מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

Thesis for the degree
Doctor of Philosophy

עבודת גמר (תזה) לתואר
דוקטור לפילוסופיה

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
Or Shalom Meir

מאת
אור שלום מאיר

בניות קומבינטוריות של מערכות הוכחה הסתברותיות
Combinatorial Constructions of Probabilistic Proof Systems

Advisor: Prof. Oded Goldreich

מנחה: פרופ' עודד גולדרייך

June 2010

סיוון ה'תשע"א

Abstract

Probabilistic proof systems is a paradigm of complexity theory whose study evolves around questions such as "how can we use randomness to prove and verify assertions?", "what do we gain from using randomness in verification procedures?", and "what assertions can be verified by probabilistic verification procedures?". The study of those questions has began in the 1980's, and led to several of the most important achievements of complexity theory since then.

Many of the key results regarding probabilistic proof systems rely on sophisticated algebraic techniques. While those algebraic techniques are very important and useful, they seem to give little intuition as to why those results hold. Given this state of affairs, it is an important goal to gain a better understanding of those results and the reasons for which they hold. In her seminal paper, Dinur (J. ACM 54(3)) has made a big step toward achieving this goal by giving an alternative proof of one of the key results in this area, namely the PCP theorem, using a combinatorial approach. Her proof is not only considerably simpler than the original proof, but also seems to shed more light on the intuitions that underlie the theorem.

In this thesis, we pursue this direction further, by providing alternative proofs for several key results about probabilistic proof systems. Our alternative proofs do not use algebra (or use almost no algebra), and are more intuitive, in our opinion. In particular:

- We show that it is possible to prove that $\mathbf{IP} = \mathbf{PSPACE}$ using general error correcting codes and their tensor products, instead of low degree polynomials.
- We provide a combinatorial construction of PCPs with verifiers that are as efficient as those obtained by the algebraic methods.
- We provide an (almost) combinatorial construction of PCPs of length $n \cdot (\log n)^{O(\log \log n)}$, coming very close to the state of the art obtained by algebraic constructions (whose proof length is $n \cdot (\log n)^{O(1)}$).
- We provide a combinatorial construction of PCPs with sub-constant soundness error that match the state of the art obtained by algebraic constructions, and along the way develop a technique of derandomized parallel repetition.

Acknowledgement

It is my pleasure to express my deepest gratitude to Oded Goldreich, my advisor. I feel that Oded has taught me how to do research, and has influenced considerably on my perspective of theoretical computer science, and of what does it mean to be a scientist in general. I also deeply appreciate his kindness and his devotion to his students. Not less important, working with Oded has been a very fun experience, and I enjoyed and learned very much from our conversations, both on professional and non-professional subjects.

I would like to thank to Gillat Kol for being a great friend throughout my graduate studies, and making my time at the Weizmann institute the enjoyable experience it was. I would also like to thank Tal Kramer, Shira Kritchman, Inbal Talgam, Irit Dinur, Dana Moshkovitz, Zvika Brakersky, Chandan Dubey, Anat Ganor, Elazar Goldenberg, and Shachar Lovett for being such a fun and interesting company. As for Irit, I am also grateful to her for a much fun and educating collaboration, and of course, for her paper that led me to this thesis.

As always, I am grateful to my parents for their everlasting support, care and love.

This thesis is based on the following works:

- "**IP** = **PSPACE** using Error Correcting Codes", by the author [Mei10b].
- "Combinatorial PCPs with Efficient Verifiers", by the author [Mei09].
- "Combinatorial PCPs with Short Proofs" by the author [Mei10a].
- "Derandomized Parallel Repetition of Structured PCPs" by Irit Dinur and the author [DM10].

Contents

Chapter 1

Introduction

Mathematicians have used mathematical proofs in order to establish their claims for thousands of years. However, it was only in the late 19th century when mathematicians began to study the notion of a mathematical proof itself. This study has lead to questions such as "Can any valid mathematical claim be demonstrated by a mathematical proof?" and "Is there a systematic way for finding mathematical proofs?". The second question has lead to the birth of theoretical computer science, when Alan Turing has shown that there does not exist an algorithm for finding mathematical proofs. Since then, the notion of a proof has been of utmost importance to theoretical computer science, and in fact the central question of this field - often phrased as whether **P** is equal to **NP** - is a question about the relation between algorithms and proofs.

During the 1980's, computer scientists began to study models for proving claims other than mathematical proofs. Let us consider the situation in which an expert wishes to convince a skeptic in the validity of some claim. In the model of mathematical proofs, the expert would simply write the proof on a paper and hand it to the skeptic, and the skeptic would read the proof and verify that it indeed follows the rules of logic. Now, computer scientists raised the following questions: "What if we allowed the skeptic to ask the expert questions? What if we allowed the skeptic to toss coins during the verification of the proof? What if the skeptic would have been willing to risk accepting a false claim with an extremely small probability?". It turns out that different models of proofs, which allow randomness and interaction between the expert and the skeptic, allow for establishing a rich family of claims that can not be demonstrated in the classic model of a mathematical proof. For example, there can be no mathematical proof that would convince a color-blind skeptic that two cards are of a different color. But, if the color-blind skeptic could shuffle cards randomly and ask the expert to distinguish them, then the expert would have no problem convincing the skeptic that the cards are indeed distinguishable.

The study of such alternative models of proofs, known as "probabilistic proof systems", has resulted in a line of interesting and surprising discoveries regarding the power of such models [GMR89, Bab85, GMW91, LFKN92, Sha92, FGL$^+$96, AS98, ALM$^+$98] (see also [Gol08] for a primer on the subject). Those discoveries are not only interesting in their own right, but have also played a key role in many of the achievements of theoretical computer science in the last two decades, and have contributed to the understanding of many other subjects. For example, the study of "Probabilistically Checkable Proofs" (PCPs) has played a key role in understanding the inherent hardness of finding approximate solutions to computational problems, and the study of "zero knowledge proofs" has been essential to the theoretical study of cryptographic protocols.

A common theme that is shared by many of the key results about probabilistic proof systems is the use of algebraic techniques. Usually, such works construct a probabilistic proof system by going along the following lines: Given the claim to be verified, they begin with "arithmetizing" the claim, i.e., reducing the claim to a related "algebraic" claim about polynomials over finite fields. In the next step, they construct a probabilistic proof system for proving the algebraic claim. Constructing the proof system for the algebraic claim, in turn, relies on arsenal of tools that employ the algebraic structure of polynomials. While those algebraic techniques are very important and useful, it seems somewhat odd that one has to go through algebra in order to prove those theorems, which do not refer to algebra. Furthermore, those techniques seem to give little intuition as to why those results hold.

Given this state of affairs, it is an important goal to gain a better understanding of those results and the reasons for which they hold[1]. In her seminal paper, Dinur [Din07] has made a big step toward achieving this goal by giving an alternative proof of one of the key results in this area, namely the PCP theorem, using a combinatorial approach[2]. Her proof is not only considerably simpler than the original proof, but also seems to shed more light on the intuitions that underlie the theorem.

In this thesis, we pursue this direction further, by providing alternative proofs for several key results about probabilistic proof systems. Our alternative proofs do not use algebra (or use almost no algebra), and are more intuitive, in our opinion. We focus on two types of proof systems, namely, interactive proof systems and probabilistically checkable proofs. In the following two sections, we describe those types of proof systems, and along the way describe our results and the structure of this thesis.

1.1 Interactive Proof Systems

In the settings of interactive proofs [GMR89], a computationally unbounded prover (i.e., an expert) wishes to convince a polynomial-time verifier (i.e., a skeptic) of the validity of some claim. The verifier and the prover may toss coins and may interact with each other, but may only exchange a polynomial number of messages. The proof system must satisfy the following property: if the claim is correct, then the prover can always convince the verifier. On the other hand, if the claim is incorrect, then no matter what strategy the prover employs, the probability that it manages to fool the verifier to accepting the claim is small. Such systems are interesting in their own right, but also have several applications in theoretical computer science, and in particular they provide the framework for defining zero knowledge proofs, which are essential for theoretical cryptography [GMR89, GMW91].

A celebrated theorem of [LFKN92, Sha92] established that the class of claims that can be proved using such a proof system is exactly the class of claims that can be decided using polynomial space, or formally that $\mathbf{IP} = \mathbf{PSPACE}$. This theorem is fundamental to our understanding of both interactive proofs and polynomial space computations, and in addition has important applications in theoretical computer science.

While the original proof of the aforementioned theorem relies on low degree polynomials, it is commonly believed that the intuition for the proof should be explained in terms of error correcting codes. In Chapter 2, we provide evidence for this intuition by showing an alternative proof of the theorem that uses error correcting codes instead of low degree polynomials. This chapter was published separately as [Mei10b].

1.2 Probabilistic Proof Systems

In the setting of probabilistically checkable proofs (PCPs [BFLS91, FGL$^+$96]), we consider again a prover that wishes to convince a polynomial-time verifier of the validity of a claim. However, this time the interaction of prover and the verifier is extremely limited: the prover may only send to the verifier an alledged proof of the claim, and may not interact with the verifier any further. The verifier, in turn, may read only few bits of the alledged proof, although it may choose which bits to read. Again, we require

[1]This goal was stated and advocated for the first time by [GS00]

[2]We mention that the works of [GS00, DR06] have mades advances toward this goal prior to Dinur's work [Din07], but fell short of obtaining Dinur's result.

that if the claim is correct, then there is always a proof that will convince the verifier. On the other hand, if the claim is incorrect, then no matter what proof is provided to the verifier, the probability that the verifier will accept the claim is small.

At first glance, this proof system may seem very weak, and it may not be clear that it can prove any interesting claims. Surprisingly, the *PCP theorem* of [AS98, ALM$^+$98] asserts that any claim that can be decided within the complexity class **NP** (roughly, any claim that has a short mathematical proof), can also be proved in a proof system of the PCP type. This theorem is one of the major achievements of complexity theory. Besides of being interesting in its own right, the theorem has also found many applications, most notably in establishing limits on the accuarcy of approximation algorithms.

As discussed above, the original proof of the PCP theorem of [AS98, ALM$^+$98], as well as its extensions [BSGH$^+$06, BSS06, BSGH$^+$05, MR08, DH09], were based on algebraic machinery, while the later work of Dinur [Din07] has suggested a simpler and more intuitive proof of the PCP theorem, which was based on a combinatorial approach. However, Dinur's approach fell short of proving the later improvements of the theorem. The second part of this thesis is devoted to the goal of matching these improvements (originally obtained by algebraic techniques) using a combinatorial approach. Specifically, this thesis deals the following aspects:

- **Efficiency:** As mentioned above, the defining feature of PCPs is that they allow verifying the validity of the claim by reading only few bits of the proof. This means that the verification procedure may potentially be extremely efficient, and in particular may run in time that is much shorter than the proof length. Indeed, in state of the art algebraic PCPs [BSGH$^+$05], the verification procedure runs in time that is poly-logarithmic in the length of the proof. On the other hand, the work of [Din07] yields a much slower verification procedure, which runs in time that is polynomial in the length of the proof. We note that the running time of the verification procedure is not only interesting for its own sake: The difference between the efficiency of the verification procedures of [BSGH$^+$05] and [Din07] is crucial for some applications (in particular, instance checking [BFL91, BK95]) .
  In Chapter 3, we show a combinatorial construction of PCPs whose verification procedure runs in poly-logarithmic time, thus matching the state of the art algebraic PCPs. This chapter was published separately as [Mei09].

- **Length:** One important parameter of PCP systems is the length of the proofs they employ. More specifically, given a claim that can be proved both by a standard (mathematical) proof and by a probabilistically checkable proof (PCP), the question is how long should the PCP be compared to the standard proof? In the original PCP theorem [AS98, ALM$^+$98], if the standard proof is of length n, then the length of the corresponding PCP is some fixed polynomial in n. However, in subsequent improvements of the PCP theorem [BSS08, Din07][3], the PCP is of length only $n \cdot \log^c n$ (for some constant c). The combinatorial approach of [Din07] yields PCPs whose length matches the PCPs of [AS98, ALM$^+$98], but falls short of matching the shorter PCPs.
  In Chapter 4, we show how to construct PCPs of length $n \cdot (\log n)^{\log \log n}$ based on a combinatorial approach, thus almost matching the state of the art PCPs. It should be mentioned that our constuction does use algebra at one point, but this use is a very restricted one, and is confined to the construction of error correcting codes with a simple multiplication property. This chapter was published separately as [Mei10a].

- **Soundness error:** An additional important parameter of the PCP systems is their soundness error, which is the probability that the verifier accepts false claims. In the original PCP theorem of [AS98, ALM$^+$98] as well as in the work of [Din07], the soundness error is a constant independent

[3]Here we refer to a different part in the work of [Din07] than the parts discussed in the rest of this document.

of the claim length. However, subsequent improvements of the theorem [DFK^{+}99, MR08, DH09] have shown that the soundness can be pushed to be a decreasing function of the input length, and can be traded with the other parameters of the PCP.

In Chapter 5, we consider PCPs with low soundness error in a range of parameter that is especially important for applications of PCPs to lower bounds of approximation algorithms. The state of the art for such PCPs are the works of [MR08, DH09], which achieve their results by combining a folklore algebraic construction of PCPs with a novel combinatorial method. In Chapter 5, we show that the algebraic component in the constructions of [MR08, DH09] can be replaced with a combinatorial substitute. Our works thus yields a fully combinatorial construction of PCPs with low soundness error which matches the state of the art parameters. Along the way, we develop a technique of derandomized parallel repetition, which may be interesting in its own right.

This chapter is based on a joint work with Irit Dinur [DM10].

Chapter 2

IP = PSPACE using Error Correcting Codes

2.1 Introduction

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

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

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

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

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

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

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

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

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

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

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

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

[1]In other words, the polynomial generated by the arithmetization is not the low-degree extension of the truth table. To see this, note that the arithmetization of an unsatisfiable formula may produce a non-zero polynomial. For example, the arithmetization of the unsatisfiable formula $x \wedge \neg x$ is $x \cdot (1 - x)$, which is not the zero polynomial.

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

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

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

The organization of this chapter. In Section 2.2, we review the basic notions of error correcting codes and define the notation that we use. In Section 2.3, we review the notion of tensor product codes, and introduce the notion of multiplication codes. In Section 2.4, we revisit the sum-check protocol and generalize it. In Section 2.5, we prove that co**NP** $\subseteq$ **IP**, and along the way present our generalization of the arithmetization technique. Finally, in Section 2.6, we prove the full **IP** theorem.

Remark regarding algebrization. Recall that the **IP** theorem is a classical example for a non-relativizing result. Recently, [AW08] suggested a framework called "algebrization" as a generalization of the notion of relativization, and showed that the **IP** theorem relativizes in this framework, or in other words, the **IP** theorem "algebrizes". We note that while our proof of the **IP** theorem does not seem to algebrize, one can generalize the algebrization framework to include our proof as well. Some details are given in a remark at the end of Section 2.5.

2.2 Preliminaries

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

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

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

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

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

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

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

Fact 2.2.1. *The exists an algorithm that when given as input $k \in \mathbb{N}$ and $\delta \in (0,1)$ and a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq \text{poly}(1/(1-\delta))$, runs in time that is polynomial in k, $\log|\mathbb{F}|$, and $1/(1-\delta)$, and outputs the generator matrix of a linear code C over $\mathbb{F}$ that has message length k, block length $\ell \overset{\text{def}}{=} k/\text{poly}(1-\delta)$, and relative distance at least δ.*

Fact 2.2.1 can be proved via a variety of techniques from coding theory, where many of them do not use polynomials (see, e.g., [Var57, ABN$^+$92, GI05][2]).

2.3 Tensor Product Codes and Multiplication Codes

In this section we review the notion of tensor product of codes (in Section 2.3.1) and introduce the notion of multiplication codes (in Section 2.3.2). We note that while the tensor product is a standard operation in coding theory, and a reader who is familiar with it may skip Section 2.3.1, with the exception of Propositions 2.3.7 and 2.3.8 which are non-standard. On the other hand, the notion of multiplication codes is a non-standard notion that we define for this work (though it may be seen as a variant of the notion of error correcting pairs, see [Köt92, Pel92, Sud01, Lect. 11 (1.4)]).

2.3.1 Tensor Product of Codes

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

Definition 2.3.1. Let $R : \mathbb{F}^{k_R} \to \mathbb{F}^{\ell_R}$, $C : \mathbb{F}^{k_C} \to \mathbb{F}^{\ell_C}$ be codes. The tensor product code $R \otimes C$ is a code of message length $k_R \cdot k_C$ and block length $\ell_R \cdot \ell_C$ that encodes a message $x \in \mathbb{F}^{k_R \cdot k_C}$ as follows: In order to encode x, we first view x as a $k_C \times k_R$ matrix, and encode each of its rows via the code R, resulting in a $k_C \cdot \ell_R$ matrix x'. Then, we encode each of th columns of x' via the code C. The resulting $\ell_C \times \ell_R$ matrix is defined to be the encoding of x via $R \otimes C$.

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

Fact 2.3.2. *Let $R : \mathbb{F}^{k_R} \to \mathbb{F}^{\ell_R}$, $C : \mathbb{F}^{k_C} \to \mathbb{F}^{\ell_C}$ be linear codes. We have the following:*

1. *An $\ell_C \times \ell_R$ matrix x over $\mathbb{F}$ is a codeword of $R \otimes C$ if and only if all the rows of x are codewords of R and all the columns of x are codewords of C.*

2. *Let δ_R and δ_C be the relative distances of R and C respectively. Then, the code $R \otimes C$ has relative distance $\delta_R \cdot \delta_C$.*

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

3. *The tensor product operation is associative. That is, if $D : \mathbb{F}^{k_D} \to \mathbb{F}^{\ell_D}$ is a code then $(R \otimes C) \otimes D = R \otimes (C \otimes D)$.*

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

Fact 2.3.3. *Let R and C be as before and let $r \in R$ and $c \in C$. Define the tensor product $r \otimes c$ of r and c as the $\ell_C \times \ell_R$ matrix defined by $(r \otimes c)_{i,j} = c_i \cdot r_j$. Then, $r \otimes c$ is a codeword of $R \otimes C$.*

Proof. Observe that each row of $r \otimes c$ is equal to r multiplied by a scalar, and therefore it is a codeword of R. Similarly, each column of $r \otimes c$ is a codeword of C. By Item 1 of Fact 2.3.2, it follows that $r \otimes c \in R \otimes C$, as required. ■

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

Notation 2.3.4. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code. For every $m \in \mathbb{N}$ we denote by $C^m : \mathbb{F}^{k^m} \to \mathbb{F}^{\ell^m}$ the code $\underbrace{C \otimes C \otimes \ldots \otimes C}_{m}$. Formally, $C^m = C^{m-1} \otimes C$.

Notation 2.3.5. When referring to the code C^m and its codewords, we will often identify the sets of coordinates $[k^m]$ and $[\ell^m]$ with the hypercubes $[k]^m$ and $[\ell]^m$ respectively. Using the latter identification, one can view a string $x \in \mathbb{F}^{k^m}$ as a function $x : [k]^m \to \mathbb{F}$, and view strings in $\mathbb{F}^{\ell^m}$ similarly. With a slight abuse of notation, we say that C^m is systematic if for every codeword $c \in C^m$, the restriction of c to $[k]^m$ equals the message encoded by c^m. It is easy to see that if C is systematic (in the usual sense), then C^m is systematic as well.

Using Fact 2.3.2, one can prove by induction the following.

Fact 2.3.6. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code. Then, the codewords of C^m are precisely all the functions $f : [\ell]^m \to \mathbb{F}$ such that the restriction of f to any axis-parallel line of the hypercube is a codeword of C. That is, a function $f : [\ell]^m \to \mathbb{F}$ is a codeword of C^m if and only if for every $1 \leq t \leq m$ and $i_1, \ldots, i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that the function $f(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ is a codeword of C.*

Less standard features. We turn to prove two less standard features of the tensor product operation that will be useful in Section 2.4. The following claim expresses a coordinate of a tensor codeword using an expression of a "sum-check" form. We will use this claim later to show that one can use the sum-check protocol to evaluate the coordinates of a tensor codeword.

Claim 2.3.7. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, and let $m \in \mathbb{N}$. Then, for every coordinate $(i_1, \ldots, i_m) \in [\ell]^m$ there exist scalars $\alpha_{t,j} \in \mathbb{F}$ (for every $1 \leq t \leq m$ and $j \in [k]$) such that for every codeword $c \in C^m$ it holds that*

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

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

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

We now assume that the claim holds for some $m \in \mathbb{N}$, and prove it for $m+1$. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, let $c \in C^{m+1}$, and let $(i_1, \ldots, i_{m+1}) \in [\ell]^{m+1}$ be a coordinate of c. We first observe that by Fact 2.3.6, it holds that $c(\cdot, i_2, \ldots, i_{m+1})$ is a codeword of C. Thus, by the same considerations as in the case of $m = 1$, it follows that there exist coefficients $\alpha_{1,j_1} \in \mathbb{F}$ for $j_1 \in [k]$ such that

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

Next, observe that Fact 2.3.6 implies that for every j_1, it holds that $c(j_1, \underbrace{\cdot, \ldots, \cdot}_{m})$ is a codeword of C^m. The induction hypothesis now implies that there exist coefficients $\alpha_{t,j} \in \mathbb{F}$ (for every $2 \le t \le m+1$ and $j \in [k]$) such that for every $j_1 \in [k]$ it holds that

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

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

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

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

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

Claim 2.3.8. *Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a code, let $m \in \mathbb{N}$, and let $c \in C^m$. Then, for every sequence of scalars $\alpha_{t,j}$ (for every $2 \le t \le m$ and $j \in [\ell]$) it holds that the function $f : [\ell] \to \mathbb{F}$ defined by*

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

is a codeword of C.

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

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

is a codeword of C. To this end, let us observe that Fact 2.3.6 implies that for every $j_{m+1} \in [\ell]$, the function $g_{j_2} : [\ell]^m \to \mathbb{F}$ defined by

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

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

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

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

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

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

2.3.2 Multiplication codes

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

Notation 2.3.9. Let $\mathbb{F}$ be a finite field, let $\ell \in \mathbb{N}$, and let $u, v \in \mathbb{F}^{\ell}$. Then, we denote by $u \cdot v$ the string in $\mathbb{F}^{\ell}$ defined by

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

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

Proposition 2.3.10. *For every $k \in \mathbb{N}$, $\delta \in (0,1)$ and a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq \text{poly}\,(1/(1-\delta))$, there exists a triplet (C_A, C_B, C_M) of systematic linear codes over $\mathbb{F}$ that have the following properties:*

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

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

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

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

Remark 2.3.11. Again, it is trivial to construct codes as in Proposition 2.3.10 using polynomials. Indeed, taking C_A, C_B, and C_M to be Reed-Solomon codes of appropriate degree would yield codes with the same multiplication property and with better parameters. The novelty of our proof of Proposition 2.3.10 is that the construction of the codes is based on generic codes, and not on polynomial codes. Specifically, we will only use the tensor product operation.

Proof. The algorithm begins by invoking the algorithm of Fact 2.2.1 on input k, $\sqrt{\delta}$, and $\mathbb{F}$. This results in a code C with message length k, relative distance $\sqrt{\delta}$, and block length[3] $\ell_C = k/\text{poly}\left(1-\sqrt{\delta}\right) \leq k/\text{poly}\,(1-\delta)$. Next, the algorithm sets $\ell = \ell_C^2$ and constructs the generating matrices of the codes C_A, C_B, and C_M that are defined as follows:

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

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

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

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

It is easy to see that C_A, C_B, and C_M have the required parameters and that their generating matrices can be constructed in polynomial time. It remains to show that for every $c_A \in C_A$ and $c_B \in C_B$ it holds that $c_A \cdot c_B \in C_M$. To this end, recall that c_A is an $\ell_C \times \ell_C$ matrix all of whose rows are equal to some codeword c_r of C, whereas c_B is an $\ell_C \times \ell_C$ matrix all of whose columns are equal to some codeword c_c of C. Finally, observe that $c_A \cdot c_B = c_r \otimes c_c$, so it follows by Fact 2.3.3 that $c_A \cdot c_B \in C^2 = C_M$, as required. ∎

It is important to note that the multiplication of the codes of Proposition 2.3.10 is much more limited than the multiplication of polynomial codes. Specifically, the multiplication of polynomials can be applied many times. That is, if $c_1, \ldots, c_t$ are codewords of polynomial codes, then $c_1 \cdot \ldots \cdot c_t$ is also a codeword of a polynomial code, as long as the degrees of $c_1, \ldots, c_t$ are sufficiently small compared to t and $|\mathbb{F}|$. On the other hand, Proposition 2.3.10 only allows the multiplication of two codewords. This limitation is the reason that our emulation of the arithmetization technique in Section 2.5 is somewhat more complicated than the standard arithmetization.

Remark 2.3.12. It is possible to generalize the construction of Proposition 2.3.10 to allow multiplication of more codewords. However, the generalized construction yields codes with block length that is exponential in the number of multiplications allowed, and therefore we can only afford a constant number of multiplications.

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

Proposition 2.3.13. *Let C_1, C_2, and C_3 be codes of the same block length such that for every two codewords $\tilde{c}_1 \in C_1$ and $\tilde{c}_2 \in C_2$ it holds that $\tilde{c}_1 \cdot \tilde{c}_2 \in C_3$. Then, for every $m \in \mathbb{N}$, and for every $c_1 \in C_1^m$, $c_2 \in C_2^m$, it holds that $c_1 \cdot c_2 \in C_3^m$.*

Proof. Let $\ell \in \mathbb{N}$ be the block length of C_1, C_2, and C_3, and fix $m \in \mathbb{N}$. Let $c_1 \in C_1^m$, $c_2 \in C_2^m$ be codewords. We view c_1, c_2 and $c_1 \cdot c_2$ as functions from $[\ell]^m$ to $\mathbb{F}$. By Fact 2.3.6, for every $1 \le t \le m$ and $i_1, \ldots i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that $c_1(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ and $c_2(i_1, \ldots, i_{t-1}, \cdot, i_{t+1}, \ldots, i_m)$ are codewords of C_1 and C_2 respectively. The multiplication property of C_1, C_2, and C_3 now implies that for every $1 \le t \le m$ and $i_1, \ldots i_{t-1}, i_{t+1}, \ldots, i_m \in [\ell]$ it holds that

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

is a codeword of C_3. By applying Fact 2.3.6 in the opposite direction, the latter claim implies that $c_1 \cdot c_2$ is a codeword of C_3^m, as required. ∎

2.4 The Sum-Check Protocol Revisited

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

Definition 2.4.1. Let $C : \mathbb{F}^k \to \mathbb{F}^\ell$ be a systematic code, let $k', m \in \mathbb{N}$ be such that $k' \le k$. We say that a codeword $c : [\ell]^m \to \mathbb{F}$ of C^m is consistent with a function $f : [k']^m \to \mathbb{F}$ if c agrees with f on $[k']^m$.

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

Remark 2.4.2. Observe that every codeword is consistent with the message it encodes. In particular, if $k' = k$ in the above definition, then c is consistent with f if and only if c is the encoding of f. On the other hand, if $k' < k$, then there may be many codewords of C^m that are consistent with f.

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

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

Such a reduction is useful, for example, when the verifier can compute $d(\bar{r})$ easily, but can not compute $c(\bar{i})$ efficiently without the help of the prover. As a concrete example, consider the case where c is the low-degree extension of the truth table of a formula ϕ and d is the polynomial obtained from the arithmetization of ϕ. Then, the sum-check protocol reduces the (hard) task of evaluating the low-degree extension (i.e., computing $c(\bar{i})$) to the (easy) task of evaluating the arithmetization polynomial (i.e. computing $d(\bar{r})$). A related example is the original proof of co**NP** $\subseteq$ **IP**, where the sum-check protocol is used to reduce the evaluation of an exponential sum (which is hard for the verifier) to the evaluation of the polynomial obtained from the arithmetization (which is easy for the verifier).

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

Theorem 2.4.3 (The sum-check protocol, informal)**.** *Let C and D be codes, and let $c \in C^m$ and $d \in D^m$ be codewords such that d is consistent with the message encoded by c. Then, there exists an interactive protocol that takes as input a claim of the form "$c\left(\bar{i}\right) = u$" and behaves as follows:*

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

Theorem 2.4.4 (The sum-check protocol, formal)**.** *There exists a public coin interactive protocol between an unbounded prover and a polynomial time verifier that behaves as follows:*

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

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

- ***Completeness:*** *If $c\left(\bar{i}\right) = u$, then there exists a strategy for the prover that makes the verifier output with probability 1 a pair $(\bar{r}, v)$ such that $d\left(\bar{r}\right) = v$.*

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

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

Remark 2.4.5. The statement of Theorem 2.4.4 may seem confusing, since the codewords c and d are not given to the prover and verifier in any way. In fact, the codewords c and d are chosen by the prover, and may be chosen arbitrarily, subject to d being consistent with the message encoded by c.

However, in this work we will use Theorem 2.4.4 as a sub-protocol of higher level protocols. In those applications, the prover will be forced to use specific choices of c and d in order to convince the verifier of the high level protocol. In particular, those specific choices of c and d will be determined by the high level protocol.

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

We begin with recalling that by Claim 2.3.7, there exist scalars $\alpha_{t,j} \in \mathbb{F}$ for $1 \leq t \leq m$ and $j \in [k]$ such that for every choice of c it holds that $c(\bar{i}) = u$ if and only if

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

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

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

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

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

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

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

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

Observe that by Claim 2.3.8 the function h is a codeword of D. This follows by applying Claim 2.3.8 to the function $d(r_1, \ldots, r_{t-1}, \cdot, \cdot, \ldots, \cdot)$, while recalling that this function is a codeword of D^{m-t+1} by Fact 2.3.6.

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

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

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

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

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

2.5 A Proof of co**NP** ⊆ **IP**

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

2.5.1 Proof overview

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

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

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

Proof via Arithmetization. We now show a protocol for verifying the value of $h_\phi(\overline{i})$ that uses arithmetization, and we will later show how to avoid the use of arithmetization. Let ϕ a Boolean formula over n variables, and let p_ϕ the polynomial that is obtained from the arithmetization of ϕ. We observe that p_ϕ is *consistent* with the formula ϕ, and it can be shown that p_ϕ is a codeword of some tensor code[4]. Therefore, we can use the sum-check protocol of Theorem 2.4.4 to reduce the task of

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

verifying a claim of the form $h_\phi(\bar{i}) = u$ to the task of verifying a claim of the form $p_\phi(\bar{r}) = v$. Finally, observe that the verifier can compute the value $p_\phi(\bar{r})$ by itself, and thus verify that $p_\phi(\bar{r}) = v$. This concludes the description of the protocol.

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

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

It can be observed that those properties are the only properties of p that we needed. This yields the following protocol: In order to verify a claim of the form $h_\phi(\bar{i}) = u$, the verifier reduces it to a claim of the form $c_{M,\phi}(\bar{r}) = v$ using the sum-check protocol of Theorem 2.4.4. Then, the verifier computes $c_{M,\phi}(\bar{r})$ by itself, and accepts if and only if $c_{M,\phi}(\bar{r}) = v$.

Specialized formulas and the construction of $c_{M,\phi}$. In general, we do not know how to construct the above codeword $c_{M,\phi}$ for every formula ϕ, but only for "specialized formulas", which will be discussed shortly. To resolve this issue, the protocol begins by transforming ϕ into an equivalent specialized formula ϕ_{sp}, and then proceeds as before while working with ϕ_{sp}. This issue is a direct consequence of the limitation of our multiplication codes that allows only one multiplication, as discussed after Proposition 2.3.10.

