-adaptive decoders, and yields a lower-bound of $n = \Omega(k^{1+1/(2^q-1)})$ for adaptive decoders. A lower-bound of $n = \Omega(k^{1+1/O(q)})$ for adaptive decoders was presented in [DJK+02], and lower-bound of $n = \Omega(k^{1+1/(q/2-1)})$ for non-adaptive decoders was presented in [KdW03]. (We note that below we use a non-adaptive (relaxed) decoder.)

the definition of Locally Decodable Codes, with the hope of obtaining more efficient constructions (e.g., $n = \mathrm{poly}(k)$).

We relax the definition of Locally Decodable Codes by requiring that, whenever few location are corrupted, the decoder should be able to recover most (or almost all) of the individual information-bits (based on few queries) and for the remaining locations the decoder may output either the right message bit or a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but is allowed to say "`don't know`" on a few bit-locations. The following definition is actually weaker; yet, the (aforementioned) stronger formulation is obtained when considering $\rho \approx 1$ (and using amplification to reduce the error from $1/3$ to any desired constant).[2] Furthermore, it is desirable to recover all bits of the information, whenever the codeword is not corrupted.

Definition 13.1.1 (Relaxed LDC) *A code* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *is* relaxed locally decodable *if for some constants* $\delta, \rho > 0$ *there exists an efficient probabilistic oracle machine* M *that makes a constant number of queries and satisfies the following three conditions with respect to any* $w \in \{0,1\}^n$ *and* $x \in \{0,1\}^k$ *such that* $\overline{\Delta}(w, \mathrm{C}(x)) \leq \delta n$*:*

1. *If* $w = \mathrm{C}(x)$ *is a codeword then the decoder correctly recovers every bit of* x *with probability at least* $2/3$*. That is, for every* $x \in \{0,1\}^k$ *and* $i \in [k]$*, it holds that* $\Pr[M^{\mathrm{C}(x)}(i) = x_i] \geq \frac{2}{3}$.

2. *On input any index* $i \in [k]$ *and given access to the oracle* w*, with probability at least* $2/3$ *machine* M *outputs either the* i*-th bit of* x *or a special failure symbol, denoted* $\perp$*. That is, for every* i*, it holds that* $\Pr[M^w(i) \in \{x_i, \perp\}] \geq \frac{2}{3}$.

3. *For at least a* ρ *fraction of the indices* $i \in [k]$*, on input* i *and oracle access to* $w \in \{0,1\}^n$*, with probability at least* $2/3$*, machine* M *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* $\Pr[M^w(i) = x_i] \geq \frac{2}{3}$.

We call δ *the* proximity parameter*.*

One may strengthen the definition by requiring that ρ be greater than $1/2$ or any other favorite constant smaller than 1 (but probably refrain from setting $\rho > 1 - \delta$ or so). A different strengthening is for Condition 1 to hold with probability 1 (i.e., $\Pr[M^{\mathrm{C}(x)}(i) = x_i] = 1$). In fact, we achieve both the stronger forms.

Remark 13.1.2 *The above definition refers only to strings* w *that are* δ*-close to the code. However, using Construction 12.2.1, any relaxed LDC can be augmented so that strings that are* δ*-far from the code are*

[2]Here error reduction may be performed by estimating the probability that the machine outputs each of the possible bits, and outputting the more frequent bit only if it has sufficient statistical support (e.g., say 50% support, which the wrong bit cannot have). Otherwise, one outputs the `don't know` symbol.

rejected with high probability (i.e., for every index i, the decoder outputs $\bot$ with high probability). This can be achieved with only a nearly linear increase in the length of the code (from length n to length $n \cdot \exp(\log^{\varepsilon} n)$).

Remark 13.1.3 *We stress that Condition 2 does* NOT *mean that, for every i and w that is δ-close to $\mathrm{C}(x)$, either $\Pr[M^w(i) = x_i] \geq \frac{2}{3}$ or $\Pr[M^w(i) = \bot] \geq \frac{2}{3}$ holds. We refer to the latter condition as Condition X, and conjecture that the seemingly minor difference between Conditions 2 and X is actually substantial. This conjecture is enforced by a recent work of Buhrman and de Wolf [BdW04] who showed that codes that satisfy Condition X are actually locally decodable in the standard, non-relaxed sense* (i.e., according to the definition of [KT00]).

13.1.1 Definitional issues and transformations

Note that it is very easy to come up with constructions that satisfy each one of the three conditions of Definition 13.1.1. For example, Condition 2 can be satisfied by (any code and) a trivial decoder that always returns $\bot$. On the other hand, the identity encoding (combined with a trivial decoder) satisfies Conditions 1 and 3.[3] Our aim, however, is to obtain a construction that satisfies all conditions and beats the performance of the known locally decodable codes.

It turns out that codes that satisfy Conditions 1 and 2 can be converted into "equally good" codes that satisfy all three conditions. Let us start with a key definition, which refers to the distribution of the decoder's queries *when asked to recover a random bit position*.

Definition 13.1.4 (Average smoothness) *Let M be a randomized non-adaptive oracle machine having access to an oracle $w \in \{0,1\}^n$ and getting input $i \in [k]$. Further suppose that M always makes q queries. Let $M(i,j,r)$ denote the j-th query of M on input i and coin tosses r. We say that M satisfies the* average smoothness *condition if, for every $v \in [n]$,*

$$\frac{1}{2n} < \Pr_{i,j,r}[M(i,j,r) = v] < \frac{2}{n}$$

where the probability is taken uniformly over all possible choices of $i \in [k]$, $j \in [q]$, and coin tosses r.

By having M randomly permute its queries, average smoothness implies that for every $j \in [q]$ and $v \in [n]$, it holds that $\frac{1}{2n} < \Pr_{i,r}[M(i,j,r) = v] < \frac{2}{n}$, where now the probability is taken uniformly over all possible choices of $i \in [k]$ and the coin tosses r). We stress that *average smoothness is different from the notion of smoothness as defined by Katz and Trevisan* [KT00]: They require that *for every $i \in [k]$* (and for every $j \in [q]$ and $v \in [n]$), it holds that $\frac{1}{2n} < \Pr_r[M(i,j,r) = v] < \frac{2}{n}$. Indeed, average smoothness is a weaker requirement, and (as we will shortly see) any code and decoder pair can be easily modified to satisfy it, while preserving decoding properties. (In contrast, Katz

[3]In case one wishes the code to have a linear distance this can be achieved too: Consider $\mathrm{C}(x) = (x, \mathrm{C}'(x))$, where C' is any code of linear length and linear distance, and a decoder that merely retrieves the desired bit from the first part.

and Trevisan [KT00] present a modification that achieves smoothness while preserving strict local-decodability, but their transformation does not preserve Definition 13.1.1.)

Lemma 13.1.5 *Let* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *be a code and* M *be a machine that satisfies Conditions 1 and 2 of Definition 13.1.1 with respect to proximity parameter* δ*. Then, for some* $n' \in [3n, 4n]$*, there exists a code* $\mathrm{C}' : \{0,1\}^k \to \{0,1\}^{n'}$ *and a machine* M' *that satisfies average smoothness as well as Conditions 1 and 2 of Definition 13.1.1 with respect to proximity parameter* $\delta' = \delta/20$*. Furthermore, the query complexity of* M' *is twice the one of* M*, and if* M *satisfies also Condition 3, with respect to a constant* ρ*, then so does* M'*.*

Jumping ahead, we mention that for a decoder that satisfies average smoothness, Conditions 1 and 2 essentially imply Condition 3, hence our interest in Lemma 13.1.5.

Proof: As noted above, we may assume without loss of generality that each of M''s queries is distributed identically. Throughout the analysis, we refer to the distribution of queries for a uniformly distributed index $i \in [k]$. Let q denote the query complexity of M.

We first modify M such that for a random $i \in [k]$, each query probes each possible location with probability $\Omega(1/n)$. This is done by adding q dummy queries, each being uniformly distributed. Thus, each location gets probed by each query with probability at least $1/2n$.