A "specialized formula" is a formula ϕ that can be written as $\phi = \phi_{\mathrm{once}} \wedge \phi_{\mathrm{eq}}$, where ϕ_{once} is a read-once formula, and ϕ_{eq} is a conjunction of equality constraints. The point is that since ϕ_{once} and ϕ_{eq} are very simple, it is easy to evaluate the encoding of their truth tables via any (tensor) code. Now, we let $c_{A,\phi_{\mathrm{once}}}$ and $c_{B,\phi_{\mathrm{eq}}}$be the encodings of ϕ_{once} and ϕ_{eq} via the multiplication codes of Proposition 2.3.10, and set $c_{M,\phi} = c_{A,\phi_{\mathrm{once}}} \cdot c_{B,\phi_{\mathrm{eq}}}$. It is easy to see that the codeword $c_{M,\phi}$ is consistent with ϕ. Moreover, the codeword $c_{M,\phi}$ is easy to evaluate, since $c_{A,\phi_{\mathrm{once}}}$ and $c_{B,\phi_{\mathrm{eq}}}$ are easy to evaluate.

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

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

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

2.5.2 Full proof

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

Definition 2.5.1. A specialized formula is a formula ϕ that can be written as $\phi = \phi_{\text{once}} \wedge \phi_{\text{eq}}$, where

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

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

Lemma 2.5.2. *Let $\mathbb{F}$ be a finite field, and let $C_M : \mathbb{F}^4 \to \mathbb{F}^{\ell}$ be the multiplication code generated by Proposition 2.3.10 for $k = 2$ and any relative distance δ. Then, there exists a polynomial time algorithm that behaves as follows:*

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

We prove Lemma 2.5.2 in Section 2.5.2.1, but first, we show how to prove that co**NP** $\subseteq$ **IP** based on Lemma 2.5.2. To this end, we use the following standard fact that says that every formula can be transformed into an "equivalent" specialized formula.

Fact 2.5.3. *Let ϕ be a Boolean formula over n variables, and let m be the total number of occurrences of variables in ϕ. Then, there exists a specialized formula ϕ_{sp} over m variables that is satisfiable if and only if ϕ is satisfiable. Furthermore, ϕ_{sp} can be computed in polynomial time from ϕ. We refer to ϕ_{sp} as the the specialized version of ϕ.*

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

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

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

Theorem 2.5.4. co**NP** $\subseteq$ **IP**

Proof. We design a protocol for verifying the unsatisfiability of a Boolean formula. Let ϕ be a Boolean formula over n variables and m occurrences, and let ϕ_{sp} be its specialized version constructed by Fact 2.5.3. It suffices to design a protocol for verifying the satisfiability of ϕ_{sp}.

Let $\mathbb{F}$ be a finite field of size at least $4m$, let C_M be the code generated by Proposition 2.3.10 for $k = 2$ and relative distance $\delta = 1 - 1/2m$, and let $c_M = c_{M,\phi_{\text{sp}}}$ be the codeword whose existence is guaranteed by Lemma 2.5.2. Let $H : \mathbb{F}^2 \to \mathbb{F}^{\ell_H}$ be any systematic linear code of distance at least $1 - 1/2m$ (for example, one may use the $|\mathbb{F}|$-ary Hadamard code), and let $h = h_{\phi_{\text{sp}}}$ be the encoding of the truth table of ϕ_{sp} via H^m. Note that h is the *(unique) encoding* of the truth table of ϕ_{sp} via H^m, but may be hard to evaluate, while c_M is merely a codeword of $(C_M)^m$ that is *consistent* with ϕ_{sp}, but is easy to evaluate.

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

At the beginning of the protocol, the verifier chooses a uniformly distributed tuple $\bar{i} \in [\ell_H]^m$, and sends it to the prover. The prover should prove to the verifier that $h(\bar{i}) = 0$. To this end, the prover and the verifier engage in the sum-check protocol of Theorem 2.4.4 with $C = H$, $D = C_M$, $c = h$, $d = c_M$, $\bar{i} = \bar{i}$, and $u = 0$. If the verifier does not reject at this stage, then the sum-check protocol outputs a pair $(\bar{r}, v)$ that is expected to satisfy $c_M(\bar{r}) = v$. Finally, the verifier uses the algorithm of Lemma 2.5.2 to compute $c_M(\bar{r})$, accepts if $c_M(\bar{r}) = v$, and rejects otherwise.

For the completeness of the protocol, note that if ϕ_{sp} is unsatisfiable, then $h(\bar{i}) = 0$. Therefore, by the completeness of the sum-check protocol of of Theorem 2.4.4, there exists a prover strategy that guarantees that the verifier does not reject and outputs a pair $(\bar{r}, v)$ such that $c_M(\bar{r}) = v$. It is easy to see that if the prover uses this strategy, the verifier will always accept.

For the soundness of the protocol, observe that if ϕ_{sp} is satisfiable, then $h(\bar{i}) \neq 0$ with probability at least $\frac{1}{2}$. Conditioned on $h(\bar{i}) \neq 0$, the soundness of Theorem 2.4.4 guarantees that with probability at least $m \cdot (1 - \delta_{C_M}) \geq \frac{1}{2}$, the verifier either rejects or outputs a pair $(\bar{r}, v)$ such that $c_M(\bar{r}) \neq v$, in which case the verifier rejects in the next step. It follows that if ϕ is satisfiable, then the verifier rejects with probability at least $\frac{1}{4}$, which suffices for our purposes. ■

2.5.2.1 Proof of Lemma 2.5.2

We turn to proving Lemma 2.5.2. Let ϕ be a specialized Boolean formula over n variables, and let (C_A, C_B, C_M) be the multiplication code generated by Proposition 2.3.10 for $k = 2$ and any relative distance δ. We seek to construct a codeword $c_M = c_{M,\phi}$ of $(C_M)^n$ that is consistent with ϕ, and such that the value of c_M at any coordinate can be computed in polynomial time.

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

We now show how to construct the codeword c_M that is consistent with ϕ. Let c_A be the encoding of the truth table of ϕ_{once} via $(C_A)^n$, and let c_B be the encoding of the truth table of ϕ_{eq} via $(C_B)^n$. We choose $c_M \overset{\text{def}}{=} c_A \cdot c_B$. Observe that $c_A \cdot c_B$ is indeed consistent with ϕ, and that it is a codeword of $(C_M)^n$ by Proposition 2.3.13.

It remains to show that for every coordinate $\bar{i}$ of c_M, the value $c_M(\bar{i})$ can be computed efficiently. Let ℓ denote the block length of C_A, C_B, and C_M. Propositions 2.5.6 and 2.5.8 below imply that for every $\bar{i} \in [\ell]^n$, the values $c_A(\bar{i})$ and $c_B(\bar{i})$ can be computed efficiently. It follows that for every $\bar{i} \in [\ell]^n$, we can compute the value $c_M(\bar{i})$ efficiently by first computing $c_A(\bar{i})$ and $c_B(\bar{i})$ and then setting $c_M(\bar{i}) = c_A(\bar{i}) \cdot c_B(\bar{i})$. This concludes the proof of Lemma 2.5.2 up to the proofs of Propositions 2.5.6 and 2.5.8.

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

Notation 2.5.5. In the statements of the following propositions, we denote by $C : \mathbb{F}^2 \to \mathbb{F}^{\ell_C}$ a fixed arbitrary code, and for every Boolean formula φ over n variables, we denote by c_φ the encoding of the truth table of φ via C^n.

Proposition 2.5.6 (Codeword for read-once formulas)**.** *There exists a polynomial time algorithm such that when the algorithm is given as input a read-once Boolean formula φ, the generator matrix of a code C, and a coordinate $\bar{i}$ of c_φ, the algorithm outputs $c_\varphi(\bar{i})$.*

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

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

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

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

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

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

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

2. Suppose that $\varphi = \neg\varphi'$. In this case, the algorithm computes $c_{\bar{1}_n}(\bar{i})$ and $c_{\varphi'}(\bar{i})$ and outputs $c_{\bar{1}_q}(\bar{i}) - c_{\varphi'}(\bar{i})$. The value $c_{\varphi'}(\bar{i})$ is computed recursively. In order to compute $c_{\bar{1}_n}(\bar{i})$, observe that $c_{\bar{1}_n} = \underbrace{c_{\bar{1}_1} \otimes \ldots \otimes c_{\bar{1}_1}}_{n}$. It therefore follows that
$$c_{\bar{1}_n}(\bar{i}) = c_{\bar{1}_1}(i_1) \cdot \ldots \cdot c_{\bar{1}_1}(i_n)$$
Thus, in order to compute $c_{\bar{1}_n}(\bar{i})$, the algorithm computes $c_{\bar{1}_1}$ by encoding the vector $(1, 1)$ with C, and outputs $c_{\bar{1}_1}(i_1) \cdot \ldots \cdot c_{\bar{1}_1}(i_n)$.

3. Suppose that $\varphi = \varphi_1 \wedge \varphi_2$. Again, we assume that the first n_1 variables of φ are the variables of φ_1, and that the last n_2 variables of φ are the variables of φ_2. Also, observe that $n = n_1 + n_2$. Then, it holds that
$$c_\varphi(\bar{i}) = c_{\varphi_1}(i_1, \ldots, i_{n_1}) \cdot c_{\varphi_2}(i_{n_1+1}, \ldots, i_n)$$
The algorithm thus computes $c_{\varphi_1}(i_1, \ldots, i_{n_1})$ and $c_{\varphi_2}(i_{n_1+1}, \ldots, i_n)$ recursively and outputs their product.

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

Remark 2.5.7. Note that the assumption that every variable occurs exactly once in φ is critical for the proof of Proposition 2.5.6. Specifically, this assumption is used in handling the case of $\varphi = \varphi_1 \wedge \varphi_2$, and allows us to simulate the effect of multiplication using the tensor product operation (i.e., by setting $c_\varphi = c_{\varphi_1} \otimes c_{\varphi_2}$). Without the assumption, it could be the case that φ_1 and φ_2 have common variables, which would imply that $c_\varphi \neq c_{\varphi_1} \otimes c_{\varphi_2}$.

Proposition 2.5.8 (Codeword for equality constraints)**.** *There exists a polynomial time algorithm such that when the algorithm is given as input a Boolean formula φ which is a conjunction of equality constraints, the generator matrix of C, and a coordinate $\bar{i}$ of c_φ, the algorithm outputs $c_\varphi(\bar{i})$.*

Proof. We first deal with the special case in which φ is satisfied if and only if *all* its variables are equal to each other. Let $\bar{i} \in [\ell_C]^n$ be a coordinate. We wish to compute $c_\varphi(\bar{i})$ efficiently. By Claim 2.3.7, there exist scalars $\alpha_{t,j} \in \mathbb{F}$ (for every $1 \leq t \leq n$ and $j \in \{0,1\}$) such that

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

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

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

Now, the above sum is easy to compute, since by Claim 2.3.7 the coefficients $\alpha_{t,j}$ can be computed efficiently.

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

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

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

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

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

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

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

Remark regarding algebrization. Recall that the arithmetization technique is the only non-relativizing ingredient of the proof of the **IP** theorem. Indeed, a main motivation of the algebrization framework of [AW08] was to try to capture the arithmetization technique. While our arithmetization generalization (Lemma 2.5.2) does not seem to fit into the algebrization framework, one can prove the following "algebrization-like" variant of this lemma: Let $\mathcal{O} = \{\mathcal{O}_n : \{0,1\}^n \to \{0,1\}\}_n$ be an infinite sequence of Boolean oracles, and let us denote $C_{A,\mathcal{O}} = \{C_{A,\mathcal{O}_n}\}_n$ where $C_{A,\mathcal{O}_n}$ is the encoding of the truth table $\mathcal{O}_n$ by $(C_A)^n$. Then, there exists an algorithm that given oracle access to $C_{A,\mathcal{O}}$ satisfies the following requirement: when the algorithm is given as input a specialized formula φ that contains oracle predicates from the sequence $\mathcal{O}$ and a coordinate $\bar{i}$ of $c_{M,\varphi}$, the algorithm outputs $c_{M,\varphi}(\bar{i})$. Here $c_{M,\varphi}$ is a codeword of $(C_M)^n$ that is consistent with φ, as before.

2.6 The Proof of IP = PSPACE

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

Theorem 2.6.1. IP = PSPACE.

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

$$\underset{y_1\in\{0,1\}}{\mathcal{Q}_1}\ \underset{y_2\in\{0,1\}}{\mathcal{Q}_2} \dots \underset{y_n\in\{0,1\}}{\mathcal{Q}_n}\ \phi(y_1,\dots,y_n) \tag{2.1}$$

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

We begin with an overview of the proof in Section 2.6.1, and then give the full details in Section 2.6.2. We mention that our proof borrows ideas from the work of [GKR08].

2.6.1 Proof overview

The formulas ψ_i. Given a quantified formula as in (2.1), we define the following quantified formulas

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

That is, ψ_t is a formula in which $y_1,\dots,y_t$ are free variables and $y_{t+1},\dots,y_n$ are bounded variables. In particular, ψ_0 is the original quantified formula and ψ_n is the formula ϕ. We also consider the encodings of the truth table of ψ_t via $(C_A)^t$ and $(C_B)^t$, and denote them by $c_{A,t}$ and $c_{B,t}$ (where (C_A,C_B,C_M) are the multiplication codes of Proposition 2.3.10).

We mention that the actual proof will work with the specialized version ϕ_{sp} instead of ϕ itself (see Definition 2.5.1 and Fact 2.5.3). We ignore this technicality throughout this overview.

The structure of the protocol. Our interactive protocol begins by reducing the task of verifying the validity of Formula (2.1) to the task of verifying a claim of the form

$$c_{A,1}(\bar{i}_1) = v_{A,1} \quad \text{and} \quad c_{B,1}(\bar{i}_1) = v_{B,1} \tag{2.2}$$

where $\bar{i}_1$ is a coordinate of $c_{A,1}$ and $c_{B,1}$ - note that $\bar{i}_1$ is shared by both equalities in (2.2).

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

$$c_{A,t}(\bar{i}_t) = v_{A,t} \quad \text{and} \quad c_{B,t}(\bar{i}_t) = v_{B,t} \tag{2.3}$$

Throughout the t-th iteration, the parties engage in a sub-protocol, in order to reduce the task of verifying the claim in (2.3) to the task of verifying a claim of the same form about $c_{A,t+1}$ and $c_{B,t+1}$.

Eventually, the parties end up with a claim about $c_{A,n}$ and $c_{B,n}$. This means that the prover is required to prove a claim about encodings of the truth table of $\psi_n = \phi$, which can be done in the same way as in the proof of co**NP** $\subseteq$ **IP**: The parties engage in the sum-check protocol in order to reduce the claim about $c_{A,n}$ and $c_{B,n}$ to a claim about a codeword of $(C_M)^n$ that is consistent with the truth table of ϕ, and then the verifier checks the latter claim by itself, by using the "arithmetization generalization" (Lemma 2.5.2).

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

$$c_{M,t}(\bar{j}) = c_{A,t+1}(\bar{j},0)\cdot c_{B,t+1}(\bar{j},1) \tag{2.4}$$

Observe that $c_{M,t}$ is indeed a codeword of $(C_M)^t$ and that it is consistent with the truth table of ψ_t. Recall that our purpose is to reduce the verification of the claim in (2.3) to the verification of the same claim for $t+1$. The codeword $c_{M,t}$ serves as a "bridge" between those two claims: On the one hand, $c_{M,t}$ is consistent with the message encoded by $c_{A,t}$ and $c_{B,t}$, whereas on the other hand $c_{M,t}$ is related to $c_{A,t+1}$ and $c_{B,t+1}$ by Equality (2.4). Our strategy is to first reduce the verification of the claim about $c_{A,t}$ and $c_{B,t}$ to the verification of a claim about $c_{M,t}$, and then reduce the latter to the verification of a claim about $c_{A,t+1}$ and $c_{B,t+1}$.

More specifically, the parties begin the iteration by reducing the task of verifying the claim in (2.3) to the task of verifying an equality of the form

$$c_{M,t}(\overline{r}) = v_{M,t} \tag{2.5}$$

Such a reduction can be done by invoking the sum-check protocol of Theorem 2.4.4 twice in parallel, once with $C = C_A$ and $D = C_M$, and once with $C = C_B$ and $D = C_M$, with the verifier using the same randomness for both invocations. The reason for using the same randomness for both invocations is that we want both invocations to output the same coordinate $\overline{r}$.

Next, the prover sends to the verifier two functions f_A and f_B, which are expected to be $c_{t+1,A}(\overline{r},\cdot)$ and $c_{t+1,B}(\overline{r},\cdot)$ respectively. The verifier checks that f_A and f_B are indeed codewords of C_A and C_B respectively, and that $f_A(0)\cdot f_B(1) = v_{M,t}$, where $v_{M,t}$ is the value from Equality (2.5). Finally, the verifier chooses a random coordinate s, and the parties enter the next iteration with the claim

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

2.6.2 The full proof

Fix a quantified formula

$$\underset{y_1\in\{0,1\}}{\mathcal{Q}_1} \dots \underset{y_n\in\{0,1\}}{\mathcal{Q}_n} \phi(y_1,\dots,y_n) \tag{2.6}$$

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

Moving to specialized formulas. Our first step is moving to work with specialized formulas, which will allow us to use the "arithmetization generalization" (Lemma 2.5.2). To this end, consider the specialized version ϕ_{sp} of ϕ, whose existence is guaranteed by Fact 2.5.3, and let us denote its variables by $x_1,\dots,x_m$. Recall that each variable $x_{i'}$ of ϕ_{sp} is treated as a "copy" of some variable y_i. Let us relabel the variables of ϕ_{sp} such that for each $1 \le i \le n$, the variable x_i is a copy of y_i. Now, consider the formula

$$\underset{x_1\in\{0,1\}}{\mathcal{Q}_1} \dots \underset{x_n\in\{0,1\}}{\mathcal{Q}_n} \underset{x_{n+1}\in\{0,1\}}{\exists} \dots \underset{x_m\in\{0,1\}}{\exists} \phi_{\text{sp}}(x_1,\dots,x_m) \tag{2.7}$$

Observe that that Formula (2.6) is valid if and only if Formula (2.7) is valid. For the rest of the proof, we will work with the Formula (2.7). For convenience, we will denote Formula (2.7) as

$$\underset{x_1\in\{0,1\}}{\mathcal{Q}_1} \dots \underset{x_m\in\{0,1\}}{\mathcal{Q}_m} \phi_{\text{sp}}(x_1,\dots,x_m) \tag{2.8}$$

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

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

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

Let (C_A, C_B, C_M) be the multiplication codes that result from invoking the algorithm of Proposition 2.3.10 with $k \overset{\text{def}}{=} 2$, $\delta \overset{\text{def}}{=} 1 - \frac{1}{2(m+1)^2}$, and with sufficiently large finite field $\mathbb{F}$, and let ℓ be their block length. For every $1 \le t \le m$, we define $c_{A,t}$ and $c_{B,t}$ to be the encodings of the truth table of ψ_t via $(C_A)^t$ and $(C_B)^t$ respectively.

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

Theorem 2.6.2 (Single iteration, informal)**.** *There exists an interactive protocol that takes as input a claim of the form "$c_{A,t}(\bar{i}_t) = v_{A,t}$ and $c_{B,t}(\bar{i}_t) = v_{B,t}$" and and behaves as follows:*

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

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

Theorem 2.6.3 (Single iteration, formal)**.** *Let ϕ_{sp}, m, C_A, and C_B be defined as above, and for every $1 \le t \le m-1$ let ψ_t, $c_{A,t}$, and $c_{B,t}$ be defined as above with respect to ϕ_{sp}. There exists an interactive protocol between an unbounded prover and a polynomial time verifier that satisfies the following requirements:*

- ***Input:*** *The parties enter the protocol with a common input $(\bar{i}_t, v_{A,t}, v_{B,t})$, where $\bar{i}_t \in [\ell]^t$ and $v_{A,t}, v_{B,t} \in \mathbb{F}$. Additional inputs are the numbers m, t, the generating matrices of C_A, C_B, C_M, and the quantified formula in (2.8).*
- ***Output:*** *At the end of the protocol, the verifier either rejects, or outputs a triplet $(\bar{i}_{t+1}, v_{A,t+1}, v_{B,t+1})$, where $\bar{i}_{t+1} \in [\ell_D]^{t+1}$ and $v_{A,t+1}, v_{B,t+1} \in \mathbb{F}$.*

The output satisfies the following requirements:

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

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

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

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

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

Furthermore, observe that $c_{M,t}$ is indeed a codeword of $(C_M)^t$ for the following reasons: For each $b \in \{0,1\}$, it holds that $c_{A,t+1}(\cdot,b)$ and $c_{B,t+1}(\cdot,b)$ are codewords of $(C_A)^t$ and $(C_B)^t$ respectively. Furthermore, by Propositions 2.3.10 and 2.3.13 it holds that the multiplication of codewords of $(C_A)^t$ and $(C_B)^t$ yields a codeword of $(C_M)^t$. Finally, $(C_M)^t$ is a linear code, and therefore a sum of codewords of $(C_M)^t$ yields a codeword of $(C_M)^t$ (this is only relevant for the case that $\mathcal{Q}_{t+1} = \exists$).

The protocol starts with the parties invoking the sum-check protocol (Theorem 2.4.4) twice in parallel, using the same randomness for both invocations: The first invocation is done with $C = C_A$, $D = C_M$, $c = c_{A,t}$, $d = c_{M,t}$, $\bar{i} = \bar{i}_t$, and $u = v_{A,t}$, and the second invocation is done with $C = C_B$, $D = C_M$, $c = c_{B,t}$, $d = c_{M,t}$, $\bar{i} = \bar{i}_t$, $u = v_{B,t}$. The two invocations result in two pairs $(\bar{r}, v_{M,t})$, $\left(\bar{r}, v'_{M,t}\right)$, where $\bar{r} \in [\ell]^t$ and $v_{M,t}, v'_{M,t} \in \mathbb{F}$ - note that $\bar{r}$ is common to both pairs since the verifier uses the same randomness for both invocations (see the "Furthermore" part of Theorem 2.4.4). The verifier checks that $v_{M,t} = v'_{M,t}$, and rejects otherwise.

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

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

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

The completeness of the protocol is easy to verify. We turn to prove the soundness of the protocol. Without loss of generality, suppose that $c_{A,t}\left(\bar{i}_t\right) \neq v_{A,t}$ and that $\mathcal{Q}_i = \forall$- the cases where $c_{B,t}\left(\bar{i}_t\right) \neq v_{B,t}$ and $\mathcal{Q}_i = \exists$ can be handled similarly. By the soundness part of Theorem 2.4.4, with probability at least $1 - t \cdot (1-\delta)$ it holds that either the verifier rejects or $c_M(\bar{r}) \neq v_{M,t}$. Now, if the verifier does not reject, then it must hold that $f_A(0) \cdot f_B(1) = v_{M,t}$, and therefore $f_A(0) \cdot f_B(1) \neq c_M(\bar{r})$. By the definition of c_M, this implies that either $f_A(0) \neq c_{A,t+1}(\bar{r},0)$ or that $f_B(1) \neq c_{B,t+1}(\bar{r},1)$ - without loss of generality, assume the first. In this case, f_A is a codeword of C_A that differs from the codeword $c_{A,t+1}(\bar{r},\cdot)$. Thus, with probability at least δ, it holds that $f_A(j) \neq c_{A,t+1}(\bar{r},j)$, or in other words, that $c_{A,t+1}(\bar{i}_{t+1}) \neq v_{A,t+1}$. By the union bound, it follows that with probability at least $1-(t+1)\cdot(1-\delta)$, the verifier either rejects or $c_{A,t+1}(\bar{i}_{t+1}) \neq v_{A,t+1}$, as required. ■

The full protocol. We finally turn to describe the full protocol for verifying the validity of the Quantified Formula (2.8). At the beginning of the protocol, the prover sends two functions $g_A, g_B : [\ell] \to \mathbb{F}$, that are supposed to be $c_{A,1}$ and $c_{B,1}$ respectively if the prover is honest. The verifier checks that

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

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

The parties then proceed in iterations for $1 \leq t \leq m-1$, each iteration invoking the protocol of Theorem 2.6.3. The parties finish the last iteration with a triplet $\left(\bar{i}_m, v_{A,m}, v_{B,m}\right)$, such that if the prover is honest it holds that $c_{A,m}(\bar{i}_m) = v_{A,m}$ and $c_{B,m}(\bar{i}_m) = v_{B,m}$. Observe that $c_{A,m}$ and $c_{B,m}$ are the encodings of the truth table of ϕ_{sp} via $(C_A)^m$ and $(C_B)^m$ respectively.

By Lemma 2.5.2, there exists a codeword $c_{M,\phi_{\text{sp}}}$ of $(C_M)^m$ that is consistent with ϕ_{sp}, such that for every $\bar{j} \in [\ell]^m$, the value $c_{M,\phi_{\text{sp}}}(\bar{j})$ can be computed efficiently (note that $c_{M,\phi_{\text{sp}}}$ is *not* the same as the codeword $c_{M,t}$ in the proof of Theorem 2.6.3). The parties now engage in the sum-check protocol (Theorem 2.4.4) twice: The first invocation is with $C = C_A$, $D = C_M$, $c = c_{A,m}$, $d = c_{M,\phi_{\text{sp}}}$, $\bar{i} = \bar{i}_m$, and $u = v_{A,m}$, and the second invocation with $C = C_B$, $D = C_M$, $c = c_{B,m}$, $d = c_{M,\phi_{\text{sp}}}$, $\bar{i} = \bar{i}_m$, $u = v_{B,m}$. The two invocations result in two pairs $(\bar{r}, v_M)$, $(\bar{r}', v'_M)$, where $\bar{r}, \bar{r}' \in [\ell]^m$ and $v_M, v'_M \in \mathbb{F}$. Finally, the verifier computes $c_{M,\phi_{\text{sp}}}(\bar{r})$ and $c_{M,\phi_{\text{sp}}}(\bar{r}')$ by itself, accepts if $c_{M,\phi_{\text{sp}}}(\bar{r}) = v_M$ and $c_{M,\phi_{\text{sp}}}(\bar{r}') = v'_M$, and rejects otherwise.

Remark 2.6.4. The full protocol could be defined slightly differently. Specifically, one could replace the first stage of the protocol with an additional invocation Theorem 2.6.3 for $t = 0$. This approach has a formal problem, since ψ_0 is not a function but rather a scalar, but the approach can still be implemented by a suitable modification of the relevant definitions. We preferred the current presentation.

Analysis. When given as input a quantified formula over n variables and m occurrences, the foregoing protocol uses $O(m^2)$ rounds: In the first stage, the protocol invokes for each $1 \leq t \leq m-1$ a sum-check protocol of t rounds (twice in parallel), and in the second stage a sum-check protocol of m rounds is invoked (twice). This can be compared to the protocols of [Sha92, She92, GKR08], which use $O(n^2)$ rounds. This difference between our protocol and the previous protocols results from the fact that we work with the specialized formula (2.7) instead of the original formula (2.6).

The completeness of the protocol is easy to verify. As for the soundness, note that due to considerations similar to those of the proof of Theorem 2.6.3, if the input quantified formula is not valid then with probability at least δ it holds that either $c_{A,1}(i_1) \neq v_{A,1}$ or that $c_{B,1}(i_1) \neq v_{B,1}$. By applying the soundness of Theorem 2.6.3 for the $m-1$ iterations, we get that with probability at least $1 - m \cdot (m+1) \cdot (1-\delta)$ it holds that either $c_{A,m} \neq v_{A,m}$ or that $c_{B,m} \neq v_{B,m}$. Finally, due to the soundness of the sum-check protocol (Theorem 2.4), we get that with probability at least $1 - m \cdot (1-\delta)$ it holds that either $c_{M,\phi}(\bar{r}) \neq v_M$ or that $c_{M,\phi_{\text{sp}}}(\bar{r}') \neq v'_M$, in which case the verifier rejects. By applying the union bound, it follows that if the input quantified formula is not valid, then the verifier rejects with probability at least $1 - (m+1)^2 \cdot (1-\delta) \geq \frac{1}{2}$, as required.

Chapter 3

Combinatorial PCPs with efficient verifiers

3.1 Introduction

3.1.1 Background and Our Results

A PCP (Probabilistically Checkable Proof) is a proof system that allows checking the validity of a claim by reading only a constant number of bits of the proof. The PCP theorem asserts the existence of PCPs of polynomial length for any claim that can be stated as membership in an **NP** language. In this chapter, we consider the efficiency of the verification procedure, and provide a combinatorial construction of PCPs whose verification procedure is as efficient as the ones obtained from the algebraic constructions.

Let L be a language in **NP**, and consider a PCP verifier for verifying claims of the form "$x \in L$". Note that, while in order to verify that $x \in L$, the verifier must run in time which is at least linear in the length of x (since the verifier has to read x), the effect of the proof length on the verifier's running time may be much smaller. Using the algebraic techniques, one can construct PCP verifiers whose running time depends only *poly-logarithmically* on the proof length. On the other hand, the verifiers obtained from Dinur's proof of the PCP theorem are not as efficient, and depend polynomially on the proof length. While this difference does not matter much in the context of standard PCPs for **NP**, it is very significant in two related settings that we describe below.

PCPs for NEXP. While the PCP theorem is most famous for giving PCP systems for languages in **NP**, it can be scaled to higher complexity classes, up to **NEXP**. Informally, the PCP theorem for **NEXP** states that for every language $L \in$ **NEXP**, the claim that $x \in L$ can be verified by reading a constant number of bits from an exponentially long proof, where the verifier runs in *polynomial time*. Note that in order to meet the requirement that the verifier runs in polynomial time, one needs to make sure the verifier's running time depends only *poly-logarithmically on the proof length*.

The PCP theorem for **NEXP** can be proved by combining the algebraic proof of the PCP theorem for **NP** (of [AS98, ALM$^+$98]) with the ideas of Babai et al. [BFL91]. Dinur's proof, on the other hand, is capable of proving the PCP theorem for **NP**, but falls short of proving the theorem for **NEXP** due to the running time of its verifiers. Our first main result in this chapter is the first combinatorial proof of the PCP theorem for **NEXP**:

Theorem 3.1.1 (PCP theorem for **NEXP**, informal)**.** *For every $L \in$ **NEXP**, there exists a probabilistic polynomial time verifier that verifies claims of the form $x \in L$ by reading only a constant number of bits from a proof of length* $\exp(\text{poly}(|x|))$

Indeed, Theorem 3.1.1 could already be proved by combining the works of [AS98, ALM$^+$98] and [BFL91], but we provide a combinatorial proof of this theorem.

PCPs of Proximity. PCPs of Proximity ([BSGH$^+$06, DR06]) are a variant of PCPs that allows a super-fast verification of claims while compromising on their accuracy. Let $L \in \mathbf{NP}$ and suppose we wish to verify the claim that $x \in L$. Furthermore, suppose that we are willing to compromise on the accuracy of the claim, in the sense that we are willing to settle with verifying that x is *close* to some string in L. PCPs of Proximity (abbreviated PCPPs) are proofs that allow verifying that x is *close* to L by reading only a constant number of bits from *both* x and the proof. Using the algebraic methods, one can construct PCPPs with verifiers that run in time which is *poly-logarithmic in* $|x|$ (see, e.g., [BSGH$^+$05]). Note that this is highly non-trivial even for languages L that are in $\mathbf{P}$.