Next we modify the code and the decoder such that each location is probed with almost uniform distribution. The idea is to repeat heavily-probed locations for an adequate number of times, and have the decoder probe a random copy. Specifically, let p_v be the probability that location v is probed (i.e., $p_v \triangleq \Pr_{i\in[k],r}[M(i,1,r) = v]$ or equivalently $p_v = \sum_{i\in[k],j\in[2q]} \Pr_{i,j,r}[M(i,j,r) = v]/2kq$). By the above modification, we have $p_v \geq 1/2n$. Now, we repeat location v for $r_v = \lfloor 4np_v \rfloor$ times. Note that $r_v \leq 4np_v$ and $r_v > 4np_v - 1 \geq 2 - 1$ (and so $r_v \geq 2$). We obtain a new code C' of length $n' = \sum_v r_v \leq 4n$. (Note that $n' > 3n$.) The relative distance of C' is at least one fourth that of C, and the rate changes in the same way. The new decoder, M', when seeking to probe location v will select and probe at random one of the r_v copies of that location. (Interestingly, there is no need to augment this decoder by a testing of the consistency of the copies of an original location.)

Each new location is probed with probability $p'_v \triangleq p_v \cdot \frac{1}{r_v}$ (by each of these queries). Recalling that $\frac{p_v}{r_v} = \frac{p_v}{\lfloor 4np_v \rfloor}$, it follows that $p'_v \geq 1/4n$ and $p'_v \leq \frac{p_v}{4np_v - 1} \leq 1/2n$ (using $p_v \geq 1/2n$). Recalling that $n' \in [3n, 4n]$, each p'_v is in $[(3/4)\cdot(1/n'), 2\cdot(1/n')]$, i.e., within a factor of 2 from uniform.

Clearly, M' satisfies Condition 1 (of Definition 13.1.1) and we turn to show that it (essentially) satisfies Condition 2 as well. Let $w = (w_1, ..., w_n) \in \{0,1\}^{n'}$ be δ'-close to $\mathrm{C}'(x)$, where $|w_v| = r_v$. Let Y_v be a 0-1 random variable that represents the value of a random bit in w_v; that is, $\Pr[Y_v = 1]$ equals the fraction of 1's in w_v. Then, $\Pr[Y_v \neq \mathrm{C}(x)_v] > 0$ implies that $\overline{\Delta}(w_v, c_v) \geq 1$, where $\mathrm{C}'(x) = (c_1, ..., c_n)$ and $|c_v| = r_v$. For $Y = Y_1 \cdots Y_n$, it follows that $\mathrm{E}(\overline{\Delta}(Y, \mathrm{C}(x))) \leq \overline{\Delta}(w, \mathrm{C}'(x))$, and so $\mathrm{E}(\overline{\Delta}(Y, \mathrm{C}(x))) \leq \delta' n' \leq \frac{\delta}{5}\cdot n$ (since $\delta' = \delta/20$ and $n' \leq 4n$). Thus, with probability at least

$4/5$, the random string Y is δ-close to $\mathrm{C}(x)$, in which case the M must succeed with probability at least $2/3$. Noting that $M'^w(i)$ merely invokes $M^Y(i)$, we conclude that

$$\begin{aligned}
\Pr[M'^w(i) \in \{x_i, \perp\}] &= \Pr[M^Y(i) \in \{x_i, \perp\}] \\
&\geq \Pr[\overline{\Delta}(Y, \mathrm{C}(x)) \leq \delta n] \cdot \Pr[M^Y(i) \in \{x_i, \perp\} \,|\, \overline{\Delta}(Y, \mathrm{C}(x)) \leq \delta n] \\
&\geq \frac{4}{5} \cdot \frac{2}{3} = \frac{8}{15}
\end{aligned}$$

An analogous argument can be applied in the case M satisfies Condition 3. In both cases, additional error-reduction is needed in order to satisfy the actual conditions, which require success with probability at least $2/3$. (For details see Footnote 2.) ∎

Lemma 13.1.6 *Let* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *be a code and* M *be a machine that satisfies Conditions 1 and 2 of Definition 13.1.1 with respect to a constant* δ*. Suppose that* M *satisfies the average smoothness condition and has query complexity* q*. Then, invoking* M *for a constant number of times* (and ruling as in Footnote 2) *yields a decoder that satisfies all three conditions of Definition 13.1.1. Specifically, Condition 3 holds with respect to a constant* $\rho = 1 - 18q\delta$*. Furthermore, for any* w *and* x*, for* $1 - 18q\overline{\Delta}(w, \mathrm{C}(x))$ *fraction of the* i*'s, it holds that* $\Pr[M^w(i) = x_i] \geq 5/9$*.*

Our usage of the average smoothness condition actually amounts to using the hypothesis that, for a uniformly distributed $i \in [k]$, each query hits any fixed position with probabilty at most $2/n$.

Proof: By Condition 1, for any $x \in \{0,1\}^k$ and every $i \in [k]$, it holds that $\Pr[M^{\mathrm{C}(x)}(i) = x_i] \geq 2/3$. Considering any w that is δ-close to $\mathrm{C}(x)$, the probability that on input a *uniformly distributed* $i \in [k]$ machine M queries a location on which w and $\mathrm{C}(x)$ 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, the queries are almost uniformly distributed; specifically, no position is queried with probabilty greater than $2/n$ (by a single query).

Let p_i^w denote the probability that on input i machine M queries a location on which w and $\mathrm{C}(x)$ disagree. We have just established that $(1/k) \cdot \sum_{i=1}^k p_i^w \leq 2q\delta$. For $I_w \triangleq \{i \in [k] : p_i^w \leq 1/9\}$, it holds that $|I_w| \geq (1 - 18q\delta) \cdot k$. Observe that for any $i \in I_w$, it holds that $\Pr[M^w(i) = x_i] \geq (2/3) - (1/9) = 5/9$. Note that, by replacing δ with $\overline{\Delta}(w, \mathrm{C}(x))/n$, the above argument actually establishes that for $1 - 18q \cdot \overline{\Delta}(w, \mathrm{C}(x))$ fraction of the i's, it holds that $\Pr[M^w(i) = x_i] \geq 5/9$.

Additional error-reduction is needed in order to satisfy the actual definition (of Condition 3), which require success with probability at least $2/3$. The error-reduction should be done in a manner that preserves Conditions 1 and 2 of Definition 13.1.1. For details see Footnote 2. ∎

In view of the furthermore clause of Lemma 13.1.6, it makes sense to state a stronger definition of relaxed locally decodable codes.

Definition 13.1.7 (Relaxed LDC, revisited) *A code* $C : \{0,1\}^k \to \{0,1\}^n$ *is* relaxed locally decodable *if for some constants* $\delta > 0$ *there exists an efficient probabilistic oracle machine* M *that makes a constant number of queries and satisfies the following two conditions with respect to any* $w \in \{0,1\}^n$ *and* $x \in \{0,1\}^k$ *such that* $\overline{\Delta}(w, C(x)) \le \delta n$*:*

1. *For every* $i \in [k]$ *it holds that* $\Pr[M^w(i) \in \{x_i, \perp\}] \ge \frac{2}{3}$.
2. *There exists a set* $I_w \subseteq [k]$ *of density at least* $1 - O(\overline{\Delta}(w, C(x))/n)$ *such that for every* $i \in I_w$ *it holds that* $\Pr[M^w(i) = x_i] \ge \frac{2}{3}$.

Note that the "everywhere good" decoding of codewords (i.e., Condition 1 of Definition 13.1.1) is implied by Condition 2 of Definition 13.1.7. By combining Lemmas 13.1.5 and 13.1.6, we get:

Theorem 13.1.8 *Let* $C : \{0,1\}^k \to \{0,1\}^n$ *be a code and* M *be a machine that makes a constant* q *number of queries and satisfies Conditions 1 and 2 of Definition 13.1.1 with respect to a constant* δ*. Then, for some* $n' \in [3n, 4n]$*, there exists a code* $C' : \{0,1\}^k \to \{0,1\}^{n'}$ *that is relaxed locally decodable with respect to proximity parameter* $\delta' = \delta/20$*. Furthermore, this code satisfies Definition 13.1.7.*

13.2 Constructions

Using Lemma 13.1.6, we focus on presenting codes with decoders that satisfy Conditions 1 and 2 of Definition 13.1.1 as well as the average smoothness property. We will start with a code that has nearly quadratic length (i.e., $n = k^{2+o(1)}$), which serves as a good warm-up towards our final construction in which $n = k^{1+\varepsilon}$, for any desired constant $\varepsilon > 0$.