One can also construct PCPPs using Dinur's techniques, but the resulting verifiers are not as efficient, and run in time $\text{poly}(|x|)$. While those verifiers still have the property that they only read a constant number of bits from x, they seem to lose much of their intuitive appeal. Our second main result in this chapter is a combinatorial construction of PCPPs that allow super-fast verification:

Theorem 3.1.2 (PCPPs with super-fast verifiers, informal)**.** *For every $L \in \mathbf{NP}$, there exists a probabilistic verifier that verifies claims of the form "x is close to L" by reading only a constant number of bits from x and from a proof of length* $\text{poly}(|x|)$*, and that runs in time* $\text{poly}(\log |x|)$.

Again, a stronger version of Theorem 3.1.1 was already proved in [BSGH$^+$05], but we provide a combinatorial proof of this theorem.

3.1.2 Our Techniques

Our techniques employ ideas from both the works of Dinur [Din07] and Dinur and Reingold [DR06]. In this section we review these works and describe the new aspects of our work. For convenience, we focus on the construction of super-fast PCPPs (Theorem 3.1.2).

3.1.2.1 On Dinur's proof of the PCP theorem

We begin by taking a more detailed look at Dinur's proof of the PCP theorem, and specifically at her construction of PCPP verifiers. The crux of Dinur's construction is a combinatorial amplification technique for increasing the probability of PCPP verifiers to reject false claims. Specifically, given a PCPP verifier that uses a proof of length ℓ, and rejects false claims with probability ρ, the amplification transforms the verifier into a new verifier that rejects false claims with probability $2 \cdot \rho$, but uses a proof of length $\beta \cdot \ell$ for some constant $\beta > 1$.

Using the amplification technique, we can construct PCPP verifiers that use proofs of polynomial length as follows. Let $L \in \mathbf{NP}$. We first observe that L has a trivial PCPP verifier that uses proofs of length $\text{poly}(n)$ and has rejection probability $\frac{1}{\text{poly}(n)}$ - for example, consider the verifier that reduces L to the problem of 3SAT, then verifies that a given assignment satisfies a random clause. Next, we apply the amplification to the trivial verifier iteratively, until we obtain a PCPP verifier that rejects false claims with constant probability (which does not depend on n). Clearly, the number of iterations required is $O(\log n)$, and therefore the final PCPP verifier uses proofs of length $\beta^{O(\log n)} \cdot \text{poly}(n) = \text{poly}(n)$, as required.

As we mentioned before, this proof yields PCPP verifiers that run in time $\text{poly}(n)$, while we would have wanted our verifiers to be super-fast, i.e., run in time $\text{poly}(\log n)$. The reason for the inefficiency of Dinur's PCPP verifiers is that the amplification technique increases the running time of the verifier to which it is applied by at least a constant factor. Since the amplification is applied for $O(\log n)$ iterations, the resulting blow-up in the running time is at least $\text{poly}(n)$.

3.1.2.2 On Dinur and Reingold's construction of PCPPs

In order to give a construction of PCPPs with super-fast verifiers, we consider another combinatorial construction of PCPPs, which was proposed by Dinur and Reingold [DR06] prior to Dinur's proof of the PCP theorem. We refer to this construction as the "DR construction". Like Dinur's construction, the DR construction is an iterative construction. However, unlike Dinur's construction, the DR construction uses only $O(\log \log n)$ iterations. This means that if their construction can be implemented in a way such that each iteration incurs a linear blow-up to the running time of the verifiers, then the final verifiers will run in time $\text{poly} \log n$ as we desire. *Our first main technical contribution is showing that such an implementation is indeed possible.* Providing such an implementation requires developing new ideas, as well as revisiting several known techniques from the PCP literature and showing that they have super-fast implementations.

Still, the DR construction has a significant shortcoming: its verifiers use proofs that are too long; specifically, this construction uses proofs of length $n^{\text{poly} \log n}$. *Our second main technical contribution is showing how to modify the DR construction so as to have proofs of length* $\text{poly}(n)$ *while maintaining the high efficiency of the verifiers.*

3.1.2.3 Our construction vs. the DR construction

Following Dinur and Reingold, it is more convenient to describe our construction in terms of "assignment testers" (ATs). Assignment testers are PCPPs that verify that an assignment is close to a satisfying assignment of a given circuit. Any construction of ATs yields a construction of PCPs and PCPPs, and therefore our goal is to construct ATs whose running time is poly-logarithmic in the size of the given circuit.

The crux of the DR construction is a reduction that transforms an AT that acts on circuits of size k to an AT that acts on circuits of size k^c (for some constant $c > 0$). Using such a reduction, it is possible to construct an AT that works on circuits of size n by starting from an AT that works on circuits of constant size and applying the reduction for $O(\log \log n)$ times. However, the DR reduction also increases the proof length from ℓ to $\ell^{c'}$ (for some constant $c' > c$), which causes the final ATs to have proof length $n^{\text{poly} \log n}$. Moreover, the reduction runs in time that is polynomial in the given circuit, rather than poly-logarithmic. We turn to discuss the issues of improving the proof length and improving the running time separately.

The proof length A close examination of the DR reduction shows that its superfluous blow-up stems from two sources. The first source is the use of a "parallel repetition"-like error reduction technique, which yields a polynomial blow-up to the proof length. This blow-up can be easily reduced by using the more efficient amplification technique from Dinur's work.

The second source of the blow-up is the use of a particular circuit decomposition technique. The DR reduction uses a procedure that decomposes a circuit into an "equivalent" set of smaller circuits. This part of the reduction yields a blow-up that is determined by the parameters of the decomposition. The DR reduction uses a straightforward method of decomposition that incurs a polynomial blow-up. In this chapter, we present an alternative decomposition method that is based on packet-routing ideas and incurs a blow-up of only a poly-logarithmic factor, as required.

The running time For the rest of the discussion, it would be convenient to view the DR reduction as constructing a "big" AT that acts on "big" circuits from a "small" AT that acts on "small" circuits. The big AT works roughly by decomposing the given circuit to an equivalent set of smaller circuits, invoking the small AT on each of the smaller circuits, and combining the resulting residual tests in a sophisticated way. However, if we wish the big AT to run in time which is linear in the running time of the small AT, we can not afford invoking the small AT on each of the smaller circuits since the the number of those circuits is super-constant. We therefore modify the reduction so that it does not invoke the small AT on each of the smaller circuits, but rather invoke it once on the "universal circuit", and use this single invocation for testing the smaller circuits. When designed carefully, the modified reduction behaves like the original reduction, but has the desired poly-logarithmic running time.

In addition to the foregoing issue, we must also show that our decomposition method and Dinur's amplification technique have sufficiently efficient implementations. This is easily done for the decomposition. However, implementing Dinur's error reduction is non-trivial, and can be done only for PCPs that possess a certain property. The efficient implementation of Dinur's error reduction method, and of several other known PCP techniques, is an additional contribution of this chapter.

Organization of this chapter. In Section 3.2 we cover the relevant background for this chapter and give a formal statement of our main results. In Section 3.3 we give a high-level overview of this chapter. In Sections 3.4 and 3.5 we define super-fast assignment testers, which are central for this chapter, and develop tools for working with those testers. Finally, in Sections 3.6, 3.7, and 3.8, we prove the main technical results of this chapter. A more detailed description of the organization of this chapter is given at the end of the overview in Section 3.3.5.

3.2 Preliminaries and Our Main Results

3.2.1 Notational Conventions

For any $n \in \mathbb{N}$, we denote $[n] \overset{\text{def}}{=} \{1, \ldots, n\}$. For any $S \subseteq [n]$ and $x \in \{0,1\}^n$, we denote by $x_{|S}$ the projection of x to S. That is, if $S = \{i_1, \ldots, i_s\}$ for $i_1 < \ldots < i_s$, then $x_{|S} = x_{i_1} \ldots x_{i_s}$

For every functions $g, f_1, \ldots, f_m : \mathbb{N} \to \mathbb{N}$, we denote by $g = \text{poly}\,(f_1, \ldots, f_m)$ the fact that g is asymptotically bounded by some polynomial in $f_1, \ldots, f_m$. We use the notation $g = \text{poly}\log\,(f_1, \ldots, f_m)$ as an abbreviation for $g = \text{poly}\,(\log f_1, \ldots, \log f_m)$.

For any two strings $x, y \in \{0,1\}^n$, we denote by $\text{dist}\,(x, y)$ the relative Hamming distance between x and y, i.e., $\text{dist}(x, y) \overset{\text{def}}{=} \Pr_{i \in [n]}\,[x_i \neq y_i]$. For any string $x \in \{0,1\}^*$ and a set $S \subseteq \{0,1\}^*$, we denote by $\text{dist}\,(x, S)$ the relative Hamming distance between x and the nearest string of length $|x|$ in S, and we use the convention that $\text{dist}\,(x, S) = 1$ if no string of length $|x|$ exists in S. In particular, we define $\text{dist}(x, \emptyset) = 1$.

For any circuit φ, we denote by $\text{SAT}(\varphi)$ the set of satisfying assignments of φ. We define the size of φ to be the number of wires in φ.

3.2.2 PCPs

As discussed in the introduction, the focus of this chapter is on proofs that can be verified by reading a small number of bits of the proof *while running in a short time*. It is usually also important to keep track of the randomness complexity of the verifier. This leads to the following definition of PCPs.

Definition 3.2.1. Let $r, q, t : \mathbb{N} \to \mathbb{N}$. A (r, q, t)-PCP verifier V is a probabilistic oracle machine that when given input $x \in \{0,1\}^*$, runs for at most $t(|x|)$ steps, tosses at most $r(|x|)$ coins, makes at most $q(|x|)$ *non-adaptive* queries to its oracle, and outputs either "accept" or "reject". We refer to r, q, and t, as the randomness complexity, query complexity and time complexity of the verifier respectively.

Remark 3.2.2. There is a tight connection between the length of the proof that a PCP verifier uses to its randomness complexity and query complexity. To see it, note that for an (r, q, t)-PCP verifier V

and an input x, the verifier V can make at most $2^{r(|x|)} \cdot q(|x|)$ different queries, and hence the "effective proof length" of V is upper bounded by $2^{r(|x|)} \cdot q(|x|)$.

Now that we have defined the verifiers, we can define the languages for which membership can be verified.

Definition 3.2.3. Let $r, q, t : \mathbb{N} \to \mathbb{N}$, let $L \subseteq \{0,1\}^*$ and let $\rho \in (0,1]$. We say that $L \in \mathbf{PCP}_\rho [r, q, t]$ if there exists an (r, q, t)-PCP verifier V that satisfies the following requirements:

- **Completeness:** For every $x \in L$, there exists $\pi \in \{0,1\}^*$ such that $\Pr[V^\pi(x) \text{ accepts}] = 1$.
- **Soundness:** For every $x \notin L$ and for every $\pi \in \{0,1\}^*$ it holds that $\Pr[V^\pi(x) \text{ rejects}] \geq \rho$.

Remark 3.2.4. Note that Definition 3.2.3 specifies the rejection probability, i.e., the probability of false claims to be rejected. We warn that it is more common in PCP literature to specify the error probability, i.e., the probability that false claims are *accepted*.

Remark 3.2.5. Note that for any two constants $0 < \rho_1 < \rho_2 < 1$, it holds that $\mathbf{PCP}_{\rho_1}[r, q, t] = \mathbf{PCP}_{\rho_2}[O(r), O(q), O(t)]$ by a standard amplification argument. Thus, as long as we do not view constant factors as significant, we can ignore the exact constant ρ and refer to the class $\mathbf{PCP}[r, q, t]$.

Remark 3.2.6. The standard notation usually omits the running time t, and refers to the class $\mathbf{PCP}[r, q]$, which equals $\mathbf{PCP}[r, q, \text{poly}(n)]$.

The PCP theorem is usually stated as $\mathbf{NP} \subseteq \mathbf{PCP}[O(\log n), O(1)]$, but in fact, the original proof of [AS98, ALM$^+$98], when combined with earlier ideas of [BFL91], actually establishes something stronger. In order to state the full theorem, let us say that a function $f : \mathbb{N} \to \mathbb{N}$ is admissible if it can be computed in time $\text{poly}\log f(n)$ (this definition can be extended for functions f of many variables).

Theorem 3.2.7 (implicit in the PCP theorem of [BFL91, AS98, ALM$^+$98])**.** *For any admissible function $T(n) = \Omega(n)$, it holds that*

$$\mathbf{NTIME}(T(n)) \subseteq \mathbf{PCP}[O(\log T(n)), O(1), \text{poly}(n, \log T(n))]$$

In her paper [Din07], Dinur presented a combinatorial proof of the PCP theorem. However, while her proof matches the randomness and query complexity of Theorem 3.2.7, it only yields a weaker guarantee on the time complexity of the verifier:

Theorem 3.2.8 (Implicit in Dinur's PCP theorem [Din07])**.** *For any admissible function $T(n) = \Omega(n)$, it holds that*

$$\mathbf{NTIME}(T(n)) \subseteq \mathbf{PCP}[O(\log T(n)), O(1), \text{poly}(T(n))]$$

Note that due to the difference in the time complexity, Theorem 3.2.7 implies Theorem 3.1.1 (PCP theorem for **NEXP**) as a special case for $T(n) = \exp(\text{poly}(n))$, while Theorem 3.2.8 does not. *One contribution of this chapter is a combinatorial proof for Theorem 3.2.7.* This proof is actually an immediate corollary of our proof of Theorem 3.2.16 to be discussed in the next section.

3.2.3 PCPs of Proximity

3.2.3.1 The definition of PCPPs

We turn to formally define the notion of PCPs of Proximity (PCPPs). We use a definition that is more general than the one discussed in Section 3.1. In Section 3.1, we have described PCPPs as verifiers that a string x is close to being in a language $L \in \mathbf{NP}$ by reading a constant number of bits from x and from an additional proof. For example, if L is the language of graphs with a clique of size at least $\frac{n}{4}$ (where n is the number of vertices in the graph), then one can use a PCPP verifier to verify that x is close to representing such a graph. We can consider a more general example, in which we wish to verify that x is close to a graph of n vertices that contains a clique of size m, where m is a parameter given as an input to the verifier. In such a case, we would still want the PCPP verifier to read only a constant number of bits of x, but we would want to allow the verifier to read all of m, which is represented using $\log n$ bits. It therefore makes sense to think of PCPP verifiers that are given two inputs:

1. An *explicit input* that is given on their input tape, and which they are allowed to read entirely.
2. An *implicit input* to which they are given oracle access, and of which they are only allowed to read a constant number of bits.

In order to define the languages that such verifiers accept, we need to consider languages of pairs (w, x), where w is the explicit input and x is the implicit input. This motivates the following definition:

Definition 3.2.9. A pair-language is a relation $L \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $x \in \{0,1\}^*$, we denote $L(w) \overset{\text{def}}{=} \{x : (w, x) \in L\}$.

Using Definition 3.2.9, we can describe the task of PCPP verifiers as follows: Given $w, x \in \{0,1\}^*$, verify that x is close to $L(w)$ by reading all of w and a constant number of bits from x and from an additional proof. For super-fast verifiers, we would also require a running time of $\text{poly}(|w|, \log|x|)$. In the foregoing example of cliques of size k, the explicit input w will be of the form (n, m) and $L(w)$ will be the set of all graphs of n vertices that contain a clique of size m. Note that in this example w can be represented using $O(\log n)$ bits, so super-fast verifiers will indeed run in time poly-logarithmic in the size of the graph.

PCPs of Proximity were defined independently by Ben-Sasson et al. [BSGH$^+$06] and by Dinur and Reingold [DR06][1], where the latter used the term "Assignment Testers". The question of the efficiency of the verifiers is more appealing when viewed using the definition of [BSGH$^+$06], and therefore we chose to present the foregoing intuitive description of PCPPs in the spirit of [BSGH$^+$06]. Below, we present the definition of PCPPs of [BSGH$^+$05], which is in the spirit of [BSGH$^+$06] but better suits our purposes. We then state our results according to this definition.

Definition 3.2.10 (PCPP verifier, following [BSGH$^+$05])**.** Let $r, q : \mathbb{N} \to \mathbb{N}$ and let $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. An (r, q, t)-PCPP verifier is a probabilistic oracle machine that has access to two oracles, and acts has follows:

1. The machine expects to be given as an explicit input a pair (w, m), where $w \in \{0,1\}^*$ and $m \in \mathbb{N}$. The machine also expects to be given access to a string $x \in \{0,1\}^m$ in the first oracle as well as to a string $\pi \in \{0,1\}^*$ in the second oracle.

2. The machine runs for at most $t(|w|, m)$ steps, tosses at most $r(|w| + m)$ coins and makes at most $q(|w| + m)$ queries to both its oracles non-adaptively.

3. Finally, the machine outputs either "accept" or "reject".

We refer to r, q and t as the randomness complexity, query complexity and time complexity of the verifier respectively. Note that $t(n, m)$ depends both on $|w|$ and on $|x|$, while $r(n)$ and $q(n)$ depend only on their sum. The reason is that we want the time complexity to depend differently on $|w|$ and on $|x|$ (e.g., to depend polynomially on $|w|$ and poly-logarithmically on $|x|$).

[1]We mention that PCPs of Proximity are related to the previous notion holographic proofs of [BFLS91] and to the work of [Sze99], see [BSGH$^+$06] for further discussion.

For a PCPP verifier V and strings $w, x, \pi \in \{0,1\}^*$, we denote by $V^{x,\pi}(w)$ the output of V when given $(w, |x|)$ as explicit input, x as the first oracle and π as the second oracle. That is, $V^{x,\pi}(w)$ is a short hand for $V^{x,\pi}(w, |x|)$.

Definition 3.2.11 (PCPP)**.** Let $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair-language and let $\rho > 0$. We say that $L \in \mathbf{PCPP}_\rho\left[r(n), q(n), t(n,m)\right]$ if there exists an $(r(n), q(n), t(n,m))$-PCPP verifier V that satisfies the following requirements:

- **Completeness:** For every $(w, x) \in L$, there exists $\pi \in \{0,1\}^*$ such that $\Pr\left[V^{x,\pi}(w) \text{ accepts}\right] = 1$.
- **Soundness:** For every $w, x \in \{0,1\}^*$ and for every $\pi \in \{0,1\}^*$ it holds that $\Pr\left[V^{x,\pi}(w) \text{ rejects}\right] \geq \rho \cdot \operatorname{dist}(x, L(w))$.

We refer to ρ as the rejection ratio of V.

Notation 3.2.12. Similarly to the PCP case, when we do not view constant factors as significant, we will drop ρ from the notation $\mathbf{PCPP}_\rho[r,q,t]$, since for every $0 < \rho_1 < \rho_2$ it holds that $\mathbf{PCPP}_{\rho_1}[r,q,t] = \mathbf{PCPP}_{\rho_2}[O(r), O(q), O(t)]$.

We turn to discuss two important features of Definition 3.2.11.

The soundness requirement. Note that in general, the probability that V rejects a pair (w, x) must depends on the distance of x to $L(w)$. The reason is that if V is given access to some x that is very close to $x' \in L(w)$, then the probability that it queries a bit on which x and x' differ may very small. We mention that the requirement that the rejection probability would be *proportional to* $\operatorname{dist}(x, L(w))$ is a fairly strong requirement, and that PCPPs that satisfy this requirement are sometimes referred to in the literature as "Strong PCPPs". One may also consider weaker soundness requirements. However, we use the stronger requirement since we can meet it, and since it is very convenient to work with.

PCPPs versus PCPs The following corollary shows that PCPPs are in a sense a generalization of PCPs:

Corollary 3.2.13 ([BSGH+06, Proposition 2.4])**.** *Let $PL \in \mathbf{PCPP}_\rho\left[r(n), q(n), t(n,m)\right]$ be a pair-language, let $p : \mathbb{N} \to \mathbb{N}$ be such that for every $(w, x) \in L$ it holds that $|x| \leq p(n)$, and define a language $L' \overset{\text{def}}{=} \{w : \exists x \text{ s.t. } (w, x) \in PL\}$. Then it holds that $L' \in \mathbf{PCP}_\rho\left[r(n), q(n), t(n, p(n))\right]$.*

Proof. Let V be the PCPP verifier for PL. We construct a verifier V' for L' as follows. For any $w \in L'$, a proof π' that convinces V' to accept w will consist of a string x such that $(w, x) \in L'$ and a proof π that convinces V to accept (w, x). When invoked, the verifier V' simply invokes V on explicit input w, implicit input x, and proof π. The analysis of V' is trivial. ■

Remark 3.2.14. The proof of Corollary 3.2.13 is based on a different perspective on PCPPs than the one we used throughout this section. So far we have treated the implicit input x as a claim to be verified, and the explicit input w as auxiliary parameters. However, one can also view w as the claim to be verified and x as the witness for the claim w. In this perspective, the role of the PCPP verifier is to verify that x is close to being a valid witness for the claim w. While this perspective is often more useful for working with PCPPs, we feel that it is less appealing as a motivation for the study of PCPPs, and therefore did not use it in our presentation.

3.2.3.2 Constructions of PCPPs and our results

The following notation is useful for stating the constructions of PCPPs.

Notation 3.2.15. Let $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ let $PL \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair-language. We say that L is decidable in time T if there exists a Turing machine M such that when M us given as input a pair (w, x), the machine M runs in time at most $T(|x|, |w|)$, and accepts if and only if $(w, x) \in PL$,

Super-fast PCPPs were defined and explicitly constructed for the first time by [BSGH$^+$05]. However, one can obtain a simpler and weaker construction of super-fast PCPPs by combining the techniques of the earlier works of [BFL91, AS98, ALM$^+$98], their algebraic techniques can be modified and combined with earlier ideas from [BFL91] to yield the following PCPPs[2]:

Theorem 3.2.16 (PCPPs that can be obtained from [BFL91, AS98, ALM$^+$98])**.** *For any admissible function $T(n, m)$ and a pair-language PL that is decidable in time T it holds that*

$$PL \in \mathbf{PCPP}\left[O\left(\log T(n,m)\right), O(1), \operatorname{poly}\left(n, \log T(n,m)\right)\right]$$

In her work [Din07], Dinur has also given a construction of PCPPs. Her focus in this part of the work was giving PCPPs with short proofs, and in order to construct them she combined her combinatorial techniques with the previous algebraic techniques (i.e., [BSS08]). However, one can also obtain the following PCPPs using only Dinur's combinatorial techniques:

Theorem 3.2.17 (PCPPs that can be obtained from [Din07])**.** *For any admissible function $T(n, m)$ and a pair-language PL that is decidable in time T it holds that*

$$PL \in \mathbf{PCPP}\left[O\left(\log T(n,m)\right), O(1), \operatorname{poly}\left(T(n,m)\right)\right]$$

Again, it can be shown that Theorem 3.2.16 implies Theorem 3.1.2 as a special case for $T(n) = \operatorname{poly}(n)$, while Theorem 3.2.17 does not, due to the difference in the time complexity. *Our main result in this chapter is a combinatorial proof of Theorem 3.2.16.* Observe that Theorem 3.2.16 implies Theorem 3.2.7 using Corollary 3.2.13. We thus focus on proving Theorem 3.2.16.

We conclude the discussion in PCPPs by showing that Theorem 3.2.16 indeed implies a formal version of Theorem 3.1.2.

Corollary 3.2.18 (Special case of [BSGH$^+$06, Proposition 2.5])**.** *Let $L \in$ **NP**, and let PL be the pair-language $\{(\lambda, x) : x \in L\}$ (where λ is the empty string). Then $PL \in \mathbf{PCPP}[O(\log m), O(1), \operatorname{poly}\log m]$ (note that here m denotes the length of x).*

Proof sketch. The main difficulty in proving Corollary 3.2.18 is that PL may not be decidable in polynomial time (unless $\mathbf{P} = \mathbf{NP}$), and therefore we can not use Theorem 3.2.16 directly. The naive solution would be to use Theorem 3.2.16 to construct a PCPP verifier V_1 for the *efficiently decidable* pair-language

$$PL_1 = \{(\lambda, x \circ y) : y \text{ is a valid witness for the claim that } x \in L\}$$

and then construct a verifier V for PL by asking the prover to provide a witness y in the proof oracle and emulating the action of V_1 on x and y. The problem is that if x is significantly shorter than y, it might be the case that x is far from L and yet $x \circ y$ is close to $PL_1(\lambda)$. Instead, we use Theorem 3.2.16 to construct a PCPP verifier V_2 for the *efficiently decidable* pair-language

$$PL_2 = \left\{\left(\lambda, \underbrace{x \circ x \circ \ldots \circ x}_{|y|/|x|} \circ y\right) : y \text{ is a valid witness for the claim that } x \in L\right\}$$

[2]See discussion in [BSGH$^+$06, Sections 1.3 and 2.2] and in [BSGH$^+$05]. The main improvement of [BSGH$^+$05] over those PCPPs is in the proof length, which is not the focus of the current chapter.

and then construct a verifier V for PL by emulating V_2 as before. For more details, see [BSGH$^+$06, Proposition 2.5]. ■

Remark 3.2.19. We have stated Theorems 3.2.16 and 3.2.17 only for pair-languages that are decidable in *deterministic* time, but in fact, one can use them to construct PCPPs for pair-languages that are decidable in *non-deterministic* time, using the same proof idea of Corollary 3.2.18. For details, see [BSGH$^+$06, Proposition 2.5].

3.2.4 Error Correcting Codes

We review the basics of error correcting codes [MS88]. A code C is a one-to-one function from $\{0,1\}^k$ to $\{0,1\}^l$, where k and l are called the code's message length and block length, respectively. The rate of the code C is defined to be $R_C \stackrel{\text{def}}{=} \frac{k}{l}$. We say that $c \in \{0,1\}^l$ is a codeword of C if c is an image of C, i.e., if there exists $x \in \{0,1\}^k$ such that $c = C(x)$. We denote the fact that c is a codeword of C by $c \in C$. The relative distance of a code C is the minimal relative Hamming distance between two different codewords of C, and is denoted by $\delta_C \stackrel{\text{def}}{=} \min_{c_1 \neq c_2 \in C} \{\text{dist}(c_1, c_2)\}$.

3.2.5 Routing networks

In our circuit decomposition method (see Sections 3.3.2 and 3.7) we use a special kind of graphs called permutation routing networks (see, e.g., [Lei92]). In order to motivate this notion, let us think of the vertices of the graph as computers in a network, such that two computers can communicate if and only if they are connected by an edge. There are two special sets of computers in the network: the set of sources (denoted S), and the set of targets (denoted T). Each computer in S needs to send a message to some computer in T, and furthermore, each computer in T needs to receive a message from exactly one computer in S (in other words, the mapping from sources to targets is a bijection). Then, the property of the routing network says that we can route the messages in the network such that each computer in the network forwards exactly one message. Formally, we use the following definition of routing networks.

Definition 3.2.20. A routing network of order n is a graph $G = (V, E)$ with two specialized sets S and T of size n, such that the following requirement holds: For every bijection $\sigma : S \to T$, there exists a set $\mathcal{P}$ of vertex-disjoint paths in G that connect each $v \in S$ to $\sigma(v) \in T$.

Routing networks were studied extensively in the literature of distributed computing, and several constructions of efficient routing networks are known. In particular, we use the following fact on routing networks, whose requirements are satisfied by several constructions.

Fact 3.2.21 (see, e.g, [Lei92])**.** *There exists an infinite family of routing networks $\{G_n\}_{n=1}^{\infty}$, where the network G_n being of order n, and such that:*

1. *G_n has $\tilde{O}(n)$ vertices.*
2. *The in-degree and out-degree of every vertex in the network are upper bounded by a constant, say* 2.
3. *The family is* strongly explicit*: For each $n \in \mathbb{N}$, there exists a circuit ν_n of size* poly log n *that when given as input the index of a vertex v of G_n, outputs the indices of the neighbors of v via incoming edges and outgoing edges. Moreover, there exists a polynomial time algorithm that on input n outputs ν_n.*
4. *The vertices of the first layer S are indexed from* 1 *to n, and the vertices of the last layer T are indexed from $n+1$ to $2n$.*

Routing multiple messages. We now discuss a small extension of the property of routing networks which we use in Section 3.7. Suppose now that each source computer in the network needs to send at most d messages to target computers, and that each target computer needs to receive at most d messages from source computers. In such case, we can route the messages such that every computer in the network forwards at most d messages, as can be seen in the following result.

Proposition 3.2.22 (Routing of multiple messages)**.** *Let $G = (V, E)$ be a routing network of order n, let $S, T \subseteq V$ be the sets of sources and targets of G respectively, and let $d \in \mathbb{N}$. Let $\sigma \subseteq S \times T$ be a relation such that each $s \in S$ is the first element of at most d pairs in σ, and such that each $t \in T$ is the second element of at most d pairs in σ. We allow σ to be a multi-set, i.e., to contain the same element multiple times. Then, there exists a set $\mathcal{P}$ of paths in G such that the following holds:*

1. *For each $(s, t) \in \sigma$, there exists a path $p \in \mathcal{P}$ that corresponds to (s, t), whose first vertex is s and whose second vertex in t.*
2. *Every vertex of G participates in at most d paths in $\mathcal{P}$.*

Proof sketch. Without loss of generality, assume that each $s \in S$ is the first element of *exactly* d pairs in σ, and same for T. Then, we can decompose the relation σ into d disjoint permutations (see, e.g.,[Cam98, Prop. 8.1.2]). We now find vertex-disjoint paths for each of the permutations separately as in Definition 3.2.20, and take the union of all the resulting sets of paths. ■

3.3 Overview

In this section we give a high-level overview of our construction of PCPPs (which, in turn, implies the construction of PCPs). For most of this overview we focus on describing the construction itself while ignoring the issues of efficient implementation. Then, in Section 3.3.4, we describe how this construction can be implemented efficiently.

3.3.1 The structure of the construction

Our construction and the DR construction share a similar structure. In this section we describe this structure and discuss the differences between the constructions.

3.3.1.1 Assignment testers

Assignment Testers are an equivalent formulation of PCPPs that was introduced by [DR06]. Both our construction and the DR construction are more convenient to describe as constructions of assignment testers than as constructions of PCPP. We therefore start by describing the notion of assignment testers.

An assignment tester is an algorithm that is defined as follows. The assignment tester takes as an input a circuit φ of size n over a set X of Boolean variables. The output of the assignment tester is a collection of circuits $\psi_1, \ldots, \psi_R$ of size $s \ll n$ whose inputs come from the set $X \cup Y$, where Y is a set of auxiliary variables. The circuits $\psi_1, \ldots, \psi_R$ should satisfy the following requirements:

1. For any assignment x to X that satisfies φ, there exists an assignment y to Y such that the assignment $x \circ y$ satisfies all the circuits $\psi_1, \ldots, \psi_R$.
2. For any assignment x to X that is far (in Hamming distance) from any satisfying assignment to φ, and every assignment y to Y, the assignment $x \circ y$ violates at least ρ fraction of the circuits $\psi_1, \ldots, \psi_R$.

There is a direct correspondence between assignment testers and PCPPs: An assignment tester can be thought of as a PCPP that checks the claim that x is a satisfying assignment to φ. The auxiliary variables Y correspond to the proof string of the PCPP, and the circuits $\psi_1, \ldots, \psi_R$ correspond to the various tests that the verifier performs on the R possible outcomes of its random coins. In particular, the query complexity of the PCPP can be upper bounded by s. Furthermore, note that the fraction ρ corresponds to the rejection ratio of the PCPP, and we therefore refer to ρ as the rejection ratio of the assignment tester. With a slight abuse of notation, we will also say that the circuits $\psi_1, \ldots, \psi_R$ have rejection ratio ρ.

Our main technical result is a combinatorial construction of an assignment tester that has $R(n) = \text{poly}(n)$, $s(n) = O(1)$ and that runs in time $\text{poly}\log n$, which implies the desired PCPs. Note that it is impossible for an assignment tester to run in time $\text{poly}\log n$ when using the foregoing definition of assignment testers, since the assignment tester needs time of at least $\max\{n, R \cdot s\}$ only to read the input and to write the output. However, for now we ignore this problem, and in the actual proof we work with an alternative definition of assignment testers that uses implicit representations of the input and the output (see discussion in Section 3.3.4).

3.3.1.2 The iterative structure

Our construction and the DR construction are iterative constructions: The starting point of those constructions is an assignment tester for circuits of constant size[3], which is trivial to construct. Then, in each iteration, those constructions start from an assignment tester for circuits of size[3] k, and use it to construct an assignment tester for circuits of size $\approx k^{c_0}$ (for some constant $c_0 > 1$). Thus, if we wish to construct an assignment tester for circuits of size n, we use $O(\log\log n)$ iterations.

The key difference between our construction and the DR construction is in the effect of a single iteration on the number of output circuits R. In the DR construction, each iteration increases the number of output circuits from R to $R^{c'}$ for some constant $c' > c_0$ (where c_0 is the foregoing constant). Thus, after $O(\log\log n)$ iterations the final assignment testers have $n^{\text{poly}\log n}$ output circuits, which in turn imply a PCPPs that use proofs of length $n^{\text{poly}\log n}$ and have randomness complexity $\text{poly}\log n$. In contrast, in our construction a single iteration increases the number of output circuits from R to $\tilde{O}(R^{c_0})$, and thus the final assignment testers have $\text{poly}(n)$ output circuits. Such assignment testers imply a PCPPs that use proofs of length $\text{poly}(n)$ and have randomness complexity $O(\log n)$, which is our goal.

3.3.1.3 The structure of a single iteration

We proceed to describe the structure of a single iteration. For the purpose of this description, it is convenient to assume that we wish to construct an assignment tester for circuits of size n using an assignment tester for circuits of size n^γ for some constant $\gamma < 1$ (we take $\gamma = 1/c_0$, where c_0 is the constant from Section 3.3.1.2).

The general structure of an iteration. We begin with a general description of an iteration that fits both our construction and the DR construction. Suppose that we wish to construct an assignment tester A for circuits of size n, and assume that we already have a "small" assignment tester A_S that can take as input any circuit of size $n' \leq n^\gamma$, and outputs $R(n')$ output circuits of constant size. Let ρ denote the rejection ratio of A_S . When given as input a circuit φ of size n over a set of Boolean variables X, the assignment tester A proceeds in three main steps:

[3] By "assignment tester for circuits of size k" we refer to an assignment tester that can only take as an input a circuit of size at most k.

1. The assignment tester A decomposes φ, in a way to be explained in Section 3.3.2, into set of circuits $\psi_1, \ldots, \psi_{m(n)}$ of size $s(n)$ for some $m(n), s(n) < n^{\gamma}$ (to be specified later).

2. The assignment tester A combines the circuits $\psi_1, \ldots, \psi_{m(n)}$ with the assignment tester A_S in some sophisticated way that resembles the tensor product of error-correcting codes (see Section 3.3.3 for details). The result of this operation is approximately

$$R' = R\left(O\left(m(n)\right)\right) \cdot R\left(O\left(s(n)\right)\right)$$

circuits $\xi_1, \ldots \xi_{R'}$ of constant size over variables $X \cup Y \cup Z$, that have rejection ratio $\Omega\left(\rho^2\right)$.

3. The assignment tester A applies an error-reduction transformation (to be specified later) to the circuits $\xi_1, \ldots, \xi_{R'}$ obtained in Step 2 in order to increase their rejection ratio back to ρ, and outputs the resulting circuits.

Our construction versus the DR construction. Our construction differs from the DR iteration in the circuit decomposition method used in Step 1 and in the error-reduction transformation used in Step 3. We begin by discussing the latter. The DR construction uses a variant of the parallel repetition technique in order to do the error reduction. This technique incurs a polynomial blow-up in the number of output circuits of A, while in order to have the desired number of output circuits we can only afford a blow-up by a poly-logarithmic factor.

In our construction, we replace the parallel repetition technique with Dinur's amplification technique (outlined in Section 3.1.2.1), which only incurs a constant factor blow-up. This is indeed a fairly simple modification, and the reason that Dinur's technique was not used in the original DR construction is that it did not exist at that time. However, we note that in order to use Dinur's amplification in our context, we need to show that it can be implemented in a super-fast way, which was not proved in the original work of Dinur [Din07] (see further discussion in Section 3.3.4).

We turn to discuss the circuit decomposition method used in Step 1. Recall that Step 2 generates a set of $R\left(O\left(m(n)\right)\right) \cdot R\left(O\left(s(n)\right)\right)$ circuits. Thus, the choice of the functions $m(n)$ and $s(n)$ is crucial to the number of output circuits of A. In particular, it can be verified that the recurrence relation $R\left(n\right) = R\left(O\left(m(n)\right)\right) \cdot R\left(O\left(s(n)\right)\right)$ is solved to a polynomial only if the product $m\left(n\right) \cdot s\left(n\right)$ is upper bounded by approximately n. However, since the decomposition method used in Step 1 must have certain properties that are needed for Step 2, it is not trivial to find a decomposition method for a good choice of $m(n)$ and $s(n)$.

The original DR construction uses a straightforward decomposition method that decomposes a circuit of size n into $m\left(n\right) = O(n^{3\alpha})$ circuits of size $s\left(n\right) = O(n^{1-\alpha})$, where α is a constant arbitrarily close to 0. Thus, $m(n) \cdot s(n) = O(n^{1+2\alpha})$, which causes the final assignment testers of the DR construction to have $n^{\mathrm{poly}\log n}$ output circuits. Our technical contribution in this regard is devising an alternative decomposition method that decomposes a circuit of size n into $m(n) = \tilde{O}\left(\sqrt{n}\right)$ circuits of size $s(n) = \tilde{O}\left(\sqrt{n}\right)$. Thus, $m(n) \cdot s(n) = \tilde{O}\left(n\right)$, which is good enough to make the whole construction have a polynomial number of output circuits.

3.3.2 Our circuit decomposition method

In this section we describe the circuit decomposition we use in Step 1 (of Section 3.3.1.3). A circuit decomposition is an algorithm that takes as input a circuit φ over Boolean variables X and "decomposes" it to set of smaller circuits $\psi_1, \ldots, \psi_m$ over Boolean variables $X \cup Y$, such that an assignment x to X satisfies φ if and only if there exists an assignment y to Y such that $x \circ y$ satisfies all the circuits ψ_i. Note that a circuit decomposition can be viewed as an assignment tester with the trivial rejection ratio $\frac{1}{m}$. Alternatively, a circuit decomposition can be viewed as a generalization of the Cook-Levin reduction that transforms a circuit into a 3-CNF formula, by taking the smaller circuits $\psi_1, \ldots, \psi_m$ to be the clauses of the formula.

In order to be useful for the foregoing construction, a circuit decomposition must have an additional property, namely, it needs to have "matrix access": We say that a decomposition has matrix access if it is possible to arrange the variables in X and the variables of Y in two matrices such that each circuit ψ_i reads *a constant number of rows of the matrix.* The property of matrix access is reminiscent of the parallelization technique used in the PCP literature, and we refer the reader to Section 3.3.3.2 for more details regarding how it is used.

3.3.2.1 The DR decomposition

Before describing our decomposition, we briefly sketch the DR decomposition. Given a circuit φ of size n over a variables set X, they transform φ into a 3-CNF formula by adding $O(n)$ auxiliary variables Y. Next, they choose some arbitrarily small constant $\alpha > 0$, and arrange the variables in $X \cup Y$ arbitrarily in an $O(n^\alpha) \times O(n^{1-\alpha})$ matrix. Finally, they construct, for each triplet of rows of the matrix, a circuit ψ_i that verifies all the clauses that depend on variables that reside only in those three rows (relying on the fact that each clause depends on three variables). This results in $m = O(n^{3\alpha})$ circuits of size $O(n^{1-\alpha})$, as described in Section 3.3.1.3.

3.3.2.2 Our decomposition

The inefficiency of the DR decomposition results from the fact that we construct a circuit ψ_i for every possible triplet of rows, since we do not know in advance which variables will be used by each clause. Prior works in the PCP literature have encountered a similar problem in the context of efficient arithmetization of circuits, and solved the problem by embedding the circuit into a deBrujin graph using packet-routing techniques (see, e.g., [BFLS91, PS94]). While we could also use an embedding into a deBrujin graph in our context, we actually use a simpler solution, taking advantage of the fact that the requirements that we wish to satisfy are weaker. We do mention, however, that our solution is still in the spirit of "packet-routing" ideas.

We turn to sketch the way our decomposition method works. Fix a circuit φ of size n, and for simplicity assume that every gate in φ has exactly two incoming wires and two outgoing wires. The decomposition acts on φ roughly as follows:

1. For each gate g in φ, the decomposition adds an auxiliary variable k_g. Similarly, for each wire (g_1, g_2) in φ, the decomposition adds an auxiliary variable $k_{(g_1 g_2)}$. For a specific assignment x to φ and each wire (g_1, g_2) in φ, the variables k_{g_1} and $k_{(g_1,g_2)}$ are supposed to be assigned the bit that g_1 outputs when φ is invoked on x.

2. The decomposition arranges the variables in an $O(\sqrt{n}) \times O(\sqrt{n})$ matrix M such that for each gate g_1 that has outgoing wires to gates g_2 and g_3, the variables k_{g_1}, $k_{(g_1,g_2)}$, and $k_{(g_1,g_3)}$ are in the same row of M. Then, for each row of M, the decomposition outputs a circuit that checks for each such triplet of variables in the row that $k_{g_1} = k_{(g_1,g_2)} = k_{(g_1,g_3)}$.

3. The decomposition outputs additional circuits that "rearrange" the variables in a new order, by routing the variables through a routing network, while using additional auxiliary variables to represent the order of the variables at each intermediate layer of the routing network. After the routing, the variables are arranged in an $O(\sqrt{n}) \times O(\sqrt{n})$ matrix N that has the following property: For each gate g_1 of φ that has incoming wires from gates g_2 and g_3, the variables k_{g_1}, $k_{(g_2,g_1)}$, and $k_{(g_3,g_1)}$ are in the same row of N.

4. Next, for each row of N, the decomposition outputs a circuit that checks, for each gate g_1 in the row that has incoming wires from gates g_2 and g_3, that the variable k_{g_1} contains the output of g_1 when given inputs $k_{(g_2,g_1)}$ and $k_{(g_3,g_1)}$.

5. Finally, D outputs an output circuit that checks that the variable k_g that corresponds to the output gate of φ is assigned 1.

The straightforward way for implementing the routing network in Step 3 above is the following: We first take a routing network $G_{O(n)}$ of order of $O(n)$ (see Fact 3.2.21). We then identify each vertex in the sources set S of $G_{O(n)}$ with an entry of the matrix M, and each vertex of the targets set T of $G_{O(n)}$ with an entry of N. Next, we construct a bijection σ that maps each entry of M to its corresponding entry in N, and construct a set of vertex disjoint paths $\mathcal{P}$ that route that connect each vertex v of the sources set S to the vertex $\sigma(v)$ in the targets set T (see the definition of routing networks, Definition 3.2.20). Finally, we add an auxiliary variable for each vertex of $G_{O(n)}$, and for each edge (u, w) that belongs to one of the paths in $\mathcal{P}$, we output a circuit ψ_i that checks that the auxiliary variables that correspond to u and w are equal.

While the foregoing implementation of Step 3 is sound, note that it yields $\tilde{O}(n)$ output circuits of size $O(1)$ - for each vertex of $G_{O(n)}$, we have one output circuit checking equality of two variables. However, we need a decomposition that outputs $\tilde{O}(\sqrt{n})$ circuits of size $\tilde{O}(\sqrt{n})$. To this end, we modify the foregoing implementation by using the possibility to route multiple messages (Proposition 3.2.22). More specifically, we use a routing network $G_{O(\sqrt{n})}$ of order $O(\sqrt{n})$, and route $O(\sqrt{n})$ messages from each source and to each target. Here, the sources and targets of $G_{O(\sqrt{n})}$ are the rows of M and N respectively, the "messages" are again the entries of M and N, and each vertex of $G_{O(\sqrt{n})}$ participates in the routing of $O(\sqrt{n})$ entries. This method of routing indeed yields a decomposition with $\tilde{O}(\sqrt{n})$ circuits of size $\tilde{O}(\sqrt{n})$.

It remains to check that the resulting decomposition indeed has matrix access. To this end, we arrange the variables $X \cup Y$ in a $\tilde{O}(\sqrt{n}) \times O(\sqrt{n})$ matrix whose rows consist of: the rows of M; the rows of N; and a row for each vertex of $G_{O(\sqrt{n})}$, where each such row contains one variable. One can show that each output circuit of the decomposition queries a constant number of rows of this matrix, by using the fact that the degrees of the vertices of $G_{O\left(\sqrt{n}\right)}$ are bounded by a constant.

3.3.3 The tensor product lemma

In this section we outline the proof of the following lemma, which is used in Step 2 (of Section 3.3.1.3).

Lemma 3.3.1 (Tensor Product Lemma, simplified). *Let D be a circuit decomposition that when given a circuit of size n outputs $m(n)$ circuits of size $s(n)$ and that has matrix access. Let A_S be an assignment tester that takes as input circuits of size n' for any $n' \leq O\left(\max\left\{m(n), s(n)\right\}\right)$, outputs $R(n')$ circuits of size $O(1)$ and has rejection ratio ρ. Then, we can use D and A_S to construct an assignment tester A that when given as input a circuit of size n, outputs $R\left(O\left(m(n)\right)\right) \cdot R\left(O\left(s(n)\right)\right)$ circuits of size $O(1)$ and has rejection ratio $\Omega(\rho^2)$.*

The construction of the assignment tester A from A_S and D is somewhat similar to the tensor product of error correcting codes, hence the name of the lemma. We note that the proof of this lemma is implicit in [DR06], although they only proved it for their specific choice of D and A_S. We stress that the proof of [DR06] does not maintain the super-fast running time of the assignment tester, and that one of our main contributions is proving the lemma for super-fast assignment testers (see discussion in Section 3.3.4).

3.3.3.1 Warm-up: Ignoring issues of robustness

As a warm-up, we consider the following thought-experiment: Let A' be an assignment tester that has rejection ratio ρ. We say that A' is idealized if when given an input circuit φ, the tester A' rejects *every unsatisfying assignment x of φ*, and not only unsatisfying assignments *that are far from any satisfying assignment to φ*. Little more formally, we say that A' is idealized if for *every unsatisfying assignment x* of φ and every assignment y to Y, at least an ρ fraction of the output circuits of A' reject $x \circ y$. Of course, it is impossible to construct an idealized assignment tester with the parameters we desire, but for the purpose of this warm-up discussion we ignore this fact.

We now show how to prove the tensor product lemma when both A and A_S are idealized (we note that in this case, the assumption that D has matrix access is not needed). When given as input a circuit φ of size n, the assignment tester A acts as follows:

1. The assignment tester A applies the circuit decomposition D to φ, resulting in a variable set Y and in $m(n)$ circuits $\psi_1, \ldots, \psi_{m(n)}$ of size $s(n)$ over $X \cup Y$.
2. The assignment tester A applies the assignment tester A_S to each of the circuits ψ_i, each time resulting in a variables set Z_i and in $R(s(n))$ circuits $\xi_{i,1}, \ldots, \xi_{i,R(s(n))}$ of size $O(1)$ over $X \cup Y \cup Z_i$.
3. The assignment tester A constructs circuits $\eta_1, \ldots, \eta_{R(s(n))}$ of size $O(m(n))$ over $X \cup Y \cup \bigcup_i Z_i$ by defining $\eta_j \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m(n)} \xi_{i,j}$. Note that those circuits correspond to the columns of the matrix whose elements are the circuits $\xi_{i,j}$.
4. The assignment tester A applies the assignment tester A_S to each of the circuits η_j, each time resulting in a variables set W_j and in $R(O(m(n)))$ circuits $\tau_{1,j}, \ldots, \tau_{R(O(m(n))),j}$ of size $O(1)$ over $X \cup Y \cup \bigcup_i Z_i \cup W_j$.
5. Finally, A outputs the $R(O(m(n))) \cdot R(s(n))$ circuits $\tau_{1,1}, \ldots, \tau_{R(O(m(n))),R(s(n))}$ of size $O(1)$ over $X \cup Y \cup \bigcup_i Z_i \cup \bigcup_j W_j$.

Clearly, the assignment tester A has the correct number and size of output circuits. It remains to show that it has rejection ratio $\Omega(\rho^2)$. Let $Y' = Y \cup \bigcup_i Z_i \cup \bigcup_j W_j$ be the variables set of A. Fix an assignment x to X that does not satisfy φ and fix some assignment y' to Y'. Since x does not satisfy φ, there must exist some circuit ψ_i that rejects $x \circ y'$. This implies that at least an ρ fraction of the circuits $\xi_{i,1}, \ldots, \xi_{i,R(s(n))}$ reject $x \circ y'$, let us denote those circuits by $\xi_{i,j_1}, \ldots, \xi_{i,j_k}$. Now, observe that since $\xi_{i,j_1}, \ldots, \xi_{i,j_k}$ reject $x \circ y'$, the circuits $\eta_{j_1}, \ldots, \eta_{j_k}$ must reject $x \circ y'$ as well, and that $\eta_{j_1}, \ldots, \eta_{j_k}$ form ρ fraction of the circuits $\eta_1, \ldots, \eta_{R(s(n))}$. Finally, for each circuit η_{j_h} that rejects $x \circ y'$, it holds that at least ρ fraction of the circuits $\tau_{1,j_h}, \ldots, \tau_{R(O(m(n))),j_h}$ reject $x \circ y'$, and it therefore follows that at least ρ^2 fraction of the circuits $\tau_{1,1}, \ldots, \tau_{R(O(m(n))),R(s(n))}$ reject $x \circ y'$.

3.3.3.2 The actual proof

We turn to discuss the proof of the tensor product lemma for the actual definition of assignment testers, i.e., non-idealized assignment testers. In this case, the analysis of Section 3.3.3.1 breaks down. To see it, fix an assignment x to X that is far from satisfying φ and an assignment y' to Y'. As argued in Section 3.3.3.1, we are guaranteed that there exists a circuit ψ_i that is not satisfied by $x \circ y'$. However, we can no longer conclude that ρ fraction of the circuits $\xi_{i,1}, \ldots, \xi_{i,R(s(n))}$ reject $x \circ y'$: This is only guaranteed when $x \circ y'$ is far from any satisfying assignment to ψ_i, which may not be the case.

The analysis of Section 3.3.3.1 does go through, however, provided that the circuits $\psi_1, \ldots, \psi_{m(n)}$ and $\eta_1, \ldots, \eta_{R(O(s(n)))}$ have a property called "robustness" (see Section 3.5.4). The PCP literature contains few techniques for "robustizing" the output of an assignment tester, provided that the assignment tester has specific properties. In this chapter, we define and analyze a generalization of the robustization technique of [DR06], which requires the assignment tester to have the following property:

Definition 3.3.2. We say that an assignment tester A that outputs circuits of size s is has block access if the variables in $X \cup Y$ can be partitioned to blocks such that each output circuit of A reads a constant number of whole blocks.

For example, every assignment tester that has matrix access also has block access, with the blocks being the rows of the corresponding matrix. We now have the following lemma:

Lemma 3.3.3 (Robustization, simplified)**.** *Any assignment tester that has block access can be transformed into a robust one, with a linear blow-up in the number and size of output circuits and a decrease of a constant factor in the rejection ratio. Furthermore, a similar claim holds* circuit decomposition *that has block access.*

Therefore, in order to prove the tensor product lemma, it suffices to show that the circuits $\psi_1, \ldots, \psi_{m(n)}$ and $\eta_1, \ldots, \eta_{R(s(n))}$ are have block access. This is easy to do for $\psi_1, \ldots, \psi_{m(n)}$, since the decomposition D has matrix access by assumption, and thus in particular D has block access.

In order to show that the circuits $\eta_1, \ldots, \eta_{R(s(n))}$ have block access, we also need the assignment tester A_S to be oblivious. An assignment tester is said to be oblivious if for every i, the variables in $X \cup Y$ that are queried by the i-th output circuit ψ_i depend only on i, and in particular do not depend on the input circuit φ. The work of [DR06] has showed that every assignment tester can be transformed into an oblivious one with an almost-linear loss in the parameters.

We now observe that if A_S is oblivious, then the variables in $X \cup Y \cup \bigcup_i Z_i$ can be arranged in two matrices, such that each circuit η_j queries a constant number of columns of those matrices. This implies that the circuits $\eta_1, \ldots, \eta_{R(s(n))}$ have block access, by taking the blocks to be the columns of the latter matrices.

More specifically, we arrange the variables in $X \cup Y$ in the matrix M which is guaranteed by the fact that D has matrix access, and arrange the variables $\bigcup_i Z_i$ in the matrix N whose rows are the sets Z_i. Next, observe that due to the obliviousness of A_S, if we consider a single query of an output circuit $\xi_{i,j}$, then the column of M or N to which the query belongs depends only on j and not on i. Hence, corresponding queries of $\xi_{1,j} \ldots, \xi_{O(m(n)),j}$ belong to the same columns, and this implies that η_j queries only a constant number of columns of M and N, as required.

3.3.4 Efficiency issues

Finally, we explain the modifications that we make to the foregoing construction in order to implement it efficiently.

3.3.4.1 Modifying the formalism

In Section 3.3.1.1, we defined an assignment tester as an algorithm that takes as input a circuit φ of size n and outputs R circuits $\psi_1, \ldots, \psi_R$ of size s. Clearly, such an algorithm must run in time at least $\max\{n, R \cdot s\}$. However, we want our assignment testers to run in time $\operatorname{poly}\log n$, which is much smaller than both n and $R \cdot s$. We therefore have to use a different notion of assignment tester in order to obtain the desired running time.

To this end, we use succinct representations of both the input circuit and the output circuits. The key point is that in order to construct a PCPP for **NP**, we only need to run our assignment tester on circuits that are obtained from the standard reduction from **NP** to CIRCUIT-SAT, and that those circuits can be succinctly represented using $\operatorname{poly}\log n$ bits. An assignment tester can therefore be defined[4] as

[4]We mention that this definition of assignment testers can be viewed as a variant of the notion of "verifier specifications" of [BSGH$^+$05].

an algorithm that takes as input a succinct representation of a circuit φ and an index i, and outputs a succinct representation of a circuit ψ_i. Indeed, such an algorithm can run in time $\text{poly}(\log n, \log(R \cdot s))$.

Using this new definition of assignment testers adds an additional level of complexity to our construction, since we have to implement all the ingredients of our construction (and in particular, the decomposition method, the tensor product lemma, and Dinur's amplification theorem) so as to work with succinct representations.

3.3.4.2 Efficient implementation of the tensor product lemma

Working with succinct representation of circuits instead of with the circuits themselves is especially difficult in the implementation of tensor product lemma. The main difficulty comes from the need to generate succinct representations of the circuits $\eta_1, \ldots, \eta_{R(s(n))}$ (as defined in Section 3.3.3.1). Recall that those circuits are defined by $\eta_j \overset{\text{def}}{=} \bigwedge_{i=1}^{m(n)} \xi_{i,j}$. While each of the circuits $\xi_{i,j}$ has a succinct representation, this does not imply that their conjunction has a succinct representation. In particular, a naive implementation of a representation for η_j would yield a representation of size $\Omega(m(n))$, which we can not afford.

In order to solve this problem, we observe that the output circuits $\psi_1, \ldots, \psi_{m(n)}$ of D must be "similar", since they are all generated by the same super-fast decomposition. We then show that for each j, the similarity of the circuits $\psi_1, \ldots, \psi_{m(n)}$ can be translated into a similarity of the circuits $\xi_{1,j}, \ldots, \xi_{m(n),j}$, and that this similarity of $\xi_{1,j}, \ldots, \xi_{m(n),j}$ can be used to construct a succinct representation of η_j.

To be more concrete, we sketch a simplified version of our solution[5]. Consider the "universal circuit" U that is given as input a circuit ζ and a string x, and outputs $\zeta(x)$. Now, for each i, instead of invoking A_S on ψ_i, we invoke A_S on U, and whenever one of the output circuits $\xi_{i,j}$ makes a query to an input bit of U that corresponds to the circuit ζ, we hard-wiring the answer to the query to be the corresponding bit of ψ_i. The circuits $\xi_{i,j}$ that are constructed in this manner should simulate the circuits ξ_{ij} that were constructed in Section 2. The point is that for any fixed j, the circuits $\xi_{1,j}, \ldots, \xi_{m(n),j}$ are now identical to each other, up to the foregoing hard-wiring of the answers to their queries. Using this fact, and the fact that the representation of each of the circuits ψ_i is generated by the super-fast decomposition, it is easy to construct a succinct representation of the circuit η_j. We note that in the actual construction, the circuit U is not given the circuit ζ but rather an error-correcting encoding of the succinct representation of ζ, and that U takes as an input an additional proof string.

3.3.4.3 A finer analysis of Dinur's amplification theorem

In order for our assignment testers to run in time $\text{poly}\log n$, we need to make sure that all the steps taken in a single iteration incur only a constant factor blow-up in the running time. In particular, we need to show this holds for Dinur's amplification theorem, since this was not proved in [Din07].

It turns out that in order to analyze the running time of Dinur's amplification theorem, one should make additional requirements from the assignment testers. Specifically, recall that the proof of the amplification theorem works by representing the assignment testers as "constraint graphs", and by applying various transformations to those graphs. The running time of those transformations depends on the explicitness of those graphs. Thus, in order to be able to present a super-fast implementation of Dinur's amplification technique, we must make sure that the corresponding constraint graphs are strongly-explicit.

In the context of assignment testers, a strongly-explicit constraint graph corresponds to an assign-

[5]We mention that a similar technique was used in [DR06] in order to transform an assignment tester to an oblivious one.

ment tester that has a super-fast "reverse lister" (a.k.a "reverse sampler"[6]). A reverse lister for an assignment tester A is an algorithm that behaves as follows: Suppose that on input φ over variables set X, the assignment tester A outputs circuits $\psi_1, \ldots, \psi_R$ over variables set $X \cup Y$. Then, given the name of a variable $v \in X \cup Y$, the reverse lister allows retrieving the list of all circuits $\psi_{i_1}, \ldots, \psi_{i_m}$ that take v as input.

We therefore have to make sure that all the assignment testers we construct in this chapter have corresponding super-fast reverse listers, which turns out to be quite non-trivial in some of the constructions. See Section 3.5.1 for more details regarding the definition of reverse listers.

Remark 3.3.4. We note that reverse listers are also used in the proof of the tensor product lemma, and not just in the implementation of the Dinur's amplification theorem.

3.3.4.4 Increasing the representation size

Recall that our iterative construction yields assignment testers that work only for circuits of a given size, and each iteration increases the size of the circuits that can be handled. Moreover, recall that super-fast assignment testers work with succinct representations of circuits rather than the circuits themselves. Therefore, during our iterative construction, we need to make sure that the size of the succinct representations for which the assignment tester works increases along with the circuits' size.

In this chapter, we observe that the technique used by [DR06] to transform an assignment tester to an oblivious one can also be used to increase the size of the succinct representations with which the assignment tester can work, with the cost of decreasing the corresponding size of the input circuits by a related factor. This observation essentially removes the need to pay attention to the size of the succinct representations. See Section 3.5.7 for more details.

3.3.4.5 Bounding the fan-in and fan-out

Throughout this chapter, we consider circuits with unbounded fan-in and fan-out. In particular, our definition of assignment testers requires an assignment tester to take as input a circuit φ with arbitrarily large fan-in and fan-out. However, it may be easier sometimes to construct an assignment tester that can only handle input circuits φ with bounded fan-in and fan-out. Thus, we would like to reduce the construction of general assignment testers to the construction of assignment testers that can only handle circuits with bounded fan-in and fan-out. While such a reduction is trivial to do in polynomial time in the size of the circuits, it is not clear that this can be done in the super-fast settings, where we only work with succinct representations.

In this chapter, we observe that the technique of [DR06] mentioned above can also be used to transform assignment testers that can only handle bounded fan-in and fan-out into full-fledged assignment testers, which can handle arbitrary fan-in and fan-out. This observation simplifies the construction of assignment testers, and in particular simplifies our circuit decomposition method. See Section 3.5.8 for more details.

3.3.4.6 Revisiting known PCP techniques

Our construction uses known PCP techniques such as composition (see [BSGH$^+$06, DR06]) and robustization (see Lemma 3.3.3). However, when using those techniques in our construction, we have to make sure that those techniques preserve the super-fast running time of the assignment testers, as well as

[6]The term "reverse sampler" was used in the context of PCPs [BG02]. However, we feel that the term "reverse lister" is more natural in the context of assignment testers.

super-fast running time of their corresponding reverse listers (needed for the analysis of Dinur's amplification theorem). We thus present new implementations of those techniques that meet both the latter conditions. In particular, we make the following contributions:

1. **Composition**: While a super-fast implementation of the composition technique has been proposed in [BSGH$^+$06], their implementation did not preserve the running time of the corresponding reverse listers. In this chapter, we give a more sophisticated implementation that does preserve the running time of the reverse listers. See Section 3.5.4 for more details.

2. **Robustization**: As mentioned in Section 3.3.3.2, in this chapter we present a generalization of the robustization technique of [DR06]. In particular, the robustization technique of [DR06] works only for assignment testers that have block access and whose blocks are all of the same size, and contain only proof bits, while the bits of the tested assignment are read separately. Waiving some of those restrictions supports a cleaner proof of the Tensor Product lemma (Section 3.3.3).
Implementing the robustization technique for super-fast assignment testers requires the assignment tester not only to have block access, but also to have block structure that has a strongly explicit representation. We define this representation and prove a super-fast robustization theorem. See Section 3.5.6 for more details.

3. **Proof length:** We revisit the connection between the randomness complexity of a PCP and its proof length, which is more complicated in the settings of super-fast PCPs than in the common settings. In the PCP literature it is common to assume that the proof length of PCPs is bounded by an exponential function in the randomness complexity. It is not clear that this can be assumed without loss of generality in the setting of super-fast PCPs. However, we show that this assumption can indeed be made for assignment testers that have super-fast reverse listers. See Section 3.5.2 for more details.

3.3.5 Organization of the rest of this chapter

In Section 3.4, we present the definition of super-fast assignment testers and state our main construction of assignment testers. In Section 3.5, we develop few generic tools that are used in our construction, but are also of independent interest, and which were mentioned in Sections 3.3.4.4 and 3.3.4.6 above. In Section 3.6, we prove our main theorem, relying on the decomposition method and on the tensor product lemma. In Section 3.7 we present our circuit decomposition method (which was sketched in Section 3.3.2). Finally, in Section 3.8, we prove the tensor product lemma, which was sketched in Section 3.3.3.

3.4 Super-Fast Assignment Testers: Definitions and Main Theorem

Recall that our final goal in this chapter is to construct the PCPPs that were stated in Theorem 3.2.16. As discussed in Section 3.3.1.1, the work of [DR06] used a different notion of PCPPs, which they named "assignment testers". Since our construction of PCPPs is based the work of [DR06], it is more convenient for us to construct assignment testers rather than to construct the PCPPs as defined in Section 3.2.3. However, the actual definition of assignment testers used by [DR06] is not suitable for constructing super-fast PCPPs, and we therefore work with a variant of their definition. In this section, we present the definition of assignment testers with which we will work throughout this chapter. We note that our definition of assignment testers borrows ideas from the notion of "verifier specifications" of [BSGH$^+$05], and may be viewed as a variant of this notion.