Motivation to our construction: We seek a code of linear distance that has some weak "local decodability" properties. One idea is to separate the codeword into two parts, the first allowing for "local decodability" (e.g., using the identity map) and the second providing the distance property (e.g., using any code of linear distance). It is obvious that a third part that guarantees the consistency of the first two parts should be added, and it is natural to try to use a PCP of proximity in the latter part. The natural decoder will check consistency (via the PCPP), and in case it detects no error will decode according to the first part. Indeed, the first part may not be "robust to corruption" but the second part is "robust to corruption" and consistency means that both parts encode the same information. Considering this vague idea, we encounter two problems. First, a PCP of proximity is unlikely to detect a small change in the first part. Thus, if we use the identity map in the first part then the decoder may output the wrong value of some (although few) bits. Put in other words, the "proximity relaxation" in PCPPs makes sense for the second part of the codewords but not for the first part. Our solution is to provide, *for each bit* (position) in the first part, a proof of the consistency of this bit (value) with the entire second part. The second problem is that the PCPPs (let alone all of them combined) are much longer than the first two parts, whereas the corruption rate is measured in terms of the entire codeword. This problem is easy to fix by repeating the first two parts sufficiently many times. However, it is important not to "overdo" this repetition, because if the third part is too short, then corrupting it may prevent meaningful decoding (as per Condition 3 of Definition 13.1.1) even at low corruption rates (measured in terms of the entire codeword). Put in other words, if the third part too short then we have no chance to satisfy the average smoothness condition.

The actual construction. Let $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ be a good code of relative distance δ_0, then we encode $x \in \{0,1\}^k$ by $\mathrm{C}(x) \triangleq (x^t, \mathrm{C}_0(x)^{t'}, \pi_1(x), ..., \pi_k(x))$, where $t = |\pi_1(x), ..., \pi_k(x)|/|x|$ (resp., $t' = |\pi_1(x), ..., \pi_k(x)|/|\mathrm{C}_0(x)|$), and $\pi_i(x)$ is a PCP of proximity to be further discussed. We first note that the replicated versions of x (resp., $\mathrm{C}_0(x)$) takes a third of the total length of $\mathrm{C}(x)$. As for $\pi_i(x)$, it is a PCP of proximity that refers to an input of the form $(z_1, z_2) \in \{0,1\}^{m+m}$ and asserts that there exists an $x = x_1 \cdots x_k$ (indeed the one that is a parameter to π_i) such that $z_1 = x_i^m$ and $z_2 = \mathrm{C}_0(x)$.[4] We use our PCP of proximity from Theorem 10.2.1, while setting its parameters such that the proximity parameter is small enough but the query complexity is a constant. Specifically, let $\delta_{\text{pcpp}} > 0$ be the proximity parameter of the PCP of proximity, which will be set to be sufficiently small, and let $q = O(1/\delta_{\text{pcpp}})$ denote the number of queries the verifier makes in order to support a soundness error of $1/6$ (rather than the standard $1/2$). A key observation regarding this verifier is that its queries to its input-oracle are uniformly distributed. The queries to the the proof oracle can be made almost uniform by a modification analogous to the one used in the proof of Lemma 13.1.5.

Observe that the code C maps k-bit long strings to codewords of length $n \triangleq 3 \cdot k \cdot \ell$, where $\ell = s_0(m)^{1+o(1)}$ denotes the length of the PCPP-proof and $s_0(m)$ denotes the size of the circuit for encoding relative to C_0. Using a good code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ (i.e., of constant relative distance δ_0, linear length $m = O(k)$, and $s_0(m) = \widetilde{O}(m)$), we obtain $n = k^{2+o(1)}$. The relative distance of C is at least $\delta_0/3$.

We now turn to the description of the decoder D. Recall that a valid codeword has the form $(x^t, \mathrm{C}_0(x)^{t'}, \pi_1(x), ..., \pi_k(x))$. The decoding of the i-th information bit (i.e., x_i) will depend on a random (possiblly wrong) copy of x_i located in the first part (which supposedly equals x^t), a random (possibly corrupted) copy of $\mathrm{C}_0(x)$ located in the second part, and the relevant (i.e., i-th) proof located in the third part (hich is also possibly corrupted). On input $i \in [k]$ and oracle access to $w = (w_1, w_2, w_3) \in \{0,1\}^n$, where $|w_1| = |w_2| = |w_3|$, the decoder invokes the PCPP-verifier while providing it with access to an input-oracle (z_1, z_2) and a proof oracle π that are defined and emulated as follows: The decoder selects uniformly $r \in [t]$ and $r' \in [t']$, and defines each bit of z_1 to equal the $((r-1)k + i)$-th bit of w_1, the string z_2 is defined to equal the r'-th (m-bit long) block

[4]Indeed z_1 is merely the bit x_i repeated $|C_0(x)|$ times in order to give equal weight to each part in measuring proximity.

of w_2, and π is defined to equal the i-th block (ℓ-bit long) of w_3. That is, when the verifier asks to access the j-th bit of z_1 (resp., z_2) [resp., π], the decoder answers with the $((r-1)k+i)$-th bit of w_1 (resp., $((r'-1)m+j)$-th bit of w_2) [resp., the $((i-1)\ell+j)$-th bit of w_3]. If the verifier rejects then the decoder outputs a special (failure) symbol. Otherwise, it outputs the $((r-1)k+i)$-th bit of w_1.

The above construction can be performed for any sufficiently small constant proximity parameter $\delta \in (0, \delta_0/18)$. All that this entails is setting the proximity parameter of the PCPP to be sufficiently small but positive (e.g., $\delta_{\rm pcpp} = (\delta_0 - 18\delta)/2$). We actually need to augment the decoder such that it makes an equal number of queries to each of the three (equal length) parts of the codeword, which is easy to do by adding (a constant number of) dummy queries. Let us denote the resulting decoder by D.

Proposition 13.2.1 *The above code and decoder satisfy Conditions 1 and 2 of Definition 13.1.1 with respect to proximity parameter $\delta \in (0, \delta_0/18)$. Furthermore, this decoder satisfies the average smoothness property.*

Proof: Condition 1 (of Definition 13.1.1) is obvious from the construction (and the completeness property of the PCPP). In fact, the perfect completeness of the PCPP implies that bits of an uncorrupted codeword are recovered with probability one (rather than with probability at least $2/3$). The average smoothness property of the decoder is obvious from the construction and the smoothness property of the PCPP. We thus turn to establish Condition 2 (of Definition 13.1.1).

Fixing any $x \in \{0,1\}^k$, we consider an arbitrary oracle $w = (w_1, w_2, w_3)$ that is δ-close to $\mathrm{C}(x)$, where w_1 (resp., w_2) denotes the alleged replication of x (resp., $\mathrm{C}_0(x)$) and $w_3 = (u_1, ..., u_k)$ denotes the part of the PCPs of proximity. Note that w_2 is 3δ-close to $\mathrm{C}_0(x)^{t'}$. To analyze the performance of $D^w(i)$, we define random variables Z_1 and Z_2 that correspond to the input-oracles to which the PCP-verifier is given access. Specifically, $Z_1 = \sigma^m$, where σ is set to equal the $((r-1)k+i)$-th bit of w_1, when r is uniformly distributed in $[t]$. Likewise, Z_2 is determined to be the r'-th block of w_2, where r' is uniformly distributed in $[t']$. Finally, we set the proof-oracle, π, to equal the i-th block of w_3 (i.e., $\pi = u_i$). We bound the probability that the decoder outputs $\neg x_i$ by considering three cases:

Case 1: $\sigma = x_i$. Recall that σ is the bit read by D from w_1, and that by construction D always outputs either σ or $\perp$. Thus, in this case, Condition 2 is satisfied (because, regardless of whether D outputs σ or $\perp$, the output is always in $\{x_i, \perp\}$).

Case 2: Z_2 is 18δ-far from $\mathrm{C}_0(x)$. Recall that w_2 is 3δ-close to $\mathrm{C}_0(x)^{t'}$, which means that the expected relative distance of Z_2 and $\mathrm{C}_0(x)$ is at most 3δ. Thus, the current case occurs with probability at most $1/6$.

Case 3: Z_2 is 18δ-close to $\mathrm{C}_0(x)$ and $\sigma \neq x_i$. Then, on one hand, (Z_1, Z_2) is $1/2$-far from $(x_i^m, \mathrm{C}_0(x))$, because $Z_2 = \sigma^t$. On the other hand, Z_2 is $(\delta_0 - 18\delta)$-far from any other codeword of C_0, because Z_2 is 18δ-close to $\mathrm{C}_0(x)$ and the codewords of C_0 are δ_0-far from one another, Thus,

(Z_1, Z_2) is $(\delta_0 - 18\delta)/2$-far from any string of the form $(y_i^m, \mathrm{C}_0(y))$. Using $\delta_{\mathrm{pcpp}} \leq (\delta_0 - 18\delta)/2$, we conclude that the PCPP verifier accepts (Z_1, Z_2) with probability at most $1/6$. It follows that, in the current case, the decoder outputs $\neg x_i$ with probability at most $1/6$.

Thus, in total, the decoder outputs $\neg x_i$ with probability at most $1/6 + 1/6 = 1/3$. ∎

Improving the rate: The reason that our code has quadratic length codewords (i.e., $n = \Omega(k^2)$) is that we augmented a standard code with proofs regarding the relation of the standard codeword to the value of *each* information bit. Thus, we had k proofs each relating to a statement of length $\Omega(k)$. Now, consider the following improvement: Partition the message into $\sqrt{k}$ blocks, each of length $\sqrt{k}$. Encode the original message as well as each of the smaller blocks, via a good error correcting code. Let w be the encoding of the entire message, and w_i ($i = 1, ..., \sqrt{k}$) be the encodings of the blocks. For every $i = 1, ..., \sqrt{k}$, append a PCP of proximity for the claim "w_i is the encoding of the i-th block of a message encoded by w". In addition, for each message bit $x_{(i-1)\sqrt{k}+j}$ residing in block i, append a PCP of proximity of the statement "$x_{(i-1)\sqrt{k}+j}$ is the j-th bit of the $\sqrt{k}$-bit long string encoded in w_i". The total encoding length has decreased, because we have $\sqrt{k}$ proofs of statements of length $O(k)$ and k proofs of statements of length $O(\sqrt{k})$, leading to a total length that is almost linear in $k^{3/2}$.

In general, for any constant ℓ, we consider ℓ successively finer partitions of the message into blocks, where the $(i+1)$-st partition is obtained by breaking each block of the previous partition into $k^{1/\ell}$ equally sized pieces. Thus, the i-th partition uses $k^{i/\ell}$ blocks, each of length $k^{1-(i/\ell)}$. Encoding is done by providing, for each $i = 0, 1, ..., \ell$, encodings of each of the blocks in the i-th partition by a good error-correcting code. Thus, for $i = 0$ we provide the encoding of the entire messages, whereas for $i = \ell$ we provide an "encoding" of individual bits. Each of these $\ell + 1$ levels of encodings will be assigned equal weight (via repetitions) in the new codeword. In addition, the new codeword will contain PCPs of proximity that assert the consistency of "directly related" blocks (i.e., blocks of consecutive levels that contain one another). That is, for every $i = 1, ..., \ell$ and $j \in [k^{i/\ell}]$, we place a proof that the encoding of the j-th block in the i-th level is consistent with the encoding of the $\lceil j/k^{1/\ell} \rceil$-th block in the $(i-1)$-st level. The i-th such sequence of proofs contains $k^{i/\ell}$ proofs, where each such proof refers to statements of length $O(k^{1-(i/\ell)} + k^{1-((i-1)/\ell)}) = O(k^{1-((i-1)/\ell)})$, which yields a total length of proofs that is upper-bounded by $k^{i/\ell} \cdot (k^{1-((i-1)/\ell)})^{1+o(1)} = k^{1+(1/\ell)+o(1)}$. Each of these sequences will be assigned equal weight in the new codeword, and the total weight of all the encodings will equal the total weight of all proofs. The new decoder will just check the consistency of the ℓ relevant proofs and act accordingly. We stress that, as before, the proofs in use are PCPs of proximity. In the current context these proofs refer to two input-oracles of vastly different length, and so the bit-positions of the shorter input-oracle are given higher "weight" (by repetition) such that both input-oracles are assigned the same weight.[5]

Construction 13.2.2 *Let C_0 be a code of minimal relative distance δ_0, constant rate, and nearly linear-sized encoding circuits. For simplicity, assume that a single bit is encoded by repetitions; that is, $\mathrm{C}_0(\sigma) = \sigma^{O(1)}$ for $\sigma \in \{0,1\}$. Let V be a PCP of proximity of membership in $S_0 = \{\mathrm{C}_0(x) : x \in \{0,1\}^*\}$ having almost-linear proof-length, query-complexity $O(1/\delta_{\mathrm{pcpp}})$ and soundness error $1/9$, for proximity parameter δ_{pcpp}. Furthermore, V's queries to both its input-oracle and proof-oracle are distributed almost uniformly.*[6] *For a fixed parameter $\ell \in \mathbb{N}$, let $b \triangleq k^{1/\ell}$. For $x \in \{0,1\}^k$, we consider ℓ different partitions of x, such that the j-th partition denoted $(x_{j,1}, ..., x_{j,b^j})$, where $x_{j,j'} = x_{(j'-1)\cdot b^{\ell-j}+1} \cdots x_{j'\cdot b^{\ell-j}}$. We define $\mathrm{C}_j(x) \triangleq (\mathrm{C}_0(x_{j,1}), \mathrm{C}_0(x_{j,2}), ..., \mathrm{C}_0(x_{j,b^j}))$, and $\pi_j(x) = (\pi_{j,1}(x), ..., \pi_{j,b^j}(x))$, where $p_{j,j'}(x)$ is a PCPP proof-oracle that asserts the consistency of j'-th block of $\mathrm{C}_j(x)$ and the $\lceil j'/b \rceil$-th block of $\mathrm{C}_{j-1}(x)$. That is, $p_{j,j'}(x)$ refers to an input oracle of the form (z_1, z_2), where $|z_1| = |z_2| = O(b^{\ell-j+1})$, and asserts the existence of x such that $z_1 = \mathrm{C}_0(x_{j,j'})^b$ and $z_2 = \mathrm{C}_0(x_{j-1,\lceil j'/b \rceil})$. We consider the following code*

$$\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^{t_0}, \mathrm{C}_1(x)^{t_0}, ..., \mathrm{C}_\ell(x)^{t_0}, \pi_1^{t_1}, ..., \pi_\ell^{t_\ell})$$

where the t_j's are selected such that each of the $2\ell+1$ parts of $\mathrm{C}(x)$ has the same length. The decoder, denoted D, operates as follows. On input $i \in [k]$ and oracle access to $w = (w_0, w_1, ..., w_\ell, v_1, ..., v_\ell)$, where $|w_0| = |w_j| = |v_j|$ for all j:

- *D selects uniformly $r_0, r_1, ..., r_\ell \in [t_0]$, and $(r'_1, r'_2, ..., r'_\ell) \in [t_1] \times [t_2] \times \cdots \times [t_\ell]$.*
- *For $j = 1, ..., \ell$, the decoder invokes the PCPP-verifier providing it with access to an input-oracle (z_1, z_2) and a proof oracle π that are defined as follows:*
  - *$z_1 = u^b$, where u is the $((r_j - 1)\cdot b^j + \lceil i/b^{\ell-j} \rceil)$-th block of w_j.*
  - *z_2 is the $((r_{j-1} - 1)\cdot b^{j-1} + \lceil i/b^{\ell-j+1} \rceil)$-th block of w_{j-1}.*
  - *π is the $((r'_j - 1)\cdot b^j + \lceil i/b^{\ell-j} \rceil)$-th block of v_j.*

  The PCPP-verifier is invoked with proximity parameter $\delta_{\mathrm{pcpp}} = 13\ell\delta > 0$, where $\delta \le \delta_0/81\ell$ is the proximity parameter sought for the decoder.
- *If the PCPP-verifier rejects, in any of the aforementioned ℓ invocations, then the decoder outputs a special* (failure) *symbol. Otherwise, the decoder outputs a random value in the $((r_\ell - 1)\cdot k + i)$-th block of w_ℓ* (which is supposedly a repetition code of x_i).
- *In order to make D satisfy the average smoothness property, we issue some dummy queries that are uniformly distributed in adequate parts of w that are queried less by the above.*

[5]Indeed, this was also done in the simpler code analyzed in Proposition 13.2.1.