This section is organized as follows. In Section 3.4.1, we review a DR-style definition of assignment testers, which does not support discussion of super-fast verifiers. Then, in Section 3.4.2, we discuss the modifications that should be made to the DR-style definition in order to support discussion of super-fast PCPPs, and present the actual definition of assignment testers with which we will work. Finally, in Section 3.4.3, we state our construction of assignment testers, and prove that this construction implies the desired construction of PCPPs.

3.4.1 DR-style assignment testers

We begin with some motivation for the notion of assignment testers. Suppose that we wish to construct a PCPP for every pair-language that can decided in polynomial time. We first observe that it suffices to construct a PCPP for the pair-language CIRCUIT-VALUE defined as follows:

$$\textsc{Circuit-Value} \overset{\text{def}}{=} \{(\varphi, x) : x \text{ is a satisfying assignment to the circuit } \varphi\}$$

To see why, suppose that a pair language PL is decided by a machine M, and consider the following reduction from PL to CIRCUIT-VALUE. Given $(w, x) \in PL$, we construct the circuit φ_w that on input x emulates $M(w, x)$ and outputs the result. Now, the reduction maps the pair $(w, x) \in PL$ to the pair $(\varphi_w, x) \in$ CIRCUIT-VALUE, and it holds that $PL(w) =$ CIRCUIT-VALUE(φ_w). Therefore, a PCPP verifier for CIRCUIT-VALUE can be used to construct a PCPP verifier for PL.

Next, consider a PCPP verifier V for CIRCUIT-VALUE. The behavior of V on a circuit φ, an assignment x, and a proof π, can be described as follows: The verifier V tosses r coins and, based on the coin tosses and on the circuit φ, chooses some "test" to be performed on x and π, where the test reads only q bits of x and π. Now, let us view the action of V differently, and assume that instead of actually performing the test, V outputs a circuit that performs the test. By running V on all 2^r possible coin tosses, we can view V as a transformation on circuits, that maps a circuit that depends on $|x|$ bits to 2^r circuits that depend on q bits. This view of the verifier V leads to the following definition.

Definition 3.4.1 (DR-style Assignment Testers). Let $R, s, \ell : \mathbb{N} \to \mathbb{N}$ and let $\rho \in (0, 1]$. An assignment tester with outputs' number $R(n)$, outputs' size $s(n)$, proof length $\ell(n)$, and rejection ratio ρ is an algorithm that satisfies the following requirements:

- **Input:** The algorithm takes as an input a circuit φ of size n over m inputs.
- **Output:** The algorithm outputs $R(n)$ circuits $\psi_1, \ldots, \psi_{R(n)}$ of size at most $s(n)$ each. The algorithm also outputs sets $Q_1, \ldots, Q_{R(n)} \subseteq [m + \ell(n)]$ such that for each $i \in [R(n)]$ the circuit ψ_i has $|Q_i|$ inputs.
- **Completeness:** For every assignment $x \in \{0,1\}^m$ that satisfies φ, there exists a string $\pi \in \{0,1\}^{\ell(n)}$ such that the following holds: For every $i \in [R(n)]$ the assignment $(x \circ \pi)_{|Q_i}$ satisfies ψ_i. We refer to π as a proof of x, or as a proof that convinces A that x satisfies φ.
- **Soundness:** For every assignment $x \in \{0,1\}^m$ and for every string $\pi \in \{0,1\}^{\ell(n)}$, the following holds: For at least $\rho \cdot \text{dist}(x, \text{SAT}(\varphi))$ fraction of the indices $i \in [R(n)]$, the assignment $(x \circ \pi)_{|Q_i}$ does not satisfy ψ_i. We refer to x as the tested assignment and to π as the proof string.

Remark 3.4.2. Note that Definition 3.4.1 does not measure the query complexity of A, which can be defined as the maximal size of a set Q_i. Needless to say, the query complexity is upper bounded by the outputs' size $s(n)$, and this bound suffices for our purposes.

Relation to Definition 3.2.11. It can be seen that the tested assignment x and the proof string π in Definition 3.4.1 correspond to the implicit input x and to the proof π in Definition 3.2.11, while the circuit φ corresponds to the explicit input. Furthermore, the rejection ratio ρ plays the same role as in Definition 3.2.11, and the outputs' number $R(n)$ is simply $2^{r(n)}$ where $r(n)$ is the randomness complexity in Definition 3.2.10.

3.4.2 Super-fast assignment testers

We turn to define assignment testers in a way that supports discussion of super-fast verification. We note that Definition 3.4.1 does not support such a discussion due to the issues discussed next.

3.4.2.1 The size of the input circuit

The first issue that prevents Definition 3.4.1 from supporting super-fast verification concerns the size of the circuit φ. Consider a pair-language PL that can be recognized in time $T(n,m)$, and suppose that we wish to verify membership in PL in time $\mathrm{poly}\,(n, \log T(n,m))$. If we apply the reduction of PL to Circuit-Value as was described in Section 3.4.1, the resulting instance of Circuit-Value will be of length at least $T(n,m)$. Thus, if we use the assignment testers of Definition 3.4.1 to verify PL, then they will run in time at least $T(n,m)$, since only reading the input circuit φ_w will take that much time. In order to solve this issue, we observe that while the circuit φ_w is of size $\mathrm{poly}\,(T(n,m))$, it has a "highly uniform" structure and can therefore be represented succinctly using only $\mathrm{poly}\log T(n,m)$ bits. Using this representation allows speeding-up the running time of the assignment testers. To be more concrete, we define representations of circuits:

Definition 3.4.3. A circuit φ^{rep} is said to be a representation of a circuit φ if it satisfies the following requirements:

1. When given as input an index of a gate g of φ, the circuit φ^{rep} outputs the the function that g computes (one of OR, AND, NOT, or one of the constants 0 and 1), and the numbers of wires going into and out of g.
2. φ^{rep} may be given as an additional input an index h of an incoming wire of g, and in such case φ^{rep} outputs the index of the gate from which the h-th incoming wire of g comes.
3. Same as Item 2, but for outgoing wires instead of incoming wires.

In addition, we require that, if the circuit φ has m inputs, then the input gates are indexed from 1 to m.

Remark 3.4.4. In order not to confuse φ^{rep} with φ, we will always refer to φ^{rep} as a "representation" and to φ as a "circuit". In particular, we will never refer to φ^{rep} as a "circuit".

Remark 3.4.5. The requirement that the indices of the input gates are the indices from 1 to m is for convenience only. Instead, we could have required the representation to allow retrieving the indices of the input gates. That is, we could have required that given an input coordinate $k \in [m]$, the representation will be able to output the index of the input gate that corresponds to k.

As noted above, we would like the representation φ^{rep} to be of size $\mathrm{poly}\log|\varphi|$. It is well-known that such a representation exists for every circuit that is obtained from applying the standard reduction of a Turing machine to a circuit:

Fact 3.4.6 (Folklore)**.** *Let $T(n,m)$ be an admissible function, and let M be a Turing machine that when given as input a pair (w,x) runs in time $T(|w|,|x|)$. Then, there exists an infinite family of circuits $\{\varphi_{n,m}\}_{n=1,m=1}^{\infty}$ of size $O\left(T(n,m)^2\right)$ such that for every $w,x \in \{0,1\}^*$ it holds that $\varphi_{|w|,|x|}(w,x) = M(w,x)$. Furthermore, there exists a constant d_{CL} such that for every $n,m \in \mathbb{N}$, the circuit $\varphi_{n,m}$ has a representation $\varphi_{n,m}^{\mathrm{rep}}$ of size at most $\log^{d_{CL}} T(n,m)$, and there exists a Turing machine that on input (n,m) outputs $\varphi_{n,m}^{\mathrm{rep}}$ in time $\log^{d_{CL}} T(n,m)$.*

We now modify the definition of assignment testers to take as an input the representation φ^{rep} instead of the circuit φ.

3.4.2.2 The size and number of the output circuits

A similar issue that should be handled is the size of the output circuits. We will sometimes be interested in assignment testers whose outputs' size $s(n)$ is larger than $\operatorname{poly}\log T(n,m)$, and therefore we can not afford to output the circuits $\psi_1,\ldots,\psi_{R(n)}$. As in the case of the input circuit, we resolve this issue by modifying the assignment testers such that that they output the representations $\psi_1^{\mathrm{rep}},\ldots,\psi_{R(n)}^{\mathrm{rep}}$ instead of the circuits $\psi_1,\ldots,\psi_{R(n)}$.

Another issue we need to deal with is that the outputs' number $R(n)$ may be larger than $\operatorname{poly}\log T(n,m)$, and therefore we can not afford to output all of the representations $\psi_1^{\mathrm{rep}},\ldots,\psi_{R(n)}^{\mathrm{rep}}$. This issue is resolved by modifying the assignment tester such that it gets as an additional input an index $i \in [R(n)]$, and such that it is only required to output the representation ψ_i^{rep}, instead of outputting all the representations $\psi_1^{\mathrm{rep}},\ldots,\psi_{R(n)}^{\mathrm{rep}}$.

3.4.2.3 The queries sets

The next issue that we need to deal with is that the queries sets $Q_1,\ldots,Q_{R(n)}$ may be larger than $\operatorname{poly}\log T(n,m)$, in which case we will not be able to output them. In order to resolve this issue, we make the following modification to the definition of assignment testers: Instead of requiring the assignment tester to output a queries set Q_i, we only require it to output the κ-th element of Q_i, where κ is given as an extra input.

For convenience, we make two more modifications to the definition of assignment testers:

1. First, instead of defining $Q_i,\ldots,Q_{R(n)}$ to be sets, we define them to be sequences. In particular, Q_i may contain the same query more than once, and its elements are ordered in some fixed order. As we shall see, making the same query more than once allows us to give different "weight" to the queries, while the possibility to choose the order of the queries will make it easier to implement some of the procedures efficiently.
   In fact, instead of treating $Q_i,\ldots,Q_{R(n)}$ as sequences, it will be more convenient to treat them as functions. That is, instead of treating Q_i as a sequence in $[m+\ell(n)]^{q_i}$, we will treat it as a function $Q_i : [q_i] \to [m+\ell(n)]$.

2. We allow $Q_1,\ldots,Q_{R(n)}$ to make dummy queries. A dummy query is not directed to any of the coordinates, and always returns the bit 0. Dummy queries will be represented by the symbol dummy. Thus, Q_i will be a function from $[q_i]$ to $[m+\ell(n)]\cup\{\mathsf{dummy}\}$. Dummy queries will make it easier to control the length of the sequences $Q_1,\ldots,Q_{R(n)}$ without complicating the implementation of the reverse listers (defined in Section 3.5.1).

The foregoing considerations lead to the following definition:

Definition 3.4.7. Let $q, n \in \mathbb{N}$ be natural numbers. A queries function is a function $Q : [q] \to [n] \cup \{\mathsf{dummy}\}$. For any string $x \in \{0,1\}^n$, we denote by $x_{|Q} \in \{0,1\}^q$ the string defined by $\left(x_{|Q}\right)_j = x_{|Q(j)}$ if $Q(j) \neq \mathsf{dummy}$ and $\left(x_{|Q}\right)_j = \mathsf{dummy}$ otherwise.

Given an assignment tester A and a circuit φ, we denote by $Q_i^{A,\varphi}$ the i-th queries function computed by A when invoked on the input circuit φ. When A and φ are clear from the context, we drop A and φ and write only Q_i.

3.4.2.4 Syntactic modifications

Except for the foregoing issues, we make two more syntactic modifications to Definition 3.4.1. Those modifications are done in order to simplify the presentation of our results, but are not essential:

1. Instead of defining an assignment tester as an *algorithm*, we define it as a *circuit*. Recall that a circuit can only handle inputs of a fixed size, rather than all input sizes. In particular, this means that the assignment testers defined below can not receive any circuit as an input, but are *defined only for circuits of a fixed size*. However, our main theorem (see Theorem 3.4.11 below) states a construction of a uniform family of assignment tester circuits, so we can still use our assignment testers for all circuit sizes.

2. An assignment tester should provide two different functionalities: Computing the representation ψ_i^{rep} of the i-th output circuit ψ_i, and computing the i-th queries function $Q_i^{A,\varphi}$, for every i. In order to support both functionalities, we think of the assignment tester as having two different modes of operation: a circuit mode, in which the assignment tester computes ψ_i^{rep}, and a query mode, in which the assignment tester computes the queries function $Q_i^{A,\varphi}$. We can implement this "two modes view" by modifying the assignment tester such that it takes an additional input bit, which determines in which mode the assignment tester is invoked.

3.4.2.5 The final definition

We are now ready to present the definition of assignment testers with which we will work throughout this chapter. In the rest of this chapter, whenever we refer to "assignment testers" we always refer to the following definition, and never to Definition 3.4.1. The following definition differs from Definition 3.4.1 only in the points discussed above.

Definition 3.4.8 (Assignment Testers, revised). An assignment tester A for circuits of size n with outputs' number R, outputs' size s, proof length ℓ, rejection ratio ρ, tester size t, input representation size n^{rep}, and output representation size s^{rep} is a circuit that satisfies the following requirements:

1. **Input and output:** The assignment tester A operates in two modes:

    a) In the circuit mode, A takes as an input a triplet $(\varphi^{\mathrm{rep}}, m, i)$, where $i \in [R]$ and φ^{rep} is a representation of a circuit φ of size at most n over m inputs. The size of φ^{rep} is required to be at most n^{rep}. The tester A then outputs a pair $(\psi_i^{\mathrm{rep}}, q_i)$, where q_i is a natural number, and ψ_i^{rep} is a representation of a circuit ψ_i of size at most s over q_i inputs. The size of ψ_i^{rep} is at most s^{rep}.

    b) In the query mode, A takes as an input a quartet $(\varphi^{\mathrm{rep}}, m, i, \kappa)$, where φ^{rep}, m and i are as in the circuit mode, and $\kappa \in [q_i]$, where q_i is as in the circuit mode. The assignment tester A outputs the κ-th query of ψ_i^{rep}, i.e., A outputs $Q_i^{A,\varphi}(\kappa)$, where $Q_i^{A,\varphi} : [q_i] \to [m+\ell] \cup \{\mathsf{dummy}\}$ is as defined in Notation 3.4.7.

2. **Completeness:** For every assignment $x \in \{0,1\}^m$ that satisfies φ there exists a string $\pi \in \{0,1\}^\ell$ such that the following holds: For every $i \in [R]$, the assignment $(x \circ \pi)_{|Q_i}$ satisfies ψ_i. We refer to π as a proof of x, or as a proof that convinces A that x satisfies φ.

3. **Soundness:** For every assignment $x \in \{0,1\}^m$, and for every string $\pi \in \{0,1\}^\ell$ the following holds: For at least $\rho \cdot \mathrm{dist}(x, \mathrm{SAT}(\varphi))$ fraction of the indices $i \in [R]$, the assignment $(x \circ \pi)_{|Q_i}$ does not satisfy ψ_i. We refer to x as the tested assignment and to π as the proof string.

4. **Size:** The size of the assignment tester A (as a circuit) is at most t.

We will sometimes refer to n as the input size of A.

Remark 3.4.9. Note that Definition 3.4.8 specifies an upper bound on the size of the assignment tester (i.e. the tester size t), while Definition 3.4.1 had no restrictions on the running time of the assignment tester.

Remark 3.4.10. Definition 3.4.8 has more parameters than we would have liked it to have. However, the most significant parameters are the circuit size n, the outputs' number R, the outputs' size s and the tester size t. The rejection ratio ρ will require only little attention throughout our construction. Furthermore, as we will see in Sections 3.5.2 and 3.5.7, we do not keep track of the proof length ℓ throughout this chapter, and the representations' sizes n^{rep} and s^{rep} are of little significance.

3.4.3 Main theorem

The rest of this chapter is devoted to proving the following theorem:

Theorem 3.4.11 (Main Theorem)**.** *There exists an infinite family of circuits $\{A_{n,n^{\mathrm{rep}}}\}_{n=1,n^{\mathrm{rep}}=1}^{\infty}$, such that $A_{n,n^{\mathrm{rep}}}$ is an assignment tester for circuits of size n with outputs' number $R(n) = \mathrm{poly}(n)$, outputs' size $s(n) = O(1)$, proof length $\ell(n) = \mathrm{poly}(n)$, rejection ratio $\rho = \Omega(1)$, tester size $t(n, n^{\mathrm{rep}}) = \mathrm{poly}(\log n, n^{\mathrm{rep}})$, input representation size n^{rep}, and output representation size $s^{\mathrm{rep}}(n, n^{\mathrm{rep}}) = O(1)$. Furthermore, there exists an algorithm that on inputs n and n^{rep}, runs in time $\mathrm{poly}(\log n, n^{\mathrm{rep}})$ and outputs $A_{n,n^{\mathrm{rep}}}$.*

We now show that the main theorem implies the desired construction of PCPPs (Theorem 3.2.16, restated below), and thus implies the desired construction of PCPs as well (Theorem 3.2.7):

Theorem (3.2.16, restated)**.** *For any admissible function $T(n,m)$ and a pair-language PL that is decidable in time T, it holds that*

$$PL \in \mathbf{PCPP}\left[O\left(\log T(n,m)\right), O(1), \mathrm{poly}\left(n, \log T(n,m)\right)\right]$$

Proof of Theorem 3.2.16 based on Theorem 3.4.11. The proof is straightforward, and consists of reducing PL to Circuit-Value as discussed in Section 3.4.1 while using the representation of the circuit instead of the circuit itself, and then applying the assignment testers of the main theorem (Theorem 3.4.11) to the resulting representation. Details follow.

Let $\ell(n)$ denote the proof length of the assignment tester of Theorem 3.4.11. Let $T(n,m)$ and PL be as in Theorem 3.2.16, and let M be the machine deciding PL in time T. We define a PCPP verifier V for PL. Let $\{\varphi_{n,m}\}_{n=1,m=1}^{\infty}$ be the infinite family of circuits that is obtained by applying Fact 3.4.6 to M, and let $\varphi_{n,m}^{\mathrm{rep}}$ be the corresponding representation of $\varphi_{n,m}$ for every $n, m \in \mathbb{N}$. Recall that for every $n, m \in \mathbb{N}$, it holds that $\varphi_{n,m}$ is of size at most $O\left(T\left(n,m\right)^2\right)$, and that $\varphi_{n,m}^{\mathrm{rep}}$ is of size at most $\log^{d_{CL}} T(n,m)$, where d_{CL} is the constant from Fact 3.4.6.

We now describe the action of V when given explicit input w, implicit input x and proof π: The verifier V begins by computing the representation $\varphi^{\text{rep}} = \varphi^{\text{rep}}_{|w|,|x|}$ of the circuit $\varphi = \varphi_{|w|,|x|}$. Next, V computes the representation φ^{rep}_w of a circuit φ_w that is obtained by hard-wiring w into the first $|w|$ inputs of φ. Observe that $PL(w) = \text{SAT}(\varphi_w)$, and it therefore remains to verify that x is close to $\text{SAT}(\varphi_w)$.

V proceeds to verify that x is close to $\text{SAT}(\varphi_w)$ as follows: V chooses $i \in [R(|\varphi_w|)]$ uniformly at random and invokes the assignment tester $A = A_{|\varphi_w|,|\varphi^{\text{rep}}_w|}$ of the main theorem on φ^{rep}_w, in order to obtain the circuit ψ_i (of size $O(1)$) and the queries function $Q_i : [q_i] \to [|x| + \ell(|\varphi_w|)] \cup \{\mathsf{dummy}\}$. Finally, V queries its oracle to obtain $(x \circ \pi)_{|Q_i}$ and checks that this string satisfies ψ_i.

We turn to analyze the parameters of V. Let $n = |w|$ and $m = |x|$. It should be clear that the verifier V has constant query complexity and rejection ratio. As for the randomness complexity, observe that V only tosses coins in order to choose i, which can be done using

$$\log R(|\varphi_w|) = \log\left(\text{poly}\left(T(n,m)^2\right)\right) = O(\log T(n,m))$$

coin tosses, since $|\varphi_w| = O\left(T(n,m)^2\right)$ (due to Fact 3.4.6). Similarly, the proof length of V is $\text{poly}(T(n,m))$.

We conclude by analyzing the time complexity of V: The representation φ^{rep} can be computed in time $\text{poly}\log T(n,m)$ by Fact 3.4.6. The representation φ^{rep}_w can be computed from φ^{rep}_w in time $\text{poly}(n, \log T(n,m))$, by hard-wiring w into φ^{rep}, and changing φ^{rep} in the straightforward way. By the main theorem, generating A and invoking it can be done in time $\text{poly}(n, \log T(n,m))$ (note that $|\varphi_w| = \text{poly}(T(n,m))$ and that $|\varphi^{\text{rep}}_w| = n + \text{poly}\log T(n,m)$). Finally, evaluating the circuit ψ_i on $(x \circ \pi)_{|Q_i}$ can be done in time $O(1)$. ■

Our main theorem can be compared to the following constructions of assignment testers of [DR06] and [Din07] (the running time stated below is implicit in those works):

Theorem 3.4.12 ([DR06, Theorem 1.2]). *Same as the main theorem, but with* $R(n) = n^{\text{poly}\log n}$ *and* $t(n, n^{\text{rep}}) = n^{\text{poly}\log n} + \text{poly}(n, n^{\text{rep}})$.

Theorem 3.4.13 ([Din07]). *Same as the main theorem, but with* $t(n, n^{\text{rep}}) = \text{poly}(n, n^{\text{rep}})$.

3.5 Tools for Constructing Assignment Testers

In this section, we develop few tools and techniques that are useful for our purposes as well as of independent interest. The section is organized as follows:

- In Section 3.5.1, we define the auxiliary notion of "reverse listers".
- In Section 3.5.2, we discuss the relation between the proof length of assignment testers to their outputs' number and size in the setting of super-fast assignment testers, and show that for the purpose of this chapter we can *do not need to keep track of the proof length* of our assignment testers.
- In Section 3.5.3, we adapt the gap amplification technique of Dinur to the setting of super-fast assignment testers by using reverse listers.
- In Section 3.5.4, we amend the known technique of PCPP composition such that it maintains the efficiency of the reverse listers of the involved assignment testers.
- In Section 3.5.5, we review useful known facts on error correcting codes in which membership can be verified efficiently. Such codes are used in Sections 3.5.6, 3.5.7, 3.5.8, and 3.8.

- In Section 3.5.6, we present a super-fast and general variant of the known robustization technique.
- In Section 3.5.7, we present a generic technique for increasing the input representation size of an assignment tester while losing only a small factor in its input circuit size. This technique reduces the significance of the input representation size and output representation size of assignment testers, and simplifies some of our proofs.
- In Section 3.5.8, we show that without loss of generality, it suffices to consider assignment testers that can only handle input circuits with bounded fan-in and fan-out.

3.5.1 Reverse Listers

In this section we define the notion of reverse lister, which is a variant of the notion of reverse sampler introduced in [BG02]. Informally, given an assignment tester A that outputs circuits $\psi_1, \ldots, \psi_R$, a reverse lister for A is a circuit that maintains for each coordinate k the list of all the circuits ψ_i that query k.

In the proof of our main result, we use reverse listers in order to analyze the effect of Dinur's amplification theorem on the tester size of assignment testers (see Section 3.5.3), as well as in the proof of the tensor product lemma (see Section 3.8). In addition, as we show in Section 3.5.2, reverse listers can be used to upper bound the proof length of assignment testers, which relieves us from the need to keep track of the proof length of our assignment testers.

Before giving the formal definition of reverse listers, we first define the notion of reverse list , which is the list that is maintained by the reverse lister:

Definition 3.5.1. Let A be an assignment tester with outputs' number R and proof length ℓ. Let φ be a a circuit over m inputs. For each $k \in [m+\ell]$, we denote the reverse list of k with respect to A and φ by

$$\mathbf{RevList}_{A,\varphi}(k) = \left\{(i,\kappa) : i \in [R], \kappa \in [q_i], Q_i^{A,\varphi}(\kappa) = k\right\}$$

We turn to present the formal definition of reverse listers. Note that similarly to an assignment tester, a reverse lister should provide a few different functionalities. Thus, as in the definition of assignment testers (Definition 3.4.8), we define the reverse lister as a circuit that has few modes of operation.

Definition 3.5.2. Let A be an assignment tester for circuits of size n, with outputs' number R, proof length ℓ, tester size t, and input representation size n^{rep}. A reverse lister RL for A is a circuit that operates in three modes:

1. **Counting mode:** When given as input a triplet $(\varphi^{\text{rep}}, m, k)$, where φ^{rep} is a representation of a circuit of size n over m inputs and $k \in [m+\ell]$, the reverse lister RL outputs $|\mathbf{RevList}_{A,\varphi}(k)|$. Here, φ^{rep} is required to be of size at most n^{rep}.
2. **Retrieval mode:** When given as input a quartet $(\varphi^{\text{rep}}, m, k, v)$ where φ^{rep}, m and k are as in the counting mode, and where $v \in [|\mathbf{RevList}_{A,\varphi}(k)|]$, the reverse lister RL outputs the v-th element of $\mathbf{RevList}_{A,\varphi}(k)$, according to some arbitrary order.
3. **Reverse retrieval mode:** When given as input a quintet $(\varphi^{\text{rep}}, m, k, i, \kappa)$ where φ^{rep}, m and k are as in the counting mode, and where $(i,\kappa) \in \mathbf{RevList}_{A,\varphi}(k)$, the reverse lister RL outputs the index v such that (i,κ) is the v-th element of $\mathbf{RevList}_{A,\varphi}(k)$, according to order used in the retrieval mode.

Remark 3.5.3. We will usually require RL to be of the same size as A, in order to avoid the need to keep track of the size of the reverse lister in addition to keeping track of the tester size.

3.5.2 On the Proof Length of Assignment Testers

In the PCP literature, it is common to assume that the proof length of a PCPP is upper bounded by $2^r \cdot q$, where r and q are the randomness and query complexity of the verifier respectively. Alternatively, in the terminology of assignment testers, the proof length of an assignment tester A is upper bounded by $R \cdot s$, where R and s are the outputs' number and size of A respectively. The justification for this upper bound is that $R \cdot s$ is the maximal number of different coordinates that the output circuits of A may query. Hence, the "effective proof length" or the number of "effective coordinates" is at most $R \cdot s$.

While in principle the assignment tester A can make queries to coordinates that are much larger than $R \cdot s$, this upper bound is indeed justified as long as we do not require our assignment testers to be super-fast. To see it, note that given an assignment tester A with outputs' number R, outputs' size s and proof length ℓ, we can always construct an equivalent assignment tester A' that has proof length $R \cdot s$ as follows: Let $\mathcal{S} \subseteq [m + \ell]$ be the set of "effective coordinates", i.e., the coordinates that are queried by at least one output circuit of A. As noted above, it holds that $|\mathcal{S}| \leq R \cdot s$. We begin the construction of A' by choosing an arbitrary one-to-one mapping ϕ of $\mathcal{S}$ into $[R \cdot s]$. Then, we define A' to be the assignment tester that emulates A while redirecting the queries of A via ϕ. It is easy to see that A' has the desired proof length, while having the same outputs' number and size as A. Furthermore, the mapping ϕ can be constructed in time that is polynomial in R, in s and in the tester size t of A. Thus, if R, s and t are polynomially bounded (as is the case in most interesting cases), this transformation can be carried out efficiently.

However, note that the foregoing transformation results in A' having tester size that is at least $R \cdot s$, since A' computes ϕ, computing ϕ requires a circuit of size $R \cdot s$ in the worst case. Thus, A' can not be super-fast. It is therefore not clear whether we can always assume that the proof length of a *super-fast* assignment tester is upper-bounded by $R \cdot s$. Nevertheless, it turns out that this bound can indeed be assumed for super-fast assignment testers that have super-fast reverse listers. The reason is that given a super-fast reverse lister, we can choose a mapping ϕ that has can be computed by a small circuit, as shown in the proof of the following result:

Theorem 3.5.4. *There exists a polynomial time procedure that satisfies the following requirements:*

- *Input:*
  1. *An assignment tester A for circuits of size n with outputs' number R, outputs' size s, proof length ℓ, rejection ratio ρ, tester size t, input representation size n^{rep} and output representation size s^{rep}.*
  2. *A reverse lister RL of size at most t.*
- *Output:*
  1. *An assignment tester A' with the same parameters as A except that its proof length is $R \cdot s$ and its tester size is $t' = O(t)$.*
  2. *A reverse lister RL' for A' of size at most t'.*

Proof sketch. As in the foregoing discussion, we denote by $\mathcal{S}$ the set of "effective coordinates" of A, i.e., the set of coordinates of the proof string that are queried by at least one output circuit of A. Theorem 3.5.4 is proved by choosing a mapping $\phi : \mathcal{S} \to [R \cdot s]$ that can be computed efficiently using the reverse lister RL. We then use the construction of A' described in the foregoing discussion, while noting that this time the resulting assignment tester A' has tester size $O(t)$. Details follow.

We define the mapping $\phi : \mathcal{S} \to [R \cdot s]$ as follows. We view the set $[R \cdot s]$ as the set of pairs $[R] \times [s]$. Let φ be a circuit of size n and let $k \in \mathcal{S}$ be an effective coordinate. Observe that this fact that k is an effective coordinate implies that the reverse list $\mathbf{RevList}_{A,\varphi}(k)$ is non-empty. Now, set $\phi(k)$ to be the first element (i,κ) of $\mathbf{RevList}_{A,\varphi}(k)$. It is easy to see that the mapping ϕ can be computed using the retrieval mode of the reverse lister RL.

We now construct A' as follows. A' has the same output circuits as A. When A' is required to compute the queries function $Q_i^{A',\varphi}(\kappa)$, it first invokes A to compute $k = Q_i^{A,\varphi}(\kappa)$, and then invokes RL to compute $\phi(k)$ and outputs it. It is not hard to verify that A' has the parameters stated in Theorem 3.5.4, and that the corresponding reverse lister RL' can be implemented by a circuit of size $O(t)$. ■

Dropping the proof length. Throughout this chapter all of our assignment testers have super-fast reverse listers. Thus, in order to simplify the presentation of this chapter, we will not keep track of the proof length of our assignment testers, and will always assume that the proof length is upper bounded by $R \cdot s$. This is acceptable, because we can always afford to apply Theorem 3.5.4 to reduce the proof length to $R \cdot S$.

3.5.3 Dinur's Amplification Theorem

In this section we review Dinur's amplification theorem [Din07]. The amplification theorem provides a transformation that increases the rejection ratio ρ of a given assignment tester A to a universal constant ρ_0 at the expense of increasing the outputs' number and tester size of A by a factor of poly $(1/\rho)$ (provided that the outputs' size of A is a constant). The original amplification theorem, given in [Din07, Section 9], was proved in a different setting than ours, and also does not analyze the effect of the amplification on the tester size. Therefore, instead of using the original theorem, we use the following variant, which can be derived from the original theorem.

Theorem 3.5.5. *There exist constants $s_0 \in \mathbb{N}$ and $\rho_0 \in (0,1)$, and a polynomial time procedure that satisfy the following requirements:*

- *Input:*
  1. *An assignment tester A for circuits of size n with outputs' number R, outputs' size s, rejection ratio ρ, tester size t, input representation size n^{rep} and output representation size at most s^{rep}.*
  2. *A reverse lister RL of size at most t.*
- *Output:*
  1. *An assignment tester A' for circuits of size n with outputs' number at most poly $\left(s, \frac{1}{\rho}\right) \cdot R$, outputs' size s_0, rejection ratio ρ_0, tester size at most $t' \stackrel{\text{def}}{=} \text{poly}\left(s, \frac{1}{\rho}\right) \cdot \left(t + \text{poly}\left(s^{\text{rep}}, \log\left(R \cdot n\right)\right)\right)$, input representation size n^{rep} and output representation size s_0.*
  2. *A reverse lister RL' for A' of size at most t'.*

Deriving Theorem 3.5.5 from the original amplification theorem of [Din07] is not very difficult, but is tedious. Since it is not the focus of this chapter, we have not included its full proof.