[6]Recall that all these conditions hold for the PCP of proximity of Theorem 10.2.1, where almost-uniformly distributed queries to the proof-oracle are obtained by a modification analogous to the proof of Lemma 13.1.5.

(Suppose that V makes q_1 (resp., q_2) queries to the first (resp., second) part of it input-oracle and q' queries to its proof oracle. Then, w_0 is accessed q_2 times, w_ℓ is accessed q_1 times, each other w_j is accessed q_1+q_2 times, and each v_j is accessed q' times. Thus, we may add dummy queries to make each part accessed $\max(q_1+q_2,q')$ times, which means increasing the number of queries by a factor of at most $(2\ell+1)/(\ell-1)$ assuming $\ell \geq 2$.)

Using an adequate PCP of proximity, it holds that $|\mathrm{C}(x)| = \ell\cdot(|x|^{1+(1/\ell)})^{1+o(1)} < |x|^{1+\varepsilon}$, for $\varepsilon = 2/\ell$. The query complexity of D is $O(\ell)\cdot O(1/\delta_{\mathrm{pcpp}}) = O(\ell^2)$. The proof of Proposition 13.2.1 can be extended, obtaining the following:

Proposition 13.2.3 *The code and decoder of Construction 13.2.2 satisfy Conditions 1 and 2 of Definition 13.1.1 with respect to proximity parameter $\delta \leq \delta_0/81\ell$. Furthermore, this decoder satisfies the average smoothness property.*

Using Lemma 13.1.6, Theorem 1.3.5 follows.

Proof: Again, Condition 1 as well as the average smoothness property are obvious from the construction, and we focus on establishing Condition 2. Thus, we fix an arbitrary $i \in [k]$ and follow the outline of the proof of Proposition 13.2.1.

We consider an oracle $(w_0, w_1, ..., w_\ell, \pi_1, ..., \pi_\ell)$ that is δ-close to an encoding of $x \in \{0,1\}^k$, where each w_j is supposed to consist of encodings of the $k^{j/\ell}$ (non-overlapping) $k^{1-(j/\ell)}$-bit long blocks of x, and π_i consists of the corresponding proofs of consistency. It follows that each w_j is $(2\ell+1)\cdot\delta$-close to $\mathrm{C}_j(x)^{t_0}$. Let Z_j denote the block of w_j that was selected and accessed by D. Thus, the expected relative distance of Z_0 from $\mathrm{C}_0(x)$ is at most $(2\ell+1)\cdot\delta$, but we do not know the same about the other Z_j's because their choice depends on i (or rather on $\lceil i/b^{\ell-j}\rceil$). Assuming, without loss of generality, that $\delta_0 < 1/3$ (and $\ell \geq 1$), we consider three cases:

Case 1: Z_ℓ is $1/9$-close to $\mathrm{C}_0(x_i)$. In this case, D outputs either $\perp$ or a uniformly selected bit in Z_ℓ, and so D outputs $\neg x_i$ with probability at most $1/9$.

Using $\delta \leq \delta_0/81\ell$ and $\delta_0 < 1/3$, it follows that $27\ell\delta < 1/9$. Thus, if Case 1 does not hold then Z_ℓ is $27\ell\delta$-far from $\mathrm{C}_0(x_i)$.

Case 2: Z_0 is $27\ell\delta$-far from $\mathrm{C}_0(x)$. This case may occur with probability at most $1/9$, because $\mathrm{E}[\overline{\Delta}(Z_0, \mathrm{C}_0(x))] \leq 3\ell\delta\cdot|\mathrm{C}_0(x)|$.

Note that if both Cases 1 and 2 do not hold then Z_0 is $27\ell\delta$-close to $\mathrm{C}_0(x)$ but Z_ℓ is $27\ell\delta$-far from $\mathrm{C}_0(x_i)$. Also note that $x = x_{0,1}$ and $x_i = x_{\ell,i}$.

Case 3: For some $j \in [\ell]$, it holds that Z_{j-1} is $27\ell\delta$-close to $\mathrm{C}_{j-1}(x_{j-1,\lceil i/b^{\ell-j+1}\rceil})$ but Z_j is $27\ell\delta$-far from $\mathrm{C}_j(x_{j,\lceil i/b^{\ell-j}\rceil})$. In this case, the pair (Z_j^b, Z_{j-1}) is $27\ell\delta/2$-far from the consistent pair

$(\mathrm{C}_j(x_{j,\lceil i/b^{\ell-j}\rceil}), \mathrm{C}_{j-1}(x_{j-1,\lceil i/b^{\ell-j+1}\rceil}))$ and is $(\delta_0 - 27\ell\delta)/2$-far from any other consistent pair. Using $\delta_{\rm pcpp} = 13\ell\delta < \min(27\ell\delta/2, \delta_0 - 27\ell\delta)/2)$, which holds because $\delta \le \delta_0/81\ell$, it follows that in the current case the PCPP-verifier accepts (and the decoder does not output $\perp$) with probability at most $1/9$.

Thus, in total, the decoder outputs $\neg x_i$ with probability at most $1/3$. ■

Conclusion (Restating Theorem 1.3.5): *For every constant $\varepsilon > 0$, there exists a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, where $n = k^{1+\varepsilon}$, that is relaxed locally decodable under Definition 13.1.7. The query complexity of the corresponding decoder is $O(1/\varepsilon^2)$ and the proximity parameter is $\varepsilon/O(1)$.*

Open Problem: We wonder whether one can obtain relaxed-LDC that can be decoded using q queries while having length $n = o(k^{1+1/(q-1)})$. The existence of such relaxed-LDC will imply that our relaxation (i.e., relaxed-LDC) is actually strict, because such codes will beat the lower-bound currently known for LDC (cf. [KT00]). Alternatively, it may be possible to improve the lower-bound for (q-query) LDC to $n > k^{1+\sqrt{c/q}}$, for any constant c and every sufficiently large constant q (where, as usual, k is a parameter whereas q is a fixed constant). (In fact, some conjecture that n must be super-polynomial in k, for any constant q.)

13.3 Linearity of the codes

We note that the codes presented above (establishing both Theorems 1.3.4 and 1.3.5) are actually GF(2)-linear codes, whenever the base code C_0 is also GF(2)-linear. Proving this assertion reduces to proving that the PCPs of proximity used (in the aforementioned constructions) have proof-oracles in which each bit is a linear functions of the bits to which the proof refers. The main part of the latter task is undertaken in Section 9.5, where we show the the main construct (i.e., the PCPs of proximity stated in Theorems 6.1.1 and 5.1.2) when applied to a linear circuit yields a an GF(2)-linear transformation of assignments (satisfying the circuit) to proof-oracles (accepted by the verifier). In addition, we need to show that also the construction underlying the proof of Theorem 10.2.1 satisfy this property. This is done next, and consequently we get:

Proposition 13.3.1 *If C is a linear circuit* (see Definition 9.5.1), *then there is a linear transformation T mapping satisfying assignments w of C to proof oracles $T(w)$ such that the PCPP verifier of Theorem 10.2.1 will, on input C, accept oracle $(w, T(w))$ with probability 1.*

Proof Sketch: In Section 9.5, we establish a corresponding result for the main construct (i.e., Proposition 9.5.2 refers to the linearity of the construction used in the proof of Theorem 9.1.1, which in turn underlies Theorems 6.1.1 and 5.1.2). Here we show that linearity is preserved in composition as well as by the most inner (or bottom) verifier.

In each composition step, we append the proof-oracle with new (inner) PCPs of proximity per each test of the (outer) verifier. Since all these tests are linear, we can apply Proposition 9.5.2 and infer that the new appended information is a linear transformation of the input-oracle and the outer proof-oracle (where, by induction, the latter is a linear transformation of the input).

At the bottom level of composition we apply a Hadamard based PCP (Chapter 4). The encoding defined there is not $GF(2)$-linear (rather it is quadratic), but this was necessary for dealing with non-linear gates. It can be verified that for a linear circuit, one can perform all necessary tests of Chapter 4 with the Hadamard encoding of the input. Thus, we conclude this final phase of the encoding is also linear, and this completes the proof of Proposition 13.3.1. ■

APPENDIX A

Low Degree Test

Low-degree tests have been a subject of much research in the context of program checking and PCPs. Most constructions of PCPs involve checking whether a given function is a low-degree polynomial as an intermediate step. Low-degree tests are procedures designed to address this verification step ,i.e., to verify that an arbitrary function $f : \mathrm{F}^m \rightarrow \mathrm{F}$ is close to some (unknown) polynomial p of degree d. We are especially interested in randomness-efficient low-degree tests. For the sake of completeness, we recall the discussion on the low-degree test of [ALM$^+$98] from Chapter 5, explain why it performs poorly with respect to randomness and finally present the randomness-efficient low-degree test of [BSVW03].

A.1 The Line-Point Test

A line in F^m is a collection of points parametrized by one variable. Specifically, given $a, b \in \mathrm{F}^m$ the line $l_{a,b} = \{l_{a,b}(t) = a + tb | t \in \mathrm{F}\}$. Several parameterizations are possible for a given line. We assume some canonical one is fixed for every line, and thus the line is equivalent to the set of points it contains. The low-degree test uses the fact that for any polynomial $p : \mathrm{F}^m \rightarrow \mathrm{F}$ of degree at most d, the function $p_l : \mathrm{F} \rightarrow \mathrm{F}$ given by $p_l(t) = p(l(t))$ is a univariate polynomial of degree at most d. The verifier tests this property for a function f by picking a random line through F^m and verifying that there *exists* a univariate polynomial that has good agreement with f restricted to this line. The verifier expects an auxiliary oracle $f_{\mathbb{L}}$ that gives such a univariate polynomial for every line. In other words, $f_{\mathbb{L}} : \mathbb{L} \rightarrow P_d$ where $\mathbb{L}$ is the set of all lines in F^m and P_d the set of all univariate polynomial of degree at most d.. This motivates the test below.

LINE–POINT–TEST

Input: A function $f : \mathrm{F}^m \to \mathrm{F}$ and an oracle $f_{\mathbb{L}} : \mathbb{L} \to P_d$.

1. Choose a random point in the space $x \in_R \mathrm{F}^m$.
2. Choose a random line l passing through x in F^m.
3. Query $f_{\mathbb{L}}$ on l to obtain the polynomial h_l. Query f on x.
4. Accept iff the value of the polynomial h_l at x agrees with $f(x)$.

It is clear that if f is a degree d polynomial, then there exists an oracle $f_{\mathbb{L}}$ such that the above test accepts with probability 1. It is non-trivial to prove any converse and Arora *et al.* [ALM$^+$98] give a strikingly strong converse.

Theorem 5.3.2 ([ALM$^+$98], Theorem 65 - restated): *There exists a universal constants $0 < \delta_0 < 1$ and $\alpha > 0$ such that the following holds. For all integers $m, d > 0$, $\delta < \delta_0$ and fields F of size at least αd^3, if $f : F^m \to F$ and $f_{\mathbb{L}} : \mathbb{L} \to P_d$ are two functions that f is at least 2δ-far from, any m-variate polynomial of degree at most d, we have the following:*

$$\Pr[\textsc{Line–Point–Test}^{f;\, f_{\mathbb{L}}} = \text{reject}] > \delta.$$

This low degree test suffices to prove the PCP Theorem. However, for the purpose of constructing short PCPs, this test is too expensive in terms of randomness for the following two reasons.

- The field size is cubic in terms of the total degree of the field. This causes a cubic blow-up in the size of the proof.
- Choosing a random line in F^m requires choosing two random points in the space F^m and this costs randomness at least $2m \log |F|$. This results in a further quadratic blowup in the proof size.

However, luckily for us, they have been several improvements in the analysis of the low-degree test since [ALM$^+$98]. We will be using the derandomized low-degree test due to Ben-Sasson *et al.* [BSVW03] which solves both the above problems: the field size needs to be only linear in the total degree of the polynomial and further more, their analysis works even when the lines are chosen from a derandomized set. In the following section, we cite the main results of the work which we would require for our PCP constructions.

A.2 *Randomness-efficient low-degree tests and the sampling lemma*

A.2.1 *λ-biased sets and the sampling lemma*

Following [BSVW03], our construction makes heavy use of small-bias spaces [NN90] to save on randomness when choosing random lines. For a field F and parameters $m \in \mathbb{Z}^+$ and $\lambda > 0$, we require a set $S \subseteq F^m$ that is *λ-biased* (with respect to the additive group of F^m). Rather than define small-bias spaces here, we simply state the properties we need. (See, e.g., [BSVW03] for definitions and background on small-bias spaces.)

Lemma A.2.1 ([AGHP92]) *For every F of characteristic 2, $m \in \mathbb{Z}^+$, and $\lambda > 0$, there is an explicit construction of a λ-biased set $S \subseteq F^m$ of size at most $(\log |F^m|)/\lambda^2$.*

We now discuss the properties of such sets that we will use.

Expanding Cayley Graphs. λ-biased sets are very useful pseudorandom sets in algebraic applications, and this is due in part to the expansion properties of the Cayley graphs they generate:

Lemma A.2.2 *If $S \subseteq F^m$ is λ-biased and we let G_S be the graph with vertex set F^m and edge set $\{(x, x + s) : x \in F^m, s \in S\}$, then all the nontrivial eigenvalues of G_S have absolute value at most $\lambda|S|$.*

Randomness-Efficient Line Samplers. In [BSVW03], Lemma A.2.2 was used to prove the following sampling lemma. This lemma says that if one wants to estimate the density of a set $B \subseteq F^m$ using lines in F^m as the sample sets, one does not need to pick a random line in F^m which costs $2 \log |F^m|$ random bits. A pseudorandom line whose slope comes from an λ-biased set will do nearly as well, and the randomness is only $(1 + o(1)) \cdot \log |F^m|$. In what follows $l_{x,y}$ is the line passing through point x in direction y, formally: $l_{x,y} = \{x + ty : t \in F\}$

Lemma A.2.3 ([BSVW03], Sampling Lemma 4.3) *Suppose $S \subseteq F^m$ is λ-biased. Then, for any $B \subseteq F^m$ of density $\mu = |B|/|F^m|$, and any $\zeta > 0$,*

$$\Pr_{x \in F^m, y \in S}\left[\left|\frac{|l_{x,y} \cap B|}{|l_{x,y}|} - \mu\right| > \zeta\right] \le \left(\frac{1}{|F|} + \lambda\right) \cdot \frac{\mu}{\zeta^2}.$$

A.2.2 Randomness-Efficient Low Degree Tests

Ben-Sasson *et al.* [BSVW03] use the randomness-efficient Sampling Lemma A.2.3 to obtain randomness efficient low degree tests, by performing a "line vs. point" test only for pseudorandom lines with a direction y coming from a small λ-biased set. That is for a set $S \subseteq F^m$, we consider lines of the form $l_{x,y}(t) = x + ty$, for $x \in F^m$ and $y \in S$, and let $\mathbb{L}_S$ be the set of all such lines, where each line is parametrized in a canonical way.

Then for functions $f : F^m \to F$, and $f_{\mathbb{L}_S} : \mathbb{L}_S \to P_d$, where P_d is the set of univariate polynomials of degree at most d over F, we let $\text{LINE–POINT–TEST}_S^{f, f_{\mathbb{L}_S}}$ be the following modified line-point low degree test:

LINE–POINT–TEST_S

Input: A function $f : \mathrm{F}^m \to \mathrm{F}$ and an oracle $f_{\mathbb{L}_S} : \mathbb{L}_S \to P_d$.

1. Choose a random point in the space $x \in_R \mathrm{F}^m$.
2. Choose a random point $y \in S$ and let l be the line $\{x + ty | t \in F\}$.
3. Query $f_{\mathbb{L}_S}$ on l to obtain the polynomial h_l. Query f on x.
4. Accept iff the value of the polynomial h_l at x agrees with $f(x)$.

We quote the main theorem of [BSVW03] and will use it in our constructions.