Theorem 3.5.5 versus the original theorem of [Din07]. As mentioned above, the main difference between the original theorem of [Din07] and our Theorem 3.5.5 is the analysis of the tester size. There are also three minor differences between the original theorem of [Din07] and Theorem 3.5.5: the original theorem of [Din07] views assignment testers as "constraint graphs"; the original theorem of [Din07] only states the effect of a single iteration of the amplification, which doubles the rejection ratio ρ, while

Theorem 3.5.5 states the effect of applying $\log \frac{\rho_0}{\rho}$ iterations, which increases the rejection ratio to ρ_0; and that the original theorem of [Din07] refers only to assignment testers with constant output size s while Theorem 3.5.5 allows an arbitrary value of s.

In order to support arbitrary values of s, we observe that the output size of any assignment tester can be reduced to from s to a constant while decreasing the rejection ratio ρ by a factor of $1/s$. This can done by composing A with an assignment tester that has input size s, constant output size, and rejection ratio $1/s$, which is trivial to construct - see Section 3.5.4 for details on composition, and Remark 3.6.2 for details on the aforementioned trivial assignment tester.

The relation of reverse listers to Dinur's amplification theorem. As mentioned in Section 3.5.1, the proof of Theorem 3.5.5 relies crucially on the assignment tester A having an efficient reverse lister. To see why the reverse lister is important, recall that in Dinur's work, an assignment tester is represented as a "constraint graph". The point is that the claim that an assignment tester A has a *super-fast* reverse lister is equivalent to the claim that A is represented by a *strongly explicit* constraint graph. Now, recall that Dinur's amplification theorem is proved by applying graph transformations to constraint graphs. The running time of those transformations depends on the explicitness of the constraint graphs, which is the reason that Theorem 3.5.5 relies on the reverse lister being super-fast.

In order to see the equivalence between the efficiency reverse listers and the explicitness of constraint graph, recall that a constraint graph is a graph whose the vertices correspond to the coordinates of the tested assignment and the proof, and whose edges correspond to the output circuits of the assignment tester. Now, given an assignment tester A that is represented as a constraint graph G, a reverse lister of A corresponds to a circuit that when given the name of a vertex v of G, lists all the edges that are adjacent to v. Thus, the reverse lister is indeed the circuit that represents G.

3.5.4 Composition of Assignment Testers

Composition of assignment testers is a technique that allows combining two assignment testers into a new assignment tester with related parameters. This technique was used, in some form, in most of the previous PCP constructions, starting from [AS98]. The interested reader is referred to [BSGH$^+$06, DR06] for a more detailed discussion of this technique. The basic idea of composition is that given an assignment tester A_1, we can decrease the size of its output circuits by applying to them a second assignment tester A_2. The assignment tester A_1 is referred to as the "outer" assignment tester, and A_2 is referred to as the "inner" assignment tester.

The composition technique can only be applied when the outer assignment tester is "robust", where "robustness" is a strengthening of the standard soundness property (Requirement 3 of Definition 3.4.8). Recall that, informally, the standard soundness property requires that, when the assignment tester is invoked on an assignment x that is far from satisfying φ, a random output circuit ψ_i rejects x with probability ρ. On the other hand, the robustness property requires that for a random output circuit ψ_i, the assignment x will be *far from satisfying* ψ_i (in expectation). Formally, robustness is defined as follows:

Definition 3.5.6. An assignment tester A is said to have (expected) robustness ρ if it satisfies the following requirement: Let φ, $\psi_1, \ldots, \psi_R$, $Q_1, \ldots, Q_R$ be as in the definition of assignment testers (Definition 3.4.8). Then, for every assignment x to φ and for every proof π it holds that

$$\mathbb{E}_{i\in[R]}\left[\operatorname{dist}\left((x\circ\pi)_{|Q_i}, \operatorname{SAT}(\psi_i)\right)\right] \geq \rho\cdot\operatorname{dist}\left(x,\operatorname{SAT}(\varphi)\right).$$

Observe that expected robustness is a strengthening of the rejection ratio parameter. That is, if an assignment tester that has (expected) robustness ρ, then it must also have rejection ratio at least ρ.

Therefore, whenever we state the robustness of an assignment tester, we avoid stating its rejection ratio. In Section 3.5.6 we show that every assignment tester whose queries have a certain structure can be modified into a robust assignment tester.

A composition theorem for super-fast assignment testers has already been proved in [BSGH$^+$05, Section 7]. However, this theorem does not preserve the efficiency of the reverse listers of the involved assignment testers. Below we state an alternative composition theorem that does preserve the efficiency reverse listers, and describe the differences between the proof of this theorem and the proofs of the previous composition theorems. We note that this theorem works under slightly more restrictive conditions than the theorem of [BSGH$^+$05], see Remark 3.5.9 below.

Theorem 3.5.7 (Composition Theorem). *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

  1. *An "outer" assignment tester A_1 for circuits of size n with outputs' number R_1, outputs' size n, robustness ρ_1, tester size t_1, input representation size n^{rep}, and output representation size s_1^{rep}. Furthermore, we require that for every input circuit φ, all the output circuits of A_1 have the same input length (though this length may vary for different input circuits φ).*
  2. *An "inner" assignment tester A_2 for circuits of size s_1 with outputs' number R_2, outputs' size s_1, rejection ratio ρ_2, tester size t_2, input representation size s_1^{rep}, and output representation size s_2^{rep}.*
  3. *Reverse listers RL_1 and RL_2 for A_1 and A_2 of sizes at most t_1 and t_2 respectively.*

- ***Output:***

  1. *An assignment tester A' for circuits of size n with outputs' number $2 \cdot R_1 \cdot R_2$, outputs' size $O(s_2)$, rejection ratio $\frac{1}{4} \cdot \rho_1 \cdot \rho_2$, tester size $t' \overset{\text{def}}{=} O(t_1 + t_2) + \text{poly}\log(n, R_1, R_2)$, input representation size n^{rep}, and output representation size $s_2^{\text{rep}} + \text{poly}\log(s_2)$.*
  2. *A reverse lister RL' for A' of size at most t'.*

Remark 3.5.8. Without loss of generality, we may assume that $n > s_1 > s_2$ (the output size of an assignment tester can always be assumed to be smaller than the input size, since otherwise the assignment tester is useless). Now, observe that the assignment tester A' improves over A_1 in its outputs' size, which is roughly $s_2 < s_1$, and improves over A_2 in its input size which is $n > s_1$. In other words, the composed assignment tester A' combines the good input size of A_1 with the good outputs' size of A_2. The cost of obtaining those improvements is that A' has larger outputs' number and tester size than both A_1 and A_2.

Remark 3.5.9. Theorem 3.5.7 works under slightly more restrictive conditions than the composition theorem of [BSGH$^+$05]. Specifically, Theorem 3.5.7 makes the following two additional requirements from A_1 and A_2:

The most restrictive requirement is that the output circuits of A_1 all have the same input length. However, we note that this requirement is less restrictive than it may seem. The reason is that in most applications of the composition technique in the literature, the robustness of the outer assignment tester is obtained by applying to it some form of "robustizing" transformation, and we can design our robustization technique (Theorem 3.5.23) such that it will guarantee that A_1 satisfies this requirement of the composition theorem.

Remark 3.5.10. We note that it is not hard to modify an assignment tester such that all its output circuits have the same input length, by taking each output circuit that has small input length and repeating its queries multiple times. However, this transformation does not seem to preserve the efficiency of the reverse lister of the assignment tester.

In the rest of this section, we describe the difference between the proof of Theorem 3.5.7 and the proofs of previous composition theorems.

In previous composition theorems, the assignment tester A' is constructed as follows: Given an input circuit φ, the assignment tester A' first applies A_1 to φ, then applies A_2 to the resulting output circuit ψ_i of A_1 and finally outputs the output of A_2. However, if A_1 and A_2 are arbitrary assignment testers, then it may be difficult to construct an efficient reverse lister for A'. For example, consider the task of counting the number of output circuits of A' that query a coordinate k in the input of φ. In order to compute this number, we need to compute the sum, over each output circuit ψ_i of A_1 that queries k, of the number of output circuits of A_2 that query the corresponding input coordinate of ψ_i. While we can use RL_1 and RL_2 to find each of the terms of this sum, the number of those terms may be too large, and we may not be able to afford to go over all of them. Thus, it is not clear how the sum of those terms can be computed.

This problem can be easily solved if we are given that, for every output circuit ψ_i of A_1 and for every coordinate k' in the input of ψ_i, the number of output circuits of A_2 that query k' is the same. However, requiring this property from A_2 is too restrictive. Instead, we require that the number of output circuits of A_2 that query k' depends only on the *input length* of ψ, and not on ψ itself or on k'. The latter requirement is sufficient for our purposes, since Theorem 3.5.7 assumes that all the output circuits of A_1 have the same input length. This calls for the following definition:

Definition 3.5.11. We say that an assignment tester A is input-uniform for every assignment length $m \in \mathbb{N}$, the size of the reverse list $\mathbf{RevList}_{A,\varphi}(k)$ is the same for all circuits φ over m inputs and all tested assignment coordinates $k \in [m]$.

Using the foregoing ideas, we can prove the following composition lemma for the case where A_2 is input-uniform. We do not provide the full proof here, but it can be found in [Mei09, App. A].

Lemma 3.5.12 (Composition Lemma for Input Uniform Inner Testers)**.** *Same as Theorem 3.5.7, with the following differences:*

1. *A_2 is required to be input-uniform.*
2. *We do not require that $R_2 \geq s_1$.*
3. *The outputs' number, output size, rejection ratio, and output representation size of A' are $R_1 \cdot R_2$, s_2, $\rho_1 \cdot \rho_2$ and s_2^{rep}, respectively.*

The more general Theorem 3.5.7 now follows as an immediate corollary of Lemma 3.5.13 and the following lemma, which shows that every assignment tester can be transformed into an input-uniform one with only a small cost:

Lemma 3.5.13. *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***
  1. *An assignment tester A for circuits of size n with outputs' number R, outputs' size s, rejection ratio ρ, tester size t, input representation size n^{rep}, and output representation size s^{rep}.*
  2. *A reverse lister RL for A of size at most t.*

- ***Output:***

  1. *An* input-uniform *assignment tester A' for circuits of size n with outputs' number $2 \cdot R$, outputs' size $O(s)$, rejection ratio $\frac{1}{4} \cdot \rho$, tester size $t' = t + \text{poly}\log(n, R)$, input representation size n^{rep}, and output representation size $s^{\text{rep}} + \text{poly}\log(n)$.*
  2. *A reverse lister RL' of size at most t'.*

Proof Sketch. We begin by defining the proof strings of A'. Let φ be a circuit of size n over m inputs, let x be a satisfying assignment of φ, and let π be the proof that convinces A that x satisfies φ. Then, the proof string that convinces A' that x satisfies φ is $\pi' = x \circ \pi$.

We turn to describe the behavior of A'. The assignment tester A' implements the following test: Suppose that A' is given a tested assignment x for a circuit φ and an alleged proof $\pi' = x' \circ \pi$. We view x and x' as partitioned to m/s blocks of size s. Now, with probability $1/2$, the tester A' invokes A to test that x' satisfies φ using the proof π, and with probability $1/2$ checks that x agrees with x' on a random block of size s. It is easy to check that A' has the required parameters, and that both A' and RL' can be implemented in size t'. ■

3.5.5 Efficiently verifiable error-correcting codes

Throughout this chapter we use the fact that there exist good error correcting codes that allow an efficient verification of the claim that for two strings w, x it holds that $w = C(x)$. Formally:

Fact 3.5.14. *There exist constants R_C and δ_C such that for every $k \in \mathbb{N}$ the following holds:*

1. *There exists a code C_k with message length k, rate R_C, relative distance δ_C and block length $l_k = k/R_C$.*
2. *There exists a circuit H_k of size $O(k)$ that takes as input strings $x \in \{0,1\}^k$ and $w \in \{0,1\}^{l_k}$, and accepts if and only if $w = C_k(x)$.*
3. *There exists an algorithm that on input k, runs in time* poly log k *and outputs a representation H_k^{rep} of H_k. In particular, H_k^{rep} is of size at most* poly log k.
4. *There exists an algorithm that on input $x \in \{0,1\}^k$, computes $C_k(x)$ in time* poly(k).

The codes of Fact 3.5.14 can be constructed from any systematic LDPC code whose parity check matrix can be represented succinctly. For example, one can use the expander codes of Spielman [Spi96], while using a strongly explicit expander for the construction.

Remark 3.5.15. We note that the codes of Fact 3.5.14 are stronger than what we need in order to prove the main results of this chapter. In particular, we could have relaxed Requirement 2 and require only that the circuit H_k will be of size $k\text{poly}\log k$, which would have made the construction of such codes much easier. However, the size of the circuit H_k affects the parameters of the robustization theorem (Theorem 3.5.23). Since this theorem may be useful for future works, we wish to prove it with the best possible parameters, and hence the use of the stronger requirements in Fact 3.5.14.

3.5.6 Robustization of Assignment Testers with Block Access

In this section, we show that every assignment tester whose queries have a certain block structure can be transformed into a robust assignment tester. This transformation will allow us to compose assignment testers in a relatively clean way.

Basically, an assignment tester has "block access" if the coordinates of the tested assignment and the proof string can be partitioned into blocks, such that each of the output circuits ψ_i of the assignment tester queries only on a small number of blocks. While it may be natural to define a block as a set of coordinates, we prefer a somewhat more involved definition that allows more slackness in the choice of the blocks, which is similar to the definition of queries functions (Definition 3.4.7). Specifically, instead of defining a block to be a subset of $[m+\ell]$, we use the following definition of a block:

Definition 3.5.16. A block of width w of $[m+\ell]$ is a function $B : [w] \to [m+\ell] \cup \{\mathsf{dummy}\}$, where dummy represents the "dummy query", which is always answered with 0. We require that every non-dummy coordinate is queried by B at most once, that is, every $k \in [m+\ell]$ has at most one preimage via B. We denote by $|B|$ the width of the block B. With some abuse of notation, we will denote by $k \in B$ the fact that k is a *non-dummy* coordinate in the image of B.

We turn to define the notion of "block access".

Definition 3.5.17. Let $Q : [q] \to [m+\ell] \cup \{\mathsf{dummy}\}$ be a queries function (as in Definition 3.4.7), and let $B_1, \ldots B_b$ be blocks of $[m+\ell]$. We say that Q queries $B_1, \ldots B_b$ if Q queries all the coordinates in the blocks consecutively, according to their order within the blocks. More formally, we say that Q queries $B_1, \ldots B_b$ if it holds that $q = \sum_{j=1}^{b} |B_j|$ and

$$\begin{aligned}
&Q(1) = B_1(1), \ldots, Q(|B_1|) = B_1(|B_1|)\\
&Q(|B_1|+1) = B_2(1), \ldots, Q(|B_1|+|B_2|) = B_2(|B_2|)\\
&\vdots\\
&Q\left(\sum_{j=1}^{b-1} |B_j| + 1\right) = B_b(1), \ldots, Q\left(\sum_{j=1}^{b} |B_j|\right) = B_b(|B_b|)
\end{aligned}$$

Definition 3.5.18. Let A be an assignment tester with outputs' number R, outputs' size s and proof length ℓ. We say that A has b-block access if for every circuit φ over m inputs there exist blocks $B_1, \ldots, B_p$ of $[m+\ell]$ whose images form a partition of $[m+\ell]$, such that the following holds: For each $i \in [R]$ there exist $B_{j_1}, \ldots, B_{j_{b'}}$ (for $b' \le b$) such that the queries function $Q_i^{A,\varphi}$ queries $B_{j_1}, \ldots, B_{j_{b'}}$.

For each $i \in [R]$, we refer to the corresponding blocks $B_{j_1}, \ldots, B_{j_{b'}}$ as the blocks queried by ψ_i (where ψ_i is the i-th output circuit obtained by applying A to φ). Note that $|B_j| \le s$ for all blocks B_j, since each circuit ψ_i has size at most s.

For technical reasons that have to do with the efficiency of the implementation, we also make the following requirements:

1. We require that every block contains either only tested assignment coordinates, or only proof coordinates (but in both cases it may contain dummy coordinates). More formally, the image of each block is either contained in $[m] \cup \{\mathsf{dummy}\}$ or in $[m+\ell] \cup \{\mathsf{dummy}\} \setminus [m]$. We refer to the first type of blocks as assignment blocks and to the second type of blocks as proof blocks.

2. We require that all the assignment blocks are of the same width.

3. We require that for every assignment block B_j, the number of non-dummy coordinates in the block image is at least $(1/3)$ fraction of the block width.

4. We require that the assignment blocks will precede the proof blocks in the order of the blocks.

Remark 3.5.19. We stress that the different proof blocks in Definition 3.5.18 may be of different widths.

Every assignment tester that has b-block access and rejection ratio ρ can be transformed into a robust assignment tester with robustness $\Omega(\rho/b)$. However, in order to make the transformation preserve the efficiency of the assignment tester, we need the assignment tester to have a block structure that can be efficiently computed. This motivates the following notion of "block access circuit", which computes the block structure efficiently.

Before giving the formal definition of block access circuits, we note that similarly to an assignment tester, a block access circuit should provide few different functionalities. Thus, as in the definitions of assignment testers and reverse listers (Definitions 3.4.8 and 3.5.2), we define the block access circuit as a circuit that has a few modes of operation.

Definition 3.5.20. Let A, R, ℓ, b, φ^{rep}, m, $\psi_1, \ldots, \psi_R$, and $B_1, \ldots, B_p$ be as in Definition 3.5.18. A block access circuit BA for A is a circuit that operates in five modes:

1. **Number of Blocks mode:** When given φ^{rep} and m, the circuit BA outputs the corresponding number of blocks p.
2. **Block to Coordinate mode:** When given φ^{rep}, m, $j \in [p]$, and $v \in [|B_j|]$, the circuit BA outputs $B_j(v)$ and$|B_j|$.
3. **Coordinate to Block mode:** When given φ^{rep}, m and $k \in [m+\ell]$, the circuit BA outputs the unique $j \in [p]$ and $v \in [|B_j|]$ such that $B_j(v) = k$.
4. **Number of Assignment Blocks mode:** When given φ^{rep} and m, the circuit BA outputs the corresponding number of assignment blocks.
5. **Circuit to Blocks mode:** When given φ^{rep}, m, and $i \in [R]$, and $v \in [b]$, the circuit BA outputs the index of the v-th block that is queried by the output circuit ψ_i, and the number b' of blocks that are queried by ψ_i.

Remark 3.5.21. As in the case of reverse listers, we will always require that the size of BA is upper bounded by the tester size of A, in order to avoid the need to introduce an extra parameter that measures the size of BA.

Remark 3.5.22. Note that if an assignment tester A has a block access circuit BA, then the queries of A are determined by BA, and especially by its Circuit to Blocks mode. In particular, the query mode of A can be implemented by simple invocations of BA.

We turn to show how to transform any assignment tester that has block access into a robust assignment tester. This transformation is a generalization of the robustization technique of [DR06] and [BSGH+06] (the latter used the term "alphabet reduction"), and is also related to the bundling technique of [BSGH+06], and to the parallelization technique of [AS98, ALM+98]. We prove the following result.

Theorem 3.5.23. *There exists a polynomial time procedure that satisfies the following requirements:*

- *Input:*

  1. *An assignment tester A for circuits of size n that* has b-block access, *outputs' number R, outputs' size s, rejection ratio ρ, tester size t, input representation size n^{rep}, and output representation size s^{rep}.*
  2. *A reverse lister RL for A of size at most t.*
  3. *A block access circuit BA for A of size at most t.*

- *Output:*
  1. *An assignment tester A' for circuits of size n with* robustness $\Omega(\rho/b)$*, outputs' number $2 \cdot R$, outputs' size $O(b \cdot s)$, tester size $t' = O(t) + b \cdot \operatorname{poly}\log(R, s, n)$, input representation size n^{rep}, and output representation size $s^{\text{rep}} + b \cdot \operatorname{poly}\log(s)$.*
  2. *A reverse lister RL' for A' of size at most t'.*

Furthermore, A' has the following property: On every input circuit φ, all the output circuits of A' have the same input length.

Remark 3.5.24. Note that the "furthermore" part is important. The reason is that this property is required in order to apply the composition theorem while using A' as the outer verifier. See the statement of the composition theorem (Theorem 3.5.7) for details.

Proof sketch. Below we sketch the proof of Theorem 3.5.23, and in particular the construction of A' and the analysis of its robustness. We do not provide here the full proof of Theorem 3.5.23, but it can be found in [Mei09, App. B].

Let φ be a circuit of size n over m inputs. We describe the action of A on φ. Let us denote by ℓ the proof length of A. Let $\psi_1, \ldots, \psi_R$ and $Q_1, \ldots, Q_R$ be the output circuits and queries functions obtained by applying A to φ, and let $B_1, \ldots, B_p$ be the blocks of A that correspond to φ.

The basic argument: The basic idea of the proof of Theorem 3.5.23 is as follows. Let x be a satisfying assignment for φ, and let π be the proof of x. We construct the proof π' that convinces A' to accept x by encoding each of the strings $(x \circ \pi)_{|B_1}, \ldots, (x \circ \pi)_{|B_m}$ via an error correcting code C with relative distance δ_C, and appending the encoding of the blocks to π (specifically, we use the codes of Section 3.5.5). Let E_j denote the encoding of the $(x \circ \pi)_{|B_j}$ via C. The assignment tester A' is constructed by modifying the circuits $\psi_1, \ldots, \psi_R$ into the following "robustized" circuits $\psi_1^{\text{rob}}, \ldots, \psi_R^{\text{rob}}$: If a circuit ψ_i queries the blocks $B_{j_1}, \ldots, B_{j_{b'}}$, then the corresponding circuit ψ'_i queries both $B_{j_1}, \ldots, B_{j_{b'}}$ and $E_{j_1}, \ldots, E_{j_{b'}}$, and verifies that $B_{j_1}, \ldots, B_{j_{b'}}$ satisfy ψ_i and that $E_{j_1}, \ldots, E_{j_{b'}}$ are indeed the correct encodings of $(x \circ \pi)_{|B_{j_1}}, \ldots, (x \circ \pi)_{|B_{j_{b'}}}$ respectively.

To see why A' should be robust, consider an assignment x that is ε-far from $\text{SAT}(\varphi)$ and some proof string π'. As a warm-up, assume that π' consists of a proof π for A, and of the correct encoding E_j of $(x \circ \pi)_{|B_j}$ via C for each block B_j. Furthermore assume that all the blocks $B_1, \ldots, B_p$ are of the same width. By the rejection ratio of A, at least $\rho \cdot \varepsilon$ fraction of the output circuits ψ_i of A reject $x \circ \pi$. We show that for each output circuit ψ_i that rejects $(x \circ \pi)_{|Q_i}$, it holds that $x \circ \pi'$ is $\Omega(1/b)$-far from satisfying ψ_i^{rob}. This will imply that for a random $i \in [R]$ it holds that $x \circ \pi'$ is $\Omega(\rho/b) \cdot \varepsilon$ far from satisfying ψ_i^{rob} (in expectation), and will therefore imply the robustness of A'.

Fix an output circuit ψ_i of A that rejects $x \circ \pi$. Then, there exists a coordinate $k \in [m + \ell]$ that is queries by k and whose value needs to flipped in order to make ψ_i accept. Let B_j denote the block to which k belongs. Now, observe that in order to make ψ_i^{rob} accept x and π', at least δ_C fraction of the bits of the *encoding* E_j need to be changed. Moreover, the encoding E_j forms at least $\Omega(1/b)$-fraction of the input of ψ_i^{rob}, since by our assumption all the blocks are of the same width. Thus, it holds that the input of ψ_i^{rob} is $\Omega(\delta_C/b)$-far from $\text{SAT}(\psi_i^{\text{rob}})$, and the required robustness follows (note that δ_C is a universal constant, and hence $\Omega(\delta_C/b) = \Omega(1/b)$). We turn to removing the warm-up assumptions.

Dealing with multiple block widths: We first remove the assumption that the blocks $B_1, \ldots, B_p$ are of the same width. Recall that we used this assumption in order to argue that the encoding E_j forms $\Omega(1/b)$-fraction of the input of ψ_i^{rob}. If the blocks are not of the same width, then the block B_j may have a very small width, in which case E_j will only constitute a small part of the input of ψ_i^{rob}. We resolve this issue by making the circuit ψ_i^{rob} query the blocks of small width several times, so those blocks form a significant portion of its input. This solution uses the convention that an output circuit of an assignment tester may query the same coordinate more than once (see discussion in Section 3.4.2.3), so the circuit ψ_i^{rob} is allowed to query the same block several times.

Dealing with inconsistencies between x and the E_j's: It remains to remove the assumption that the for each j, the string E_j is the correct encoding of $(x \circ \pi)_{|B_j}$. Note that the proof π' may not meet this condition. In such a case, the analysis of the basic argument breaks down: Even if we know that a circuit ψ_i rejects x and π, and that some bits of $(x \circ \pi)_{|B_j}$ need to be flipped to make ψ_i accept, one would still may not need to change the string E_j in order to make ψ_i accept, since E_j may be the encoding of $(x \circ \pi)_{|B_j}$ *after* flipping these bits. Thus, even though ψ_i rejects $x \circ \pi'$ in this case, the string $x \circ \pi'$ may still be close to satisfying it.

In order to resolve this issue, we consider the assignment x^{dec} and the proof π^{dec} that are obtained by decoding each string E_j in π' to the nearest legal codeword of C_k. If x^{dec} is far from SAT(φ) then the basic argument can be used as before, by replacing x with x^{dec} and π with π^{dec}. It thus remains to deal with the case that x^{dec} is close to SAT(φ).

Suppose that x^{dec} is close to SAT(φ). Note that this implies that x and x^{dec} are far from each other, since by assumption x is far from SAT(φ). We can thus detect the error in this case by checking consistency between x and x^{dec}. To this end, we modify A' to output additional "consistency" circuits $\psi_1^{\text{con}}, \ldots, \psi_R^{\text{con}}$. Each consistency circuit ψ_i^{con} queries some assignment block B_j and its purported encoding E_j, and checks that E_j is indeed the correct encoding of E_j.

We can now make the following argument: If x^{dec} is far from SAT(φ), then the foregoing basic argument shows that many of the circuits ψ_i^{rob} are far from being satisfied by $x \circ \pi'$. On the other hand, if x^{dec} is close to SAT(φ), then x is far from x^{dec} and thus many of the circuits ψ_i^{con} are far from being satisfied by $x \circ \pi'$. This establishes the robustness of A'.

We note that in the second part of the foregoing argument (i.e., when x^{dec} is close to SAT(φ)), one should be careful about a certain points: In order for the argument to hold, we need to show that if x is far from x^{dec} then, for a random assignment block B_j it holds that $x_{|B_j}$ is far from $x^{\text{dec}}_{|B_j}$. To this end, we construct the consistency circuits $\psi_1^{\text{con}}, \ldots, \psi_R^{\text{con}}$ such that each coordinate of the tested assignment is checked by roughly the same number of consistency circuits. More specifically, the consistency circuits $\psi_1^{\text{con}}, \ldots, \psi_R^{\text{con}}$ are constructed such that each assignment block is queried by roughly the same number of consistency circuits. By combining this with the assumptions of Definition 3.5.18 that all assignments blocks are of the same width, and that in each block the fraction of assignment coordinates is at least one third of the width, it follows that each coordinate of the tested assignment is checked by roughly the same number of consistency circuits. ■

3.5.7 Increasing the Representation Size, and Universal Circuits

In this section we present a technique for increasing the input representation size of an assignment tester. Along the way, we construct "universal circuits" (Section 3.5.7.2), which will also be used in the proof of the tensor product lemma (in Section 3.8)

One of the more cumbersome features of our definition of super-fast assignment testers is the need to keep track of both the input circuit size and the input representation size, rather than tracking only the input circuit size. The same holds for the outputs' size and the output representation size. In this section we discuss a result which shows that the input representation size of an assignment tester can always be made as large as we want, while paying a reasonable cost in the input circuit size. This fact has two useful implications:

1. The input representation size of an assignment tester is of little importance, since it can be made as large as needed.

2. The output representation size of an assignment tester is of little importance. The reason is that the output representation size is relevant mostly for the purposes of composition, i.e., in order to ensure that the output representations of the outer tester are not larger than the input representation size of the inner tester. Now, since the input representation size of the inner tester can always be made as large as the output representation size of the outer tester, the output representation size loses its significance as well.

The formal result is as follows.

Lemma 3.5.25 (Input Representation Lemma)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

  1. *An assignment tester A for circuits of size n that has outputs' number R, outputs' size s, rejection ratio ρ, tester size t, input representation size $n^{\text{rep}} \stackrel{\text{def}}{=} \text{poly}\log(n)$ and output representation size s^{rep}.*
  2. *A reverse lister RL for A of size at most t.*
  3. *A number $n^{\text{rep}\prime}$ that is represented in unary, such that $n^{\text{rep}\prime} \geq n^{\text{rep}}$.*

- ***Output:***

  1. *An assignment tester A' for circuits of size $n' = n/\left(n^{\text{rep}\prime} \cdot \text{poly}\log(n)\right)$ that has input representation size $n^{\text{rep}\prime}$, outputs' number R, outputs' size $s + O(1)$, rejection ratio $\rho' = \Omega(\rho)$, tester size $t' = O(t) + \text{poly}\left(\log n, n^{\text{rep}}\right)$, and output representation size $s^{\text{rep}} + \text{poly}\log(s)$.*
  2. *An reverse lister RL' of size at most t'.*

Note that the procedure stated in Lemma 3.5.25 indeed increases the input representation size of the assignment tester from n^{rep} to $n^{\text{rep}\prime}$, but decreases the input circuit size by a factor of $n^{\text{rep}\prime} \cdot \text{poly}\log(n)$.

In Section 3.5.7.1 below, we sketch the proof of Lemma 3.5.25. Then, in Section 3.5.7.2 we define and construct universal circuits, which are a central gadget of the proof and are also used in the proof of the tensor product lemma in Section 3.8. We do not provide a the full proof of Lemma 3.5.25, since it is straightforward (given the sketch below) and is tedious. However, a similar technique is used in the implementation of the tensor product lemma (Section 3.8).

Remark 3.5.26. We note that the construction of the assignment tester A' of Lemma 3.5.25 is not a new one, and in particular, [DR06] used a similar construction in order to transform assignment testers to oblivious ones. The novelty of this chapter in this context is the observation that this construction can also be used to increase the input representation size of an assignment tester.

3.5.7.1 Proof overview

The basic idea of how the input representation size can be increased is as follows. Suppose we have the following objects:

1. A circuit φ of size n' that has a representation φ^{rep} of size $n^{\text{rep}\prime}$.

2. An assignment tester A that has input circuit size $n \geq n' \cdot n^{\text{rep}\prime} \cdot \text{poly}\log(n')$ and input representation size $n^{\text{rep}} \geq \text{poly}\log(n')$.

Suppose now that we wish to invoke A on φ and on tested assignment x. The problem is that the input representation size of A may be much smaller than the size of φ^{rep}. Thus, we can not apply A directly to φ. Instead, we take the following indirect approach.

Consider a "universal circuit" U that takes as an input a representation ζ^{rep} of a circuit ζ and an input y for ζ, and outputs $\zeta(y)$ (where ζ^{rep} and ζ are of sizes $n^{\text{rep}\prime}$ and n' respectively). As we will see below (in Section 3.5.7.2), a variant of such a universal circuit can be implemented in size $n' \cdot n^{\text{rep}\prime} \cdot \text{poly}\log(n')$, and has a representation of size $\text{poly}\log(n')$. Now, we construct an assignment tester A' with input circuit size n' and input representation size as follows. When A' is invoked on input φ^{rep} and on tested assignment x, it emulates the invocation of the assignment tester A on input circuit U and on tested assignment $(\varphi^{\text{rep}}, x)$. Hopefully, this invocation of A on U is equivalent to the action of A on φ, even though A is not applied to φ directly. The emulation of A is done by invoking A on U, obtaining an output circuit ψ_i, then fixing the input gates of ψ_i that correspond to bits of ζ^{rep} to the corresponding values in φ^{rep}, and finally outputting the resulting circuit.

This construction of A' essentially emulates the action of A on the circuit φ, even though the input representation size of A may be much smaller than the size of φ^{rep}. Thus, we effectively increase the input representation size of A. While this construction almost works, there are few issues that need to be resolved, to be discussed next.

Using a weaker definition of U. The first issue is that we do not know how to implement the foregoing definition of U in size $n' \cdot n^{\text{rep}\prime} \cdot \text{poly}\log(n')$. Thus, we use a weaker definition that still suits our purposes - instead of requiring U to *compute* $\zeta(y)$ (when given as input the pair (ζ^{rep}, y)), we only require U to *verify* that ζ accepts y, and to that end allow U to use an "auxiliary witness". Note that this weaker definition of U is still useful, since assignment testers too are only required to verify that the input circuit accepts, and are allowed to use an auxiliary proof.

More specifically, we modify U such that it takes as input a tuple $(\zeta^{\text{rep}}, y, z)$ where z is an auxiliary string, and act as follows:

1. If y is a satisfying assignment of ξ, then U accepts $(\zeta^{\text{rep}}, y, z)$ for some string z.

2. If y is not a satisfying assignment of ξ, then U rejects $(\zeta^{\text{rep}}, y, z)$ for every string z.

We now modify the definition of A' as follows: On input representation φ^{rep}, tested assignment x and proof string $z \circ \pi$, the assignment tester A' emulates the invocation of A on input circuit U, on tested assignment $(\varphi^{\text{rep}}, x, z)$ and on proof string π.

Encoding φ^{rep} via an error correcting code. We turn to describe the second issue that should be resolved. Invoking A on U and on tested assignment $(\varphi^{\text{rep}}, x, z)$ does not verify that x is close to a satisfying assignment of φ, but rather that the whole triplet $(\varphi^{\text{rep}}, x, z)$ is close to a satisfying assignment of U. In particular, it could be that x is far from any satisfying assignment of φ, but φ^{rep} is close to a representation of a circuit φ' that is satisfied by x. In this case, the triplet $(\varphi^{\text{rep}}, x, z)$ is close to the satisfying assignment $(\varphi^{\text{rep}\prime}, x, z)$of U (where $\varphi^{\text{rep}\prime}$ is the representation of φ') even though x is far from any satisfying assignment of φ.

We resolve this issue by constructing an augmented universal circuit $\hat{U}$ as follows. The input of $\hat{U}$ consists of the inputs ζ^{rep}, y and z of U, and in addition, of a string c that is expected to be the encoding of ζ^{rep} via the error correcting codes of Section 3.5.5. The circuit $\hat{U}$ will now accept if and only if U accepts $(\zeta^{\text{rep}}, y, z)$, *and in addition that* c is the correct encoding of ζ^{rep}. The point is that due to the distance property of the error correcting code, the string c can not be close to encodings of two distinct representations at the same time. Therefore, if c is indeed the correct encoding of φ^{rep}, then except for an issue to be discussed next, we expect that the tuple $(\varphi^{\text{rep}}, c, x, z)$ to be close to a satisfying assignment of $\hat{U}$ if and only if x is close to a satisfying assignment of φ. Now, A' will emulate the invocation of A on input circuit $\hat{U}$ and on tested assignment $(\varphi^{\text{rep}}, c, x, z)$.

Reweighing c, x and z. The third issue is that if the length of x is very small compared to the length of the tuple $(\varphi^{\text{rep}}, c, x, z)$, then it could be the case that x is very far from a satisfying assignment to φ, but the tuple $(\varphi^{\text{rep}}, x, z)$ is close to a satisfying assignment of $\hat{U}$. A similar consideration applies to the length of c. In order to resolve this issue, we modify the input of $\hat{U}$ such that it contains many copies of x and of c, thereby increasing the "weight" of x and c within the input of $\hat{U}$. The resulting circuit $\hat{U}$ will reject if the alleged copies of x and c are not equal to one another.

Emulating the invocation of A. Finally, we elaborate a little more on how the emulation of the invocation of A is performed. Recall that we wish to emulate the invocation of A on $\hat{U}$ and on a tested assignment that consists of φ^{rep}, of multiple copies of the encoding c of φ^{rep}, of multiple copies of x, and of the witness z (where z is provided in the proof string of A'). We perform the emulation by redirecting the queries functions of A on $\hat{U}$ as follows:

1. Whenever an output circuit of A queries a coordinate of one of the multiple copies of x in the tested assignment of A, the assignment tester A' redirects the query to the corresponding coordinate of the unique copy of x in the tested assignment of A'.

2. Whenever an output circuit of A queries a coordinate of z in the tested assignment of A, the assignment tester A' redirects the query to the corresponding coordinate of z in the proof string of A'.

3. The last case, where an output circuit of A queries a coordinate of φ^{rep}or of c in the tested assignment of A, is slightly more complicated. The key point is that φ^{rep} and c are not given in the tested assignment or proof string of A', but are rather computed by A' directly. Thus, instead of redirecting the query to a coordinate of the tested assignment or proof string of A', we would like to force the query to be answered with a known bit.
   To this end, we require the proof string of A' to contain two additional special coordinates, which should contain 0 and 1. Now, whenever we would like to force the answer to a query to be 0 or 1, we redirect the query to corresponding special coordinate. In order to force the special coordinates to contain 0 and 1, we modify the output circuits of A' such that each output circuit of A' queries the special coordinates and checks that they are assigned the correct values.

Wrapping all up. We conclude by reviewing the final construction of A'. When A' is invoked in circuit mode on a representation φ^{rep} of an input circuit φ, and on index $i \in [R]$, the assignment tester A' acts as follows:

1. A' begins by invoking A to compute the representation of ψ_i, the i-th output circuit of A when invoked on the representation $\hat{U}_{\text{rep}}$.

2. A' computes the encoding c of φ^{rep}via the error correcting code of Section 3.5.5.

3. A' outputs the representation of a circuit ψ'_i, which emulates ψ_i, and in addition queries the two special coordinates and verifies that they are assigned 0 and 1 as required.

4. A' computes the queries function $Q_i^{A',\varphi}$of ψ'_i by redirecting the queries function $Q_i^{A,\hat{U}}$of ψ_i as explained above.

3.5.7.2 Universal circuits

In this section we describe how to construct the universal circuits discussed in Section 3.5.7.1. This construction is used not only in this section, but also in Section 3.5.7.2 and in the proof of the tensor product lemma in Section 3.8. We begin by proving the following result:

Lemma 3.5.27. *There exists a polynomial time procedure that when given as input numbers n, $m \leq n$, and n^{rep}, outputs a representation $U^{\mathrm{rep}} = U^{\mathrm{rep}}_{n,n^{\mathrm{rep}},m}$ (of size $\mathrm{poly}\log(n)$) of a circuit $U = U_{n,n^{\mathrm{rep}},m}$ (of size $n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log(n)$) that satisfies the following requirements:*

1. *The circuit U takes as input a representation ζ^{rep} (of size at most n^{rep}) of a circuit ζ (of size at most n) over m inputs, an input $y \in \{0,1\}^m$ to ζ, and an additional string z of length $O(n)$.*

2. *If y is a satisfying assignment of ζ, then U accepts $(\zeta^{\mathrm{rep}}, y, z)$ for some string z. We refer to z as the witness that convinces U that y satisfies ζ.*

3. *If y is not a satisfying assignment of ξ, then U rejects $(\zeta^{\mathrm{rep}}, y, z)$ for every string z.*

Proof Sketch. The construction of U is similar to the circuit decomposition method described in Sections 3.3.2 and 3.7, but is somewhat simpler. We first present a construction of a circuit U of size $n \cdot \mathrm{poly}(n^{\mathrm{rep}}, \log n)$, and then explain how modify the construction to yield a circuit U of size $n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log(n)$.

In order to construct a circuit U of size $n \cdot \mathrm{poly}(n^{\mathrm{rep}}, \log n)$, we view the auxiliary string z as a sequence of variables, where for each gate g and each wire w there are corresponding variables k_g and k_w in z. The variable k_g (respectively, k_w) is expected to be assigned the value that is output by g (respectively, carried by w) when ζ is invoked on input y. The variables are expected to be arranged in z such that each gate variable k_g is followed by the variables that correspond to the outgoing wires of g. That is, z should begin with the variable which corresponds to the first gate, followed by all the variables that correspond to the outgoing wires of that gate. The next variables in z the variable which corresponds to the second gate, again followed by all the variables that correspond to the outgoing wires of that gate, etc. The circuit U acts as follows:

1. The circuit U begins by checking that for each gate g and its outgoing wires $w_1, \ldots, w_d$ (where $d \leq 2$), it holds that $k_g = k_{w_1} = \ldots = k_{w_d}$, and rejects if one of the checks fails. Clearly, this can be done in size $O(n)$.

2. Then, the circuit U rearranges the variables in z such that each gate variable k_g is followed by the variables that correspond to the incoming wires of g rather than outgoing wires of g. That is, the new arrangement should begin with the variable which corresponds to the first gate, followed by all the variables that correspond to the *incoming* wires of that gate, and then the same for the second gate, etc. This rearrangement can be computed in size $n \cdot \mathrm{poly}(n^{\mathrm{rep}}, \log n)$ by implementing a standard sorting algorithm, where the $\mathrm{poly}(n^{\mathrm{rep}})$ factor is due to the need to invoke ζ^{rep} in order to determine the wires that are connected to each gate in ζ.

3. Finally, the circuit U checks for each gate g and its incoming wires $w_1, \ldots, w_d$ (where $d \leq 2$), that k_g is assigned the output of the gate g when given as input the values of $k_{w_1}, \ldots, k_{w_d}$. Clearly, this can be done in size $n \cdot \mathrm{poly}(n^{\mathrm{rep}})$, where the $\mathrm{poly}(n^{\mathrm{rep}})$ factor is due to the need to invoke ζ^{rep} in order to find the Boolean function computed by each gate.

It should be clear that U is of size $n \cdot \mathrm{poly}(n^{\mathrm{rep}}, \log n)$. In order to reduce the size of U to $n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log n$, note that the $\mathrm{poly}(n^{\mathrm{rep}})$ factor in the foregoing construction is since each evaluation of the representation ζ^{rep} requires a circuit of size $\mathrm{poly}(n^{\mathrm{rep}})$. Now, the crucial observation is that the circuit U does not need to evaluate ζ^{rep} by itself. Instead, we can require the auxiliary string z to contain the result of the evaluation of ζ^{rep}, and then we require U only to verify the correctness of this result, again using the auxiliary witness z . The verification of the computation of ζ^{rep} can be done in the same way the verification of the computation of ζ is done by U above, using a circuit of size $\tilde{O}(n^{\text{rep}})$. After using this efficiency improvement, we get an implementation of U of size $n \cdot n^{\text{rep}} \cdot \text{poly}\log(n)$.

It is easy to verify that U has a representation of size $\text{poly}\log(n)$, and that this representation can be constructed in polynomial time. We mention that in the foregoing calculations we used the fact that without loss of generality it holds that $n^{\text{rep}} \leq \text{poly}(n)$ and hence $\text{poly}\log(n^{\text{rep}}) = \text{poly}\log(n)$. The assumption that $n^{\text{rep}} \leq \text{poly}(n)$ can be made since every circuit of size n has a trivial representation of size $\text{poly}(n)$, and hence there is no need to consider larger representations. ■

We turn to state the construction of the augmented circuit $\hat{U}$ that was described in the overview. The following result is a direct corollary of Lemma 3.5.27:

Corollary 3.5.28. *There exists a polynomial time procedure that when given as input the numbers n, $m \leq n$, and n^{rep}, outputs a representation $\hat{U}^{\text{rep}} = \hat{U}^{\text{rep}}_{n,n^{\text{rep}},m}$ (of size $\text{poly}\log(n, n^{\text{rep}})$) of a circuit $\hat{U} = \hat{U}_{n,n^{\text{rep}},m}$ (of size $n \cdot n^{\text{rep}} \cdot \text{poly}\log(n, n^{\text{rep}})$) that satisfies the following requirements:*

1. *The circuit $\hat{U}$ takes as input a representation ζ^{rep} (of size at most n^{rep}) of a circuit ζ (of size at most n) over m inputs, a sequence of strings $c^1, \ldots, c^\alpha$ of length $O(n^{\text{rep}})$, a sequence of strings $y^1, \ldots, y^\beta$ of length m, and a string z of length $O(n)$. The numbers α and β are chosen such that the total length of each of $c^1 \circ \ldots, \circ c^\alpha$ and $y^1 \circ \ldots \circ y^\beta$ is at least $1/4$ fraction of the input length of $\hat{U}$.*

2. *For every representation ζ^{rep} and strings $c^1, \ldots, c^\alpha, y^1, \ldots, y^\beta$, there exists a string z such that $\hat{U}$ accepts ζ^{rep}, $c^1, \ldots, c^\alpha, y^1, \ldots, y^\beta$ and z if and only if the following conditions hold:*

   a) $c^1 = \ldots = c^\alpha$ and $y^1 = \ldots = y^\beta$.

   b) c^1 is the encoding of ζ^{rep} via the error correcting codes of Fact 3.5.14.

   c) y^1 is a satisfying assignment of ζ.

   As before, we refer to z as the witness that convinces U' that y^1 satisfies ζ.

3.5.8 Bounding the fan-in and fan-out of input circuits

So far, we discussed circuits with unbounded fan-in and fan-out. In particular, our definition of assignment testers requires an assignment tester to take as input a circuit φ with arbitrarily large fan-in and fan-out. However, it may be easier sometimes to construct an assignment tester that can only handle input circuits φ with bounded fan-in and fan-out. Thus, we would like to reduce the construction of general assignment testers to the construction of assignment testers that can only handle circuits with bounded fan-in and fan-out. While such a reduction is trivial to do in polynomial time in the size of the circuits, it is not clear that this can be done in the super-fast settings, where we only work with succinct representations.

In this section, we observe that it is possible to transform assignment testers that can only handle bounded fan-in and fan-out into a full-fledged assignment tester that can deal with arbitrarily large fan-in and fan-out, while paying a small cost in the parameters. This implies that, in order to construct a full-fledged assignment tester, it suffices to construct an assignment tester that can only handle input circuits φ with bounded fan-in and fan-out, which may be easier. In particular, we use this observation in Section 3.7 to simplify our circuit decomposition method. Formally, we observe the following.

Lemma 3.5.29 (Fan-in/out Lemma)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

  1. *An assignment tester A that is only guaranteed to work for input circuits with fan-in and fan-out that are upper bounded by 2, and which has input size n, outputs' number R, outputs' size s, rejection ratio ρ, tester size t, input representation size n^{rep} and output size s^{rep}.*
  2. *A reverse lister RL for A of size at most t.*

- ***Output:***

  1. *An assignment tester A' that works for arbitrary input circuits, and which has input size $n' = n/n^{\text{rep}} \cdot \text{poly}\log(n)$, outputs' number $R' = O(R)$, outputs' size $s' = \max\{s, O(1)\}$, rejection ratio $\rho' = \Omega(\rho)$, tester size $t' = O(t) + \text{poly}(n^{\text{rep}}, \log n)$, input representation size n^{rep} and output size $s^{\text{rep}} + \text{poly}\log s$.*
  2. *An reverse lister RL' of size at most t'.*

Proof idea. The proof is identical to the proof of the input representation lemma (Lemma 3.5.25) for $n^{\text{rep}\prime} = n^{\text{rep}}$, combined with the observation that the augmented universal circuit $\hat{U}$ can be implemented with fan-in and fan-out 2. Little more specifically, A' acts as follows: when given an input circuit φ with arbitrary fan-in and fan-out, we invoke A on $\hat{U}$, and hardwire the representation of φ into the output circuits of A.

In order for this argument to work, we need to show that $\hat{U}$ can indeed be implemented with fan-in and fan-out 2. This is not hard to prove, but is tedious, and we do not include the proof here. ■

Remark 3.5.30. Both the input representation lemma (Lemma 3.5.25) and the above fan-in/out lemma are instances of a more general phenomena: the augmented universal circuit $\hat{U}$ is "complete" for assignment testers, in the sense that given an assignment tester that can only take $\hat{U}_{n,n^{\text{rep}},m}$ as an input circuit, one can construct an assignment tester for arbitrary circuits of size n over m inputs that have representations of size n^{rep}. In other words, when constructing assignment testers, we can always assume without loss of generality that the input circuit φ is the circuit $\hat{U}_{n,n^{\text{rep}},m}$ (for some n, n^{rep}, and m), and then move to arbitrary circuits using the construction that is used in the proof of the input representation lemma. We do not use this more general claim in this chapter.

3.6 Proof of the Main Theorem

In this section we state our main lemmas - the decomposition lemma and the tensor product lemma - and prove the main theorem, restated below, relying on those lemmas.

Theorem (3.4.11, Main Theorem)**.** *There exists an infinite family of circuits $\{A_{n,n^{\text{rep}}}\}_{n=1,n^{\text{rep}}=1}^{\infty}$, such that $A_{n,n^{\text{rep}}}$ is an assignment tester for circuits of size n with outputs' number $R(n) = \text{poly}(n)$, outputs' size $s(n) = O(1)$, proof length $\ell(n) = \text{poly}(n)$, rejection ratio $\rho = \Omega(1)$, tester size $t(n, n^{\text{rep}}) = \text{poly}(\log n, n^{\text{rep}})$, input representation size n^{rep}, and output representation size $s^{\text{rep}}(n, n^{\text{rep}}) = O(1)$. Furthermore, there exists an algorithm that on inputs n and n^{rep}, runs in time $\text{poly}(\log n, n^{\text{rep}})$ and outputs $A_{n,n^{\text{rep}}}$.*

This section is organized as follows: First, in Section 3.6.1, we first define the notion of circuit decomposition with matrix access, which is used in both lemmas. Then, in Section 3.6.2, we state the lemmas and prove the main theorem. Recall that our main theorem is the following.

3.6.1 Circuit Decompositions with Matrix Access

A circuit decomposition is an assignment tester with the trivial soundness requirement. That is, the soundness requirement only requires that if the circuit decomposition is invoked on an input circuit φ and on assignment x *that does not satisfy* φ, then at least one output circuit rejects $x \circ \pi$. Formally, we define circuit decomposition as follows.

Definition 3.6.1. We say that D is a (circuit) decomposition if D is an assignment tester with rejection ratio $1/R$, where R is the outputs' number of D.

Remark 3.6.2. As a simple example, one can consider the circuit decomposition that when given an input circuit φ, transforms it into a 3-SAT formula and outputs each of the clauses of the formula as a separate output circuit. If φ is of size n, then this circuit decomposition has outputs' number $O(n)$ and outputs' size $O(1)$.

We turn to define the notion of "matrix access". Basically, the property of matrix access is a special case of block access, in which all the blocks are of the same width. In such case, one can think of the blocks as rows of a matrix. During the course of the proof of the tensor product lemma, we will use such a decomposition to construct a assignment tester that has block access and whose blocks are *the columns of the latter matrix*. To this end, we need to use a little more involved definition of matrix access:

1. First, recall that the definition of block access requires that each block contained coordinates of either only the tested assignment or of the proof string. In our case, this requirement should apply to both the rows and the columns, which is difficult to satisfy while using a single matrix. This issue is resolved by considering two matrices - one matrix whose rows are the assignment blocks of the decomposition (the "assignment matrix"), and a second matrix whose rows are the proof blocks of the decomposition (the "proof matrix"). In this way, the aforementioned requirement of the definition of block access is trivially satisfied.

2. Next, recall that the definition of block-based assignment tester requires that each assignment block contains at least (1/3) fraction of non-dummy coordinates. Note that the *rows* of the assignment matrix clearly satisfy this property, since a decomposition that has matrix access in particular has block access with the blocks being the rows of the matrices. However, we must also guarantee that the *columns* of the assignment matrix have this property, and we therefore add this as a requirement to the definition of matrix access.

3. For technical reasons that have to do with the proof of the tensor product lemma, we also require that every output circuit reads exactly the same numbers of assignment blocks and proof blocks, and that the assignment blocks precede the proof blocks in the input of each output circuit.

The foregoing considerations lead to the following definition of assignment tester that has matrix access.

Definition 3.6.3. A circuit decomposition D is said to have b-matrix access if it has b-block access, and for every input circuit φ the following hold: Let $B_1, \ldots, B_p$ be the partition to blocks that corresponds to φ, and let a denote the number of assignment blocks. Then:

1. All the proof blocks $B_{a+1}, \ldots, B_p$ are required to have the same width, denote it w_π. Also, recall that all the assignment blocks are already required to have the same width by the definition of block access (Definition 3.5.18), and denote this width by w_a.

2. We define the assignment matrix to be the matrix whose rows are the assignment blocks $B_1, \ldots, B_a$. Then, we is required that at least one third of the coordinates of each column of the assignment

matrix are non-dummy coordinates. Formally, for each $v \in [w_a]$, we define the v-th column $C_v : [a] \to [m+\ell]$ to be the function defined by $C_v(j) = B_j(v)$, and require that for at least one third of the indices $j \in [a]$ it holds that $C_v(j) \neq \mathsf{dummy}$.

3. Every output circuit reads the same number of assignment blocks and the same number of proof blocks.

4. For every output circuit ψ_i of D, the assignment blocks precede the output blocks in the input of ψ_i.

3.6.2 The main lemmas and the proof of the main theorem

We turn to state our main lemmas and prove the main theorem. The first lemma (the circuit decomposition lemma) says roughly that for every n there is a super decomposition that has outputs' number $\approx \sqrt{n}$, outputs' size $\approx \sqrt{n}$, and $O(1)$-matrix access. The second lemma (the tensor product lemma) says roughly that given a super-fast decomposition D for circuits of size n_D that has $O(1)$-matrix access, and a super-fast assignment tester A for circuits of size $n_A \ll n_D$, we can construct a super-fast assignment tester A' for circuits of size n_D, provided that n_A is slightly larger than both the outputs' number and the outputs' size of D. By combining the two lemmas, we obtain a procedure that takes as input a super-fast assignment tester for circuits of size $\approx \sqrt{n}$ and lifts it to a super-fast assignment tester for circuits of size n, which is the core of our construction. We begin by stating the circuit decomposition lemma, which is proved in Section 3.7.

Lemma 3.6.4 (Circuit Decomposition Lemma)**.** *There exists a procedure that when given as inputs numbers $n, n^{\mathrm{rep}} \in \mathbb{N}$, runs in time* $\mathrm{poly}(\log n, n^{\mathrm{rep}})$ *and outputs the following:*

1. *A circuit decomposition D for circuits of size n that has 6-matrix access, outputs' number $R_D(n) \stackrel{\mathrm{def}}{=} \tilde{O}(\sqrt{n})$, outputs' size $s_D(n) \stackrel{\mathrm{def}}{=} \tilde{O}(\sqrt{n})$, tester size $t_D(n, n^{\mathrm{rep}}) \stackrel{\mathrm{def}}{=} \mathrm{poly}\log n + O(n^{\mathrm{rep}})$, input representation size n^{rep}, and output representation size $s_D^{\mathrm{rep}}(n, n^{\mathrm{rep}}) \stackrel{\mathrm{def}}{=} \mathrm{poly}\log n + O(n^{\mathrm{rep}})$.*

2. *A reverse lister RL for D of size at most $t_D(n, n^{\mathrm{rep}})$.*

3. *A block-access circuit BA of size at most $t_D(n, n^{\mathrm{rep}})$.*

We proceed to state the tensor product lemma, which is proved in Section 3.8. The general statement of the lemma is somewhat involved. Therefore, in order to simplify the presentation, we first state a simplified version of the lemma that is specialized for the range of parameters that we are interested in, and then state the general version of the lemma.

Lemma 3.6.5 (Tensor Product Lemma, simplified version)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

  1. *A circuit decomposition D for circuits of size n_D that has $O(1)$-matrix access, outputs' number R_D, outputs' size s_D, tester size $t_D = \mathrm{poly}\log(n_D)$, input representation size $\mathrm{poly}\log(n_D)$, and output representation size $\mathrm{poly}\log(n_D)$.*

  2. *An assignment tester A for circuits of size $n_A \geq \tilde{O}(s_D) + O(R_D)$ that has outputs' number R_A, outputs' size $O(1)$, rejection ratio ρ_A, tester size $\mathrm{poly}\log(n_D)$, input representation size $\mathrm{poly}\log(n_D) \geq t_D$ and output representation size $O(1)$.*

  3. *Reverse listers RL_D and RL_A for D and A of sizes at most t_D and t_A respectively.*

4. *A block-access circuit BA_D for D of size at most t_D.*

- ***Output:***

1. *An assignment tester A' for circuits of size n_D with outputs' number $O(R_A^2)$, outputs' size $O(1)$, rejection ratio $\Omega\left(\rho_A^2\right)$, tester size* $\operatorname{poly}\log(n_D)$*, input representation size n_D^{rep}, and output representation size $O(1)$.*
2. *An reverse lister RL' for A' of size at most t'.*

We proceed to state the general version of the lemma. The main changes are the following: the circuit decomposition D has b-matrix access for arbitrary b (rather than $O(1)$), has arbitrary tester size t_D and input representation size n_D^{rep} (rather than $\operatorname{poly}\log(n_D)$), and has arbitrary output representation size s_D^{rep} (rather than $\operatorname{poly}\log(s_D)$); the assignment tester A has arbitrary outputs' size s_A and output representation size s_A^{rep} (rather than $O(1)$), and has arbitrary tester size t_A and input representation size n_A^{rep} (rather than $\operatorname{poly}\log(n_D)$).

Lemma 3.6.6 (Tensor Product Lemma, general version)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

1. *A circuit decomposition D for circuits of size n_D that has b-matrix access, outputs' number R_D, outputs' size s_D, tester size t_D, input representation size n_D^{rep}, and output representation size s_D^{rep}.*
2. *An assignment tester A for circuits of size n_A that has outputs' number R_A, outputs' size s_A, rejection ratio ρ_A, tester size t_A, input representation size $n_A^{\mathrm{rep}}$and output representation size s_A^{rep}.*
3. *Reverse listers RL_D and RL_A for D and A of sizes at most t_D and t_A respectively.*
4. *A block-access circuit BA_D for D of size at most t_D.*
5. *Furthermore, the following inequalities should hold:*

$$\begin{aligned} n_A &\geq b \cdot s_D \cdot s_D^{\mathrm{rep}} \cdot \operatorname{poly}\log\left(b, s_D\right) + O(R_D \cdot s_A^2) \\ n_A^{\mathrm{rep}} &\geq O\left(t_D\right) + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + s_A \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$

where the degrees of the polynomials and the constants in the big-O notations are unspecified universal constants.

- ***Output:***

1. *An assignment tester A' for circuits of size n_D with outputs' number $O(R_A^2)$, outputs' size $O(s_A)$, rejection ratio $\Omega\left(\rho_A^2/(s_A \cdot b)\right)$, tester size*

$$\begin{aligned} t' &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s_A^{\mathrm{rep}}, \log n_A, \log R_A\right) \\ &\quad + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + b \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$

input representation size n_D^{rep}, and output representation size $s_A^{\mathrm{rep}} + \operatorname{poly}\log(s_A)$.
2. *An reverse lister RL' for A' of size at most t'.*

By combining the two main lemmas, we obtain a procedure that allows us to square the input size of an assignment tester A at the cost of roughly squaring the outputs' number of A, provided that the outputs' size of the original A is constant. In order to use this procedure in our main construction, we also augment it in two ways:

1. The use of the tensor product lemma decreases[7] the input representation size of the assignment tester A, while we want the input representation size to increase, in order to accommodate the larger input size. We resolve this issue by invoking the input representation lemma (Lemma 3.5.25) to increase the input representation size.

2. The use of the tensor product lemma decreases the rejection ratio of the assignment tester, while we wish to maintain it. In order to do so, we invoke Dinur's amplification theorem (Theorem 3.5.5) to increase the rejection ratio back to its original value.

We summarize the resulting procedure in the following lemma, to which we refer as the "single iteration lemma", since it will form a single iteration in our construction of assignment testers.

Lemma 3.6.7 (Single Iteration Lemma). *Let s_0 and ρ_0 be the constants stated in the amplification theorem (Theorem 3.5.5). Then, for every sufficiently large constant $c \in \mathbb{N}$, there exists a polynomial time procedure that satisfies the following requirements:*

- *Input:*

  1. *An assignment tester A for circuits of size n with outputs' number R, outputs' size s_0, rejection ratio ρ_0, tester size t, input representation size $n^{\text{rep}} \overset{\text{def}}{=} c \cdot \log^c n$ and output representation size at most s_0.*
  2. *A reverse lister RL of size at most t.*

- *Output:*

  1. *An assignment tester A' for circuits of size at least $n' \overset{\text{def}}{=} n^2/\text{poly}\log n$ with outputs' number $O(R^2)$, outputs' size s_0, rejection ratio ρ_0, tester size at most $t' = O(t) + \text{poly}\log(R, n)$, input representation size $n^{\text{rep}\prime} \overset{\text{def}}{=} c \cdot \log^c(n')$ and output representation size s_0.*
  2. *A reverse lister RL' for A' of size at most t'.*

Proof. Let A be as in the single iteration lemma, and let $c \in \mathbb{N}$ be some constant that is sufficiently large to allow the choices of the parameters in the rest of the proof. Let D be the circuit decomposition that is obtained from the circuit decomposition lemma (Lemma 3.6.4) for input size $n_D = n^2/\text{poly}\log n$ and input representation size $n_D^{\text{rep}} = \text{poly}\log n$, where n_D and n_D^{rep} are chosen such that A and D satisfy the requirements of the tensor product lemma (Lemma 3.6.6). That is, we choose n_D and n_D^{rep} such that corresponding outputs' number R_D, outputs' size s_D, tester size t_D, and output representation size s_D of D satisfy

$$\begin{aligned} n &\geq 6 \cdot s_D \cdot s_D^{\text{rep}} \cdot \text{poly}\log(6, s_D) + O(R_D) = \sqrt{n_D} \cdot n_D^{\text{rep}} \cdot \text{poly}\log n_D, \\ n^{\text{rep}} &\geq O(t_D) + \text{poly}(s_D^{\text{rep}}, 6, \log s_D) + s_A \cdot \text{poly}\log(R_D, s_D, n_D, R_A, s_A, 6) \\ &= \text{poly}(\log n_D, n_D^{\text{rep}}), \end{aligned}$$

[7]Note that the input representation size of A' is n_D^{rep}, which is upper bounded by t_D, which in turn is upper bounded by n_A^{rep} due to the requirement regarding n_A^{rep}.

where the hidden constants are determined by the decomposition and the tensor product lemma. It can be verified that for sufficiently large choice of the constant c, one can indeed choose $n_D^{\text{rep}} = \text{poly}\log n$ in a way that satisfies the above inequalities.