Theorem A.2.4 ([BSVW03], Theorem 4.1) *There exists a universal constant $\alpha > 0$ such that the following holds. Let $d \leq |F|/3, m \leq \alpha|F|/\log|F|, S \subseteq F^m$ be a λ-biased set for $\lambda \leq \alpha/(m \log|F|)$, and $\delta \leq \alpha$. Then, for every $f : F^m \to F$ and $g : \mathbb{L} \to P_d$ such that f is at least 4δ-far from, any polynomial of degree at most md, we have the following:*

$$\Pr[\textsc{Line–Point–Test}^{f;\,g}_{S,d} = \mathrm{rej}] > \delta.$$

Bibliography

[AGHP92] Alon, N., Goldreich, O., Håstad, J., and Peralta, R. Simple constructions of almost $k-$wise independent random variables. *Journal of Random Structures and Algorithms 3*, 3 (Fall 1992), 289–304.

[Aro95] Arora, S. Reductions, codes, PCPs, and inapproximability. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science* (Milwaukee, Wisconsin, 23–25 Oct. 1995), pp. 404–413.

[ALM$^+$98] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. Proof verification and the hardness of approximation problems. *Journal of the ACM 45*, 3 (May 1998), 501–555. (Preliminary Version in *33rd FOCS*, 1992).

[AS98] Arora, S., and Safra, S. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM 45*, 1 (Jan. 1998), 70–122. (Preliminary Version in *33rd FOCS*, 1992).

[AS97] Arora, S., and Sudan, M. Improved low degree testing and its applications. In *Proc. 29th ACM Symp. on Theory of Computing* (El Paso, Texas, 4–6 May 1997), pp. 485–495.

[Bab85] Babai, L. Trading group theory for randomness. In *Proc. 17th ACM Symp. on Theory of Computing* (Providence, Rhode Island, 6–8 May 1985), pp. 421–429.

[BFLS91] Babai, L., Fortnow, L., Levin, L. A., and Szegedy, M. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing* (New Orleans, Louisiana, 6–8 May 1991), pp. 21–31.

[BFL91] Babai, L., Fortnow, L., and Lund, C. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity 1* (1991), 3–40. (Preliminary Version in *31st FOCS*, 1990).

[Bar01] Barak, B. How to go beyond the black-box simulation barrier. In *Proc. 42nd IEEE Symp. on Foundations of Comp. Science* (Las Vegas, Nevada, 14–17 Oct. 2001), pp. 106–115.

[BF90] BEAVER, D., AND FEIGENBAUM, J. Hiding instances in multioracle queries. In *Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS)* (Rouen, France, 22–24 Feb. 1990), C. Choffrut and T. Lengauer, Eds., vol. 415 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 37–48.

[BIKR02] BEIMEL, A., ISHAI, Y., KUSHILEVITZ, E., AND RAYMOND, J. F. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science* (Vancouver, British Columbia, Canada, 16–19 Nov. 2002), pp. 261–270.

[BGS98] BELLARE, M., GOLDREICH, O., AND SUDAN, M. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal of Computing 27*, 3 (June 1998), 804–915. (Preliminary Version in *36th FOCS*, 1995).

[BGLR93] BELLARE, M., GOLDWASSER, S., LUND, C., AND RUSSELL, A. Efficient probabilistically checkable proofs and applications to approximation. In *Proc. 25th ACM Symp. on Theory of Computing* (San Diego, California, 16–18 May 1993), pp. 294–304.

[BGKW88] BEN-OR, M., GOLDWASSER, S., KILIAN, J., AND WIGDERSON, A. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proc. 20th ACM Symp. on Theory of Computing* (White Plains, New York, 24–26 Oct. 1988), pp. 113–131.

[BGH$^+$04a] BEN-SASSON, E., GOLDREICH, O., HARSHA, P., SUDAN, M., AND VADHAN, S. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. In *Proc. 36th ACM Symp. on Theory of Computing* (Chicago, Illinois, 13–15 June 2004), pp. 1–10.

[BGH$^+$04b] BEN-SASSON, E., GOLDREICH, O., HARSHA, P., SUDAN, M., AND VADHAN, S. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. Tech. Rep. TR04-021, Electronic Colloquium on Computational Complexity, March 2004.

[BHR03] BEN-SASSON, E., HARSHA, P., AND RASKHODNIKOVA, S. Some 3CNF properties are hard to test. In *Proc. 35th ACM Symp. on Theory of Computing* (San Diego, California, 9–11 June 2003), pp. 345–354.

[BS04] BEN-SASON, E., AND SUDAN, M. Simple PCPs with poly-log rate and query complexity. Tech. Rep. TR04-060, Electronic Colloquium on Computational Complexity, 2004.

[BSVW03] BEN-SASSON, E., SUDAN, M., VADHAN, S., AND WIGDERSON, A. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proc. 35th ACM Symp. on Theory of Computing* (San Diego, California, 9–11 June 2003), pp. 612–621.

[BLR93] BLUM, M., LUBY, M., AND RUBINFELD, R. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences 47*, 3 (Dec. 1993), 549–595. (Preliminary Version in *22nd STOC*, 1990).

[BOT02] BOGDANOV, A., OBATA, K., AND TREVISAN, L. A lower bound for testing 3-colorability in bounded-degree graphs. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science* (Vancouver, Canada, 16–19 Nov. 2002), pp. 93–102.

[BT04] BOGDANOV, A., AND TREVISAN, L. Lower bounds for testing bipartiteness in dense graphs. In *Proc. 19th IEEE Conference on Computational Complexity* (Amherst, Massachusetts, 21–24 June 2004), pp. 75–81.

[BdW04] BUHRMAN, H., AND DE WOLF, R. On relaxed locally decodable codes. Unpublished manuscript, July 2004.

[CGH98] CANETTI, R., GOLDREICH, O., AND HALEVI, S. The random oracle methodology, revisited. In *Proc. 30th ACM Symp. on Theory of Computing* (Dallas, Texas, 23–26 May 1998), pp. 209–218.

[CGKS98] CHOR, B., GOLDREICH, O., KUSHILEVITZ, E., AND SUDAN, M. Private information retrieval. *Journal of the ACM 45*, 6 (Nov. 1998), 965–981. (Preliminary Version in *36th FOCS*, 1995).

[Coo88] COOK, S. A. Short propositional formulas represent nondeterministic computations. *Information Processing Letters 26*, 5 (Jan. 1988), 269–270.

[DJK$^+$02] DESHPANDE, A., JAIN, R., KAVITHA, T., RADHAKRISHNAN, J., AND LOKAM, S. V. Better lower bounds for locally decodable codes. In *Proc. 17th IEEE Conference on Computational Complexity* (Montréal, Québec, Canada, 21–24 May 2002), pp. 184–193.

[DR04] DINUR, I., AND REINGOLD, O. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. To appear in *Proc. 45rd IEEE Symp. on Foundations of Comp. Science* (Rome, Italy, 17–19 Oct. 2004).

[EKR99] ERGÜN, F., KUMAR, R., AND RUBINFELD, R. Fast approximate PCPs. In *Proc. 31st ACM Symp. on Theory of Computing* (Atlanta, Georgia, 1–4 May 1999), pp. 41–50.

[Fei98] FEIGE, U. A threshold of ln$n$ for approximating set cover. *Journal of the ACM 45*, 4 (July 1998), 634–652. (Preliminary Version in *28th STOC*, 1996).

[FGL$^+$96] FEIGE, U., GOLDWASSER, S., LOVÁSZ, L., SAFRA, S., AND SZEGEDY, M. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM 43*, 2 (Mar. 1996), 268–292. (Preliminary version in *32nd FOCS*, 1991).

[FRS94] FORTNOW, L., ROMPEL, J., AND SIPSER, M. On the power of multi-prover interactive protocols. *Theoretical Computer Science 134*, 2 (Nov. 1994), 545–557. (Preliminary Version in *3rd IEEE Symp. on Structural Complexity*, 1988).

[FS95] FRIEDL, K., AND SUDAN, M. Some improvements to total degree tests. In *Proc. 3rd Israel Symposium on Theoretical and Computing Systems* (Tel Aviv, Israel, 4–6 Jan. 1995), pp. 190–198.

[Gol97] GOLDREICH, O. A sample of samplers – a computational perspective on sampling. Tech. Rep. TR97-020, Electronic Colloquium on Computational Complexity, 1997.