We note that the decomposition D has outputs' number $\tilde{O}(n)$, outputs' size $\tilde{O}(n)$, tester size $t_D \overset{\text{def}}{=} \text{poly}\log(n)$, input representation size n_D^{rep}, and output representation size $\text{poly}\log(n)$, and that D is 6-matrix based. We also obtain from the circuit decomposition lemma a reverse lister RL_D for D and a block access circuit BA_D for D, both of size at most t_D.

We now invoke the tensor product lemma on D and A, resulting in an assignment tester A_1 for circuits of size n_D with outputs' number $R_1 \overset{\text{def}}{=} O(R^2)$, outputs' size s_0, rejection ratio $\rho_1 \overset{\text{def}}{=} \Omega\left(\rho_0^2/(s_0 \cdot 6)\right) = \Omega(1)$, tester size

$$\begin{aligned} t_1 &\overset{\text{def}}{=} O\left(t_D + s_0 \cdot t\right) + s_0 \cdot \text{poly}\left(s_0, \log n, \log R\right) \\ &\quad + \text{poly}\left(s_D^{\text{rep}}, 6, \log s_D\right) + 6 \cdot \text{poly}\log\left(R_D, s_D, n_D, R, s_0, 6\right), \\ &= O(t) + \text{poly}\log(n, R), \end{aligned}$$

input representation size n_D^{rep} and output representation size s_0. We also obtain a reverse lister RL_1 for A_1 of size at most t_1.

Next, we apply the input representation lemma (Lemma 3.5.25) to A_1, with input representation size $n^{\text{rep}\prime} \overset{\text{def}}{=} c \cdot \log^c(n^2)$. We therefore obtain an assignment tester A_2 for circuits of size

$$n' \overset{\text{def}}{=} n_D / n^{\text{rep}\prime} \cdot \text{poly}\log n_D = n^2/\text{poly}\log n$$

that has outputs' number $R_2 = O(R_1) = O(R^2)$, outputs' size $s_2 = \max\{s_0, O(1)\}$, rejection ratio $\rho_2 \overset{\text{def}}{=} \Omega(\rho_1) = \Omega(1)$, tester size

$$t_2 \overset{\text{def}}{=} O(t_1) + \text{poly}\left(n^{\text{rep}\prime}, \log n\right) = O(t) + \text{poly}\log(n, R),$$

input representation size $n^{\text{rep}\prime} \geq c \cdot \log^c(n')$ and output representation size $\max\{s_0, O(1)\}$. We also obtain a reverse lister RL_2 for A_2 of size at most t_2. We mention that in order for us to be able to apply the input representation lemma, we need n_D^{rep} to be sufficiently large. However, for sufficiently large choice of the constant c, it is possible to choose n_D^{rep} in a way such that the conditions of both the tensor product lemma and the input representation lemma are met.

Finally, we apply the amplification theorem (Theorem 3.5.5) to A_2, resulting in an assignment tester A' for circuits of size n' with outputs' number $\text{poly}\left(s_0, \frac{1}{\rho_2}\right) \cdot R_2 = O(R^2)$, outputs' size s_0, rejection ratio ρ_0, tester size

$$t' \overset{\text{def}}{=} \text{poly}\left(s_0, \frac{1}{\rho_2}\right) \cdot \left(t_2 + \text{poly}(s_0) + \text{poly}\log(R_2, n')\right) = O(t) + \text{poly}\log(n, R),$$

input representation size $n^{\text{rep}\prime}$ and output representation size s_0. We also obtain a reverse lister RL' for A' of size at most t'. We now output A' and RL' as the required assignment tester and reverse lister. ■

We finally turn to prove our main theorem, restated below, by starting from an assignment tester for circuits of constant size, and applying the single iteration lemma for $O(\log\log n)$ times.

Theorem (3.4.11, Main Theorem). *There exists an infinite family of circuits $\{A_{n,n^{\text{rep}}}\}_{n=1,n^{\text{rep}}=1}^{\infty}$, such that $A_{n,n^{\text{rep}}}$ is an assignment tester for circuits of size n with outputs' number $R(n) = \text{poly}(n)$, outputs' size $s(n) = O(1)$, proof length $\ell(n) = \text{poly}(n)$, rejection ratio $\rho = \Omega(1)$, tester size $t(n, n^{\text{rep}}) = \text{poly}(\log n, n^{\text{rep}})$, input representation size n^{rep}, and output representation size $s^{\text{rep}}(n, n^{\text{rep}}) = O(1)$. Furthermore, there exists an algorithm that on inputs n and n^{rep}, runs in time $\text{poly}(\log n, n^{\text{rep}})$ and outputs $A_{n,n^{\text{rep}}}$.*

Proof. Let c be a constant that will be determined later. We will choose c to be sufficiently large to match the requirements of the single iteration lemma (Lemma 3.6.7). We first show how to construct assignment testers $A_{n,n^{\mathrm{rep}}}$ for $n^{\mathrm{rep}} = c \cdot \log^c n$ by iterative application of the single iteration lemma (Lemma 3.6.7), and then use the input representation lemma (Lemma 3.5.25) to obtain assignment testers for any desired input representation size. We also mention that in the following proof we do not prove that the proof length is $\mathrm{poly}(n)$, but as in the rest of this chapter, this can be established using the upper bound $\ell \le R \cdot s$ (Theorem 3.5.4). Details follow.

Let $n \in \mathbb{N}$ and let $n^{\mathrm{rep}} = c \cdot \log^c n$. Let s_0 and ρ_0 be as in the single iteration lemma. We construct an assignment tester $A_{n,n^{\mathrm{rep}}}$ as in the theorem as follows. We begin by constructing an assignment tester A_0 for circuits of size n_0, where n_0 is some sufficiently large constant that is independent of n. We construct A_0 such that it has outputs' size s_0, rejection ratio ρ_0, input representation size $n_0^{\mathrm{rep}} \overset{\mathrm{def}}{=} c \cdot \log^c n_0$, and output representation size s_0, and such that its outputs' number R_0, and tester size t_0, are constants independent of n. Such an assignment tester A_0 can be constructed, for example, by using the circuit decomposition lemma[8] (Lemma 3.6.4) to generate a circuit decomposition D with input size n_0 and input representation size n_0^{rep}, and then invoking the amplification theorem (Theorem 3.5.5) to D in order to yield A_0. This also yields a reverse lister RL_0 for A_0.

Now, for each natural number $i \ge 1$, we let A_i and RL_i be the assignment tester and reverse lister that are obtained by invoking the single iteration lemma on the assignment tester A_{i-1} and on the reverse lister RL_{i-1}. We denote by n_i the input size of A_i, by R_i the outputs' number of A_i, and by t_i the tester size of A_i. Let k be the least natural number such that $n_k \ge n$. We output A_k as our desired assignment tester $A_{n,n^{\mathrm{rep}}}$.

We turn to analyze the parameters of A_k, and start with finding an upper bound on k. By the single iteration lemma, there exists a constant d such that for every i it holds that $n_{i+1} = n_i^2/d \cdot \log^d n$. It is not hard to show that for every i it holds that $n_i \ge \left(n_0/d \cdot 2^d \cdot \log^d n_0\right)^{2^i}$ (see [Mei09, App. C] for details). Hence, by taking n_0 to be sufficiently large such that $n_0/d \cdot 2^d \cdot \log^d n_0 > 1$, we get that

$$k \le \log_2 \log_{\left(n_0/d \cdot 2^d \cdot \log^d n_0\right)} n = \log\log n - O(1)$$

It is not hard to see that A_k has outputs' size s_0, rejection ratio ρ_0, input representation size at least $c \cdot \log^c n$, and output representation size s_0. We proceed to analyze the outputs' number and tester size of A_k. By the single iteration lemma, there exists some constant h_R such that for each $i \ge 1$ it holds that $R_i \le h_R \cdot (R_{i-1})^2$. It is not hard to prove by induction that

$$R_i = h_R^{\sum_{j=0}^{i-1} 2^j} R_0^{2^i} \le (h_R \cdot R_0)^{2^i},$$

and therefore

$$R_k \le (h_R \cdot R_0)^{2^k} \le (h_R \cdot R_0)^{\log n} \le \mathrm{poly}(n).$$

In addition, by the single iteration lemma, there exists some constant h_t such that for each $i \ge 1$ it holds that $t_i \le h_t \cdot t + \mathrm{poly}\log n$, and therefore

$$t_k \le h_t^k \cdot t_0 + k \cdot h_t^k \cdot \mathrm{poly}\log n \le \mathrm{poly}\log n,$$

as required. Finally, observe that it is possible to compute A_k in time $\mathrm{poly}\log n$.

We now consider the case of general values of n^{rep}. Let $n, n^{\mathrm{rep}} \in \mathbb{N}$. We construct the assignment tester $A_{n,n^{\mathrm{rep}}}$ as follows. We first observe that we may assume without loss of generality that $n^{\mathrm{rep}} \le \mathrm{poly}(n)$, since every circuit of size n has a trivial representation of size $\mathrm{poly}(n)$. Let $n' = n \cdot n^{\mathrm{rep}} \cdot$

[8]Actually, since we do not need D to have matrix-access here, we can use the simpler decomposition described in Remark 3.6.2, and thus have an even simpler construction of A_0.

$\operatorname{poly}\log n$, and let $n^{\text{rep}\prime} = c \cdot \log^c(n')$. We construct the assignment tester $A_{n',n^{\text{rep}\prime}}$, and invoke the input representation lemma (Lemma 3.5.25) on $A_{n',n^{\text{rep}\prime}}$ to increase its input representation size to n^{rep}. We then output the resulting assignment tester as $A_{n,n^{\text{rep}}}$. It is not hard to check that $A_{n,n^{\text{rep}}}$ has the required parameters, and that the latter invocation of the input representation lemma is indeed legal, since the assignment tester $A_{n',n^{\text{rep}\prime}}$ indeed satisfies the requirements of the input representation lemma for sufficiently large choice of c. ■

3.7 Circuit Decomposition Lemma

In this section, we prove the circuit decomposition lemma, restated below.

Lemma (3.6.4, Circuit decomposition lemma, restated)**.** *There exists a procedure that when given as inputs numbers $n, n^{\text{rep}} \in \mathbb{N}$, runs in time $\operatorname{poly}(\log n, n^{\text{rep}})$ and outputs the following:*

1. *A circuit decomposition D for circuits of size n that has 6-matrix access, outputs' number $R_D(n) \overset{\text{def}}{=} \tilde{O}(\sqrt{n})$, outputs' size $s_D(n) \overset{\text{def}}{=} \tilde{O}(\sqrt{n})$, tester size $t_D(n, n^{\text{rep}}) \overset{\text{def}}{=} \operatorname{poly}\log n + O(n^{\text{rep}})$, input representation size n^{rep}, and output representation size $s_D^{\text{rep}}(n, n^{\text{rep}}) \overset{\text{def}}{=} \operatorname{poly}\log n + O(n^{\text{rep}})$.*
2. *A reverse lister RL for D of size at most $t_D(n, n^{\text{rep}})$.*
3. *A block-access circuit BA of size at most $t_D(n, n^{\text{rep}})$.*

In Section 3.7.1, we give an overview of the proof which is more detailed than the one given in Section 3.3.2. Then, in Section 3.7.2, we provide the full proof of the lemma.

Bounding the fan-in and fan-out of circuits. In this section, we describe the construction of a circuit decomposition D that can only handle input circuits whose fan-in and fan-out are upper bounded by 2. However, such a decomposition can be transformed into a full-fledged decomposition, which can deal with arbitrary fan-in and fan-out, by using the fan-in/out lemma (Lemma 3.5.29).

3.7.1 Overview

In this section we give an overview of the construction of the circuit decomposition D from the decomposition lemma. In order to streamline the presentation, we describe D by describing its action on a fixed input circuit φ of size n over m inputs, on a fixed assignment $x \in \{0,1\}^m$, and on a fixed proof string π. As we mentioned above, the fan-in and fan-out of φ are assumed to be upper bounded by 2. Furthermore, recall that we want D to have 6-matrix access, which essentially means that:

1. The assignment x and proof string π should be arranged in two matrices, namely, the assignment matrix and the proof matrix.
2. Every output circuit of D should query 6 rows of the assignment and proof matrices. Actually, in this simplified overview, the output circuits of D will query even less than 6 rows of the matrices.

This overview is divided to two parts. First, in Section 3.7.1.1, we describe a construction of D while ignoring efficiency considerations, which yields a decomposition D that is not super-fast. Then, in Section 3.7.1.2 we describe how to modify D into a super-fast decomposition.

3.7.1.1 Warm-up: ignoring efficiency considerations

The basic idea. The basic structure of our construction of the decomposition D is similar to the construction of universal circuits in Section 3.5.7.2, and goes as follows. We require the proof string π to contain the following values:

- The value that each gate g of φ outputs when φ is invoked on x. Let us denote by k_g the coordinate of π that contains the value of g.
- The value that each wire (g_1, g_2) of φ carries when φ is invoked on x. Let us denote by $k_{(g_1,g_2)}$ the coordinate of π that contains the value of (g_1, g_2).

Then, the output circuits of D should check the following conditions hold:

1. For every input gate g, it holds that π_{k_g} equals to the corresponding assignment coordinate.
2. For the output gate g_{out} of φ, it holds that $\pi_{k_{g_{\text{out}}}} = 1$.
3. For every gate g and its outgoing wires (g, g_1) and (g, g_2), it holds that $\pi_{k_g} = \pi_{k_{(g,g_1)}} = \pi_{k_{(g,g_2)}}$.
4. For every gate g and its incoming wires (g_1, g) and (g_2, g), it holds that π_{k_g} is indeed the value that g outputs when given as input $\pi_{k_{(g,g_1)}}$ and $\pi_{k_{(g,g_2)}}$.

The nontrivial issue, of course, is to arrange the proof string π in a $O(\sqrt{n}) \times O(\sqrt{n})$ matrix such that the output circuits of D can verify the foregoing conditions and such that every output circuit queries only 6 rows of the matrix. Observe that arranging π in such a matrix would have been easy if we only wanted to verify only one of the Conditions 3 and 4:

- If we only wanted to verify the Condition 3, we could arrange π in a matrix such that for each a gate g and its outgoing wires (g, g_1) and (g, g_2), the coordinates k_g, $k_{(g,g_1)}$, and $k_{(g,g_2)}$ are in the same row. Using this arrangement, Conditions 3, 1, and 2 could be verified with each output circuit of D querying only one row of the matrix.
- On the other hand, if we only wanted to verify the Condition 4, we could arrange π in a matrix such that for each a gate g and its incoming wires (g_1, g) and (g_2, g), the coordinates k_g, $k_{(g_1,g)}$, and $k_{(g_2,g)}$ are in the same row. Using this arrangement, Conditions 4, 1, and 2 could be verified with each output circuit of D querying only one row of the matrix.

The problem is that we do not know if there is a single arrangement of π in a matrix that allows verifying both conditions simultaneously. Our first step toward solving the problem is to require π to contain two matrices M and N, such that each of M and N contains a copy of the value of each gate and each wire, and such that M is a arranged in the way that allows verifying Condition 3 (as described above), and N is arranged in a way that allows verifying Condition 4 (as described above). More specifically,

- For each gate g, the proof string π has a coordinate $k_{M,g}$ of the matrix M, and a coordinate $k_{N,g}$ of the matrix N, where both coordinates should contain the value that g outputs when φ is invoked on the assignment x. The matrices M and N also have similar coordinates $k_{M,(g_1,g_2)}$ and $k_{N,(g_1,g_2)}$ for each wire (g_1, g_2).
- For each a gate g and its outgoing wires (g, g_1) and (g, g_2), the coordinates $k_{M,g}$, $k_{M,(g,g_1)}$, and $k_{M,(g,g_2)}$ are in the same row of M. Similarly, for each a gate g and its incoming wires (g_1, g) and (g_2, g), the coordinates $k_{N,g}$, $k_{N,(g_1,g)}$, and $k_{N,(g_2,g)}$ are in the same row of N.

This way, the output circuits of D can verify that M satisfies Condition 3 and that N satisfies Condition 4, while each output circuit queries only one row of those matrices. By concatenating the matrices M and N into a single matrix, we arrange π in a single matrix such that the two conditions can be verified while each output circuit queries only one row of this matrix.

Of course, in order for the foregoing construction of D to be sound, we also need to verify that M and N are consistent. That is, in addition to verifying the above conditions, the decomposition$D$ must also verify that for each gate g and wire (g_1, g_2) it holds that

$$\pi_{k_{M,g}} = \pi_{k_{N,g}} \text{ and } \pi_{k_{M,(g_1,g_2)}} = \pi_{k_{N,(g_1,g_2)}}. \tag{3.1}$$

Moreover, D must verify that Equality 3.1 holds while maintaining the property that each output circuit queries at most 6 rows of the proof matrix. Overcoming this issue of verifying the consistency of M and N while maintaining the matrix access property is the main technical challenge that we deal with in our construction of D.

Verifying the consistency of M and N. We now describe how D checks the consistency of the matrices M and N. As a warm-up, consider the following naive solution: For each row of M, the decomposition D will output a circuit ψ_i that will check the consistency of the coordinates in this row with the corresponding coordinates in N. This solution does not work, since some of those output circuits ψ_i may read too many rows of N. To see it, observe that for the coordinates of a given row of M, the corresponding coordinates in N may spread over all the rows of N rather then being concentrated in only 5 rows of N.

On a more intuitive level, the latter naive solution does not work because the order of the coordinates in M may be very different than the order of the corresponding coordinates in N. Our solution is to add to the proof matrix *auxiliary rows* that serve as a “bridge” between M and N, and to add output circuits of D that check the consistency of those auxiliary rows. More specifically, we add auxiliary rows and output circuits such that:

- Each auxiliary row v consists of coordinates $k_{v,g}$ and $k_{v,(g_1,g_2)}$ that correspond to some of the gates and wires of φ. As before, $\pi_{k_{v,g}}$ is supposed to contain the value that the gate g outputs when φ is invoked on x, and $\pi_{k_{v,(g_1,g_2)}}$ is supposed to contain the value that the wire (g_1, g_2) carries when φ is invoked on x. However, note that each auxiliary row is of width $O(\sqrt{n})$, so it does *not* contain a coordinate for *every* gate and wire.

- For each auxiliary row v, there will be an output circuit of D that checks the consistency of the row v with at most four other rows of the proof matrix. By saying that an output circuit ψ_i checks the consistency of two rows u and v, we mean that ψ_i checks, for every two coordinates $k_{u,g}$ and $k_{v,g}$ of u and v that correspond to the same gate g, that $\pi_{k_{u,g}} = \pi_{k_{v,g}}$, and the same for the wires.

- We will choose the auxiliary rows and additional output circuits such that all the additional output circuits are satisfied if and only if M and N are consistent.

It remains to explain how to choose for each auxiliary row v:

- For which gates g and wires (g_1, g_2) does the auxiliary row v have corresponding coordinates $k_{v,g}$ and $k_{v,(g_1,g_2)}$?

- What are the other rows of the proof matrix with which the auxiliary row v is checked for consistency.

To this end, we use routing networks. Recall that a routing network of order n is a graph with two special sets S and T, called the "sources" and the "targets", and that a routing network satisfies the following property: Suppose that for every source $s \in S$ there are d messages that should be sent to targets in T, and that every target $t \in T$ should receive d messages from sources in S. Then, it is possible to find a collection of paths $\mathcal{P}$ in G such that:

- Every message is routed through some path $p \in \mathcal{P}$. We say that the path p routes the message if its connects the source of the message to its target.
- Every vertex of G participates in at most d paths in $\mathcal{P}$.

Let $G = (V, E)$ be a routing network of order $O(\sqrt{n})$ with in-degree and out-degree upper bounded by 2, and let S and T be its sources and targets respectively. We identify the vertices of G with the rows of the proof matrix, and in particular we identify the rows of M with the vertices of S, the rows of N with the vertices of T, and the auxiliary rows of the proof matrix with the other vertices of G.

Now, for each gate g, we view the value that g outputs (under the assignment x) as a message that should be sent from the row of M that contains the coordinate $k_{M,g}$ to the row of N that contains the coordinate $k_{N,g}$, where we view those rows of M and N as a source and a target of G. We also view the values of wires (g_1, g_2) as messages in a similar way. Note that, when taking this view, each row of M needs to send $O(\sqrt{n})$ messages and each row of N needs to receive $O(\sqrt{n})$ messages.

Our next step is to find a collection of paths $\mathcal{P}$ along which the messages can be routed, such that each auxiliary row participates in the routing of at most $O(\sqrt{n})$ messages (i.e., the vertex of G that is identified with the auxiliary row participates in at most $O(\sqrt{n})$ paths in $\mathcal{P}$). For each auxiliary row v and a gate g, we define the auxiliary row v to contain a coordinate $k_{v,g}$ that corresponds to g if and only if the auxiliary row v participates in routing the message that corresponds to the value of g, and the same goes for each wire (g_1, g_2).

Finally, we define the output circuits of D that verify the consistency of the auxiliary rows as follows. For each vertex v of G, the decomposition outputs a circuit ψ_i that acts as follows. Suppose that v has outgoing edges to the vertices z_1 and z_2 of G (recall that the out-degree of G is upper bounded by 2). Then, the circuit ψ_i queries the rows that correspond to v, z_1, and z_2, and performs the following consistency check. For every gate g whose corresponding message is routed through v, the circuit ψ_i verifies that $\pi_{k_{v,g}} = \pi_{k_{z_1,g}}$ if the message of g is routed through z_1, and otherwise ψ_i verifies that $\pi_{k_{v,g}} = \pi_{k_{z_2,g}}$ (since if the message is not routed through z_1 then it must be routed through z_2). The circuit ψ_i also performs an analogous consistency check for every wire (g_1, g_2) whose corresponding message is routed through v.

This concludes the description of our way of verifying the consistency of M and N, and in particular our construction of the auxiliary rows and the output circuits of D. It is not hard to see that if all the output circuits of D accept, then M and N must be consistent. Observe that every output circuit queries at most three rows. Moreover, note that every auxiliary row contains at most $O(\sqrt{n})$ coordinates, since it participates in the routing of at most $O(\sqrt{n})$ messages.

The assignment matrix. Our discussion so far has focused on the proof string π and the arrangement of its coordinates in the proof matrix. However, we still need to describe the arrangement of the assignment x in the assignment matrix. We choose the assignment to be of width $w_a = \min\{\theta(\sqrt{n}), m\}$, and arrange the coordinates of x in this matrix according to their natural order.

Another issue that we ignored so far is that the decomposition D should check the consistency between the assignment matrix and the proof matrix. More formally, for every tested assignment coordinate $k_a \in [m]$ whose corresponding input gate of φ is g, the decomposition D should check that $x_{k_a} = \pi_{k_{M,g}}$. To this end, we choose the ordering of the coordinates of the matrix M such that the assignment coordinate k_a is in the j-th row of the assignment matrix if and only if the corresponding proof coordinate $k_{M,g}$ belongs to the j-th row of M. Then, for each j, the decomposition D outputs a circuit that checks that all the coordinates in the j-th row of the assignment matrix are consistent with the corresponding coordinates in the j-th row of the proof matrix.

Conclusion. It can be seen that x is a satisfying assignment if and only if there exists a proof string π that makes all the output circuits ψ_i accept. Moreover, observe that the number and size of output circuits of D is indeed $\tilde{O}(\sqrt{n})$, and that each output circuit queries at most three rows of the assignment and proof matrices. This concludes our construction of D that ignores the efficiency issues.

Remark 3.7.1. We mention again that the idea of using routing networks in the construction of PCPs is not new, and already appeared in several works on PCPs (see, e.g., [BFLS91, PS94]).

3.7.1.2 Obtaining a super-fast circuit decomposition

We turn to explain how to modify the foregoing circuit decomposition such that it will have a super-fast implementation. The main issue that needs to be resolved is the following: Recall that the decomposition routes messages on the routing network G. More specifically, the decomposition computes a collection $\mathcal{P}$ of paths on G that connect each coordinate of M to its corresponding coordinate of N, where each vertex in G participates in at most $O(\sqrt{n})$ such paths. However, those paths can not be computed by a super-fast decomposition, since merely writing those paths down requires writing $\Omega(n)$ bits, which would force the decomposition to be of size $\Omega(n)$ rather than $\operatorname{poly}\log n$. In order to resolve this issue, we modify the decomposition such that it does not compute those paths by itself. Instead, we require the proof string to contain those paths, and modify the decomposition such that it verifies that the paths that are given in the proof string are valid.

This idea is implemented as follows: for each coordinate $k_{v,g}$ in the proof string, we add to the row of $k_{v,g}$ additional $\log n$ coordinates that are supposed to be assigned the index of g - we refer to this index as the label of $k_{v,g}$. Similarly, for each coordinate $k_{v,(g_1,g_2)}$, we add to the row of $k_{v,(g_1,g_2)}$ additional $2\log n$ coordinates that are supposed to contain the indices of g_1 and g_2, and refer to the pair of those indices as the label of $k_{v,(g_1,g_2)}$. We note that this modification is also performed on rows of M and N.

Now, we modify the output circuits ψ_i that verify the consistency of the routing as follows. Let ψ_i be the output circuit that corresponds to a vertex v, and suppose that v has incoming edges from the vertices u_1 and u_2, and has outgoing edges to the vertices z_1 and z_2. Then, the output circuit ψ_i reads the rows that correspond to u_1, u_2, v, z_1, z_2, and verifies that the following conditions holds:

1. Every label in the row of v is found either in the row of u_1 or in the row of u_2.
2. Every label in the row of v is found either in the row of z_1 or in the row of z_2.
3. If a coordinate k in the row of v has the same label as a coordinate k' in the row of u_1, then $\pi_k = \pi_{k'}$. The same condition is checked when replacing u_1 with either of u_2, z_1, or z_2.

It can be seen that, if all the modified output circuits ψ_i accept, then we are guaranteed that the matrices M and N are consistent. In addition to modifying the output circuits as above, we also add new output circuits ψ_i that check that the coordinates of M and N have the correct labels, and in particular that:

- Every gate and every wire has corresponding coordinates in M and in N.
- For each a gate g and its outgoing wires (g, g_1) and (g, g_2), the coordinates $k_{M,g}$, $k_{M,(g,g_1)}$, and $k_{M,(g,g_2)}$ are in the same row of M.

- For each a gate g and its incoming wires (g_1, g) and (g_2, g), the coordinates $k_{N,g}$, $k_{N,(g_1,g)}$, and $k_{N,(g_2,g)}$ are in the same row of N.

Note that in the super-fast setting, in which the gate or wire to which a coordinate corresponds is determined by its label, the latter three conditions must indeed be verified, and can not be assumed to hold trivially as before. However, it is not hard to construct output circuits of D that verify those conditions.

It is not hard to see that the decomposition D remains sound after the foregoing modifications. Moreover, since now D needs not compute the paths $\mathcal{P}$ for routing the messages, it can be implemented in a super-fast way. This concludes the construction.

3.7.2 Proof of the circuit decomposition lemma

Below, we describe how to construct the circuit decomposition D for a given input size n and a given input representation size n^{rep}. This section is organized as follows: In Section 3.7.2.1, we describe the block structure of D. In Section 3.7.2.2 we describe the proof strings of D. In Section 3.7.2.3, we describe the output circuits of D and their queries. The remaining parts of the proof, namely, the construction of the reverse lister RL_D, the construction of the blocks access circuit BA_D, and the analysis of the parameters, are straightforward and will not be discussed.

3.7.2.1 The block structure of D

The assignment blocks. Let w_a be the width of the assignment blocks, to be chosen shortly below. We define the assignment blocks of D on input circuit φ as follows: There are $\lceil m/w_a \rceil$ assignment blocks, where the first assignment block consists of the coordinates $1, \ldots, w_a$, the second assignment block consists of the coordinates $w_a + 1, \ldots, 2 \cdot w_a$, etc. If w_a does not divide m, then the last assignment block consists of the last $m \bmod w_a$ assignment coordinates and of additional $w_a - (m \bmod w_a)$ dummy coordinates.

We now choose $w_a = \min\{\theta(\sqrt{n}), m\}$, where the constant in the Big-Theta notation is chosen as follows: Recall that the definition of block access requires that least $\frac{1}{3}$ fraction of the coordinates of every assignment block are non-dummy coordinates. The only assignment block that contains dummy coordinates is the last block, so we only need to take care of this block. To this end, we need to choose w_a such that, if $w_a < m$, then $(m \bmod w_a) \geq \frac{1}{3} \cdot w_a$. It is not hard to show that one can choose such a value of w_a that satisfies $w_a = \min\{\theta(\sqrt{n}), m\}$, as required.

The proof blocks. We turn to define the proof blocks of D. Let $w_r \overset{\text{def}}{=} 3 \cdot \max\{\sqrt{n}, w_a\}$.We denote by w_π the width of the proof blocks, and choose it to be $w_\pi \overset{\text{def}}{=} w_r \cdot (2 \cdot \log n + 3)$. the first proof block of D consists of the coordinates $m + 1, \ldots, m + w_\pi$, the second proof block of D consists of the coordinates $m + w_\pi + 1, \ldots, m + 2 \cdot w_\pi$, etc. We view each proof block as consisting of $3 \cdot w_r$ strings of length $2 \cdot \log n + 1$, to which we refer as the records of the block, and view each record of consisting of two parts:

1. The label of the record, which consists of two elements in $[n] \cup \{\perp\}$ - this part is represented by the first $2 \cdot (\log n + 1)$ bits of the record.
2. The value of the record, which is a Boolean value, and is represented by the last bit of the record.

Let $G \overset{\text{def}}{=} G_{\sqrt{n}}$ be the routing network of order $\sqrt{n}$ whose existence is guaranteed by Fact **??**, and let S and T be its sets of sources and targets respectively. We define the proof blocks of D to be in one-to-one correspondence with the vertices of G, where the j-th proof block corresponds to the j-th vertex of G. We view the $\sqrt{n}$ proof blocks which correspond to the vertices in S of G as an $\sqrt{n} \times w_\pi$ matrix M, and the $\sqrt{n}$ proof blocks which correspond to vertices in T as an $\sqrt{n} \times w_\pi$ matrix N.

3.7.2.2 The proof strings of D

Let x be a satisfying assignment for φ. We describe the proof string π that convinces D that x satisfies φ. We consider the collection of all the records in all of the blocks, and view each record as corresponding to a gate or a wire of φ, where each gate (or wire) has many records that correspond to it. The content of π at a given record depends on whether the record corresponds to a gate or to a wire, and is determined as follows:

1. If the record corresponds to a gate g, then the first element of the label of the record contains the index of g (which is a number from 1 to n), and the second element of the label contains the symbol $\perp$. The value of the record is set to be the value that g outputs when φ is invoked on the assignment x.

2. If the record corresponds to a wire (g_1, g_2), then first element of the label of the record contains the index of g_1 and the second element is of the label contains the index of g_2 (recall that those indices are numbers from 1 to n). The value of the record is set to be the value that is passed through (g_1, g_2) when φ is invoked on the assignment x.

3. Finally, some records are *dummy records* that do not correspond to a gate or a wire of φ. In this case, both elements of the label of the record are contain the symbol $\perp$, and the value of the record is arbitrary.

It remains to describe the correspondence between records and gates/wires - note that since the content of a record is determined by the gate/wire to which it corresponds, this correspondence determines π completely. We describe this correspondence separately for the matrix M, the matrix N, and the rest of the proof blocks.

The matrix M. There is a one-to-one correspondence between the records in M and the gates and wires of φ. If there are more records than gates and wires, then the superfluous records are set to be dummy records.
The order of the records in M is as follows: Each row of M consists of $w_r/3$ triplets of records, where each such triplet corresponds to a gate g of φ and is referred to as the triplet of g in M. Given a gate g, the triplet of g contains the record that corresponds to g, and also the records that corresponds to the (at most two) outgoing