[GGR98] GOLDREICH, O., GOLDWASSER, S., AND RON, D. Property testing and its connection to learning and approximation. *Journal of the ACM 45*, 4 (July 1998), 653–750. (Preliminary Version in *37th FOCS*, 1996).

[GL89] GOLDREICH, O., AND LEVIN, L. A. A hard-core predicate for all one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing* (Seattle, Washington, 15–17 May 1989), pp. 25–32.

[GMW91] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM 38*, 3 (July 1991), 691–729. (Preliminary Version in *27th FOCS*, 1986).

[GR02] GOLDREICH, O., AND RON, D. Property testing in bounded degree graphs. *Algorithmica 32*, 2 (Jan. 2002), 302–343. (Preliminary Version in *29th STOC*, 1997).

[GS00] GOLDREICH, O., AND SAFRA, S. A combinatorial consistency lemma with application to proving the PCP theorem. *SIAM Journal of Computing 29*, 4 (2000), 1132–1154. (Preliminary Version in *RANDOM*, 1997).

[GS02] GOLDREICH, O., AND SUDAN, M. Locally testable codes and PCPs of almost linear length. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science* (Vancouver, Canada, 16–19 Nov. 2002), pp. 13–22. (See ECCC Report TR02-050, 2002).

[GW97] GOLDREICH, O., AND WIGDERSON, A. Tiny families of functions with random properties: A quality–size trade–off for hashing. *Journal of Random structures and Algorithms 11*, 4 (Dec. 1997), 315–343. (Preliminary Version in *26th STOC*, 1994).

[GMR89] GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing 18*, 1 (Feb. 1989), 186–208. (Preliminary Version in *17th STOC*, 1985).

[GLST98] GURUSWAMI, V., LEWIN, D., SUDAN, M., AND TREVISAN, L. A tight characterization of NP with 3-query PCPs. In *Proc. 39th IEEE Symp. on Foundations of Comp. Science* (Palo Alto, California, 8–11 Nov. 1998), pp. 18–27.

[HS00] HARSHA, P., AND SUDAN, M. Small PCPs with low query complexity. *Computational Complexity 9*, 3–4 (Dec. 2000), 157–201. (Preliminary Version in *18th STACS*, 2001).

[Hås99] HÅSTAD, J. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica 182* (1999), 105–142. (Preliminary Version in *28th STOC*, 1996 and *37th FOCS*, 1997).

[Hås01] HÅSTAD, J. Some optimal inapproximability results. *Journal of the ACM 48*, 4 (July 2001), 798–859. (Preliminary Version in *29th STOC*, 1997).

[HS66] HENNIE, F. C., AND STEARNS, R. E. Two-tape simulation of multitape Turing machines. *Journal of the ACM 13*, 4 (Oct. 1966), 533–546.

[KT00] KATZ, J., AND TREVISAN, L. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd ACM Symp. on Theory of Computing* (Portland, Oregon, 21–23 May 2000), pp. 80–86.

[KdW03] KERENIDIS, I., AND DE WOLF, R. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *Proc. 35th ACM Symp. on Theory of Computing* (San Diego, California, 9–11 June 2003), pp. 106–115.

[Kho04] KHOT, S. Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. To appear in *Proc. 45rd IEEE Symp. on Foundations of Comp. Science* (Rome, Italy, 17–19 Oct. 2004).

[Kil92] KILIAN, J. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th ACM Symp. on Theory of Computing* (Victoria, British Columbia, Canada, 4–6 May 1992), pp. 723–732.

[LS91] LAPIDOT, D., AND SHAMIR, A. Fully parallelized multi prover protocols for NEXP-time (extended abstract). In *Proc. 32nd IEEE Symp. on Foundations of Comp. Science* (San Juan, Puerto Rico, 1–4 Oct. 1991), pp. 13–18.

[Lei92] LEIGHTON, F. T. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[Lip91] LIPTON, R. J. New directions in testing. In *Proc. DIMACS Workshop on Distributed Computing and Cryptography* (Providence, Rhode Island, 1991), vol. 2 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 191–202.

[LFKN92] LUND, C., FORTNOW, L., KARLOFF, H. J., AND NISAN, N. Algebraic methods for interactive proof systems. *Journal of the ACM 39*, 4 (Oct. 1992), 859–868. (Preliminary Version in *31st FOCS*, 1990).

[LY94] LUND, C., AND YANNAKAKIS, M. On the hardness of approximating minimization problems. *Journal of the ACM 41*, 5 (Sept. 1994), 960–981.

[Mic00] MICALI, S. Computationally sound proofs. *SIAM Journal of Computing 30*, 4 (2000), 1253–1298. (Preliminary Version in *35th FOCS*, 1994).

[NN90] NAOR, J., AND NAOR, M. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd ACM Symp. on Theory of Computing* (Baltimore, Maryland, 4–16 May 1990), pp. 213–223.

[PY91] PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences 43*, 3 (Dec. 1991), 425–440. (Preliminary Version in *20th STOC*, 1988).

[PF79] PIPPENGER, N., AND FISCHER, M. J. Relations among complexity measures. *Journal of the ACM 26*, 2 (Apr. 1979), 361–381.

[PS94] POLISHCHUK, A., AND SPIELMAN, D. A. Nearly-linear size holographic proofs. In *Proc. 26th ACM Symp. on Theory of Computing* (Montréal, Québec, Canada, 23–25 May 1994), pp. 194–203.

[Raz98] RAZ, R. A parallel repetition theorem. *SIAM Journal of Computing 27*, 3 (June 1998), 763–803. (Preliminary Version in *27th STOC*, 1995).

[RS97] RAZ, R., AND SAFRA, S. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing* (El Paso, Texas, 4–6 May 1997), pp. 475–484.

[RS96] RUBINFELD, R., AND SUDAN, M. Robust characterizations of polynomials with applications to program testing. *SIAM Journal of Computing 25*, 2 (Apr. 1996), 252–271. (Preliminary Version in *23rd STOC*, 1991 and *3rd SODA*, 1992).

[ST00] SAMORODNITSKY, A., AND TREVISAN, L. A PCP characterization of NP with optimal amortized query complexity. In *Proc. 32nd ACM Symp. on Theory of Computing* (Portland, Oregon, 21–23 May 2000), pp. 191–199.

[Sch77] SCHÖNHAGE, A. Schnelle multiplikation von polynomen über Körpern der charakteristik 2 (German). *Acta Informatica 7*, 4 (1977), 395–398.

[SS71] SCHÖNHAGE, A., AND STRASSEN, V. Schnelle multiplikation großer zahlen (German). *Computing 7*, 3–4 (1971), 281–292.

[Sch80] SCHWARTZ, J. T. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM 27*, 4 (Oct. 1980), 701–717.

[Sha92] SHAMIR, A. IP = PSPACE. *Journal of the ACM 39*, 4 (Oct. 1992), 869–877.

[Spi95] SPIELMAN, D. A. *Computationally Efficient Error-Correcting Codes and Holographic Proofs*. PhD thesis, Massachusetts Institute of Technology, June 1995.

[Spi96] SPIELMAN, D. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory 42*, 6 (Nov. 1996), 1723–1732. (Preliminary Version in *27th STOC*, 1995).

[Str73] STRASSEN, V. Vermeidung von Divisionen (German). *J. Reine Angew. Math.*, 264 (1973), 184–202.

[STV01] SUDAN, M., TREVISAN, L., AND VADHAN, S. P. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences 62*, 2 (Mar. 2001), 236–266. (Preliminary Version in 31*st STOC*, 1999).

[Sze99] SZEGEDY, M. Many-valued logics and holographic proofs. In *Proc.* 26*th International Colloquium of Automata, Languages and Programming (ICALP '99)* (Prague, Czech Republic, 11–15 July 1999), J. Wiedermann, P. van Emde Boas, and M. Nielsen, Eds., vol. 1644 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 676–686.

[Tre04] TREVISAN, L. Some applications of coding theory in computational complexity. Tech. Rep. TR04-043, Electronic Colloquium on Computational Complexity, 2004.

[Zip79] ZIPPEL, R. Probablistic algorithms for sparse polynomials. In *Proc. International Symposium of Symbolic and Algebraic Computation (EUROSAM '79)* (Marseille, France, June 1979), Edward W. Ng, Ed., vol. 72 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 216–226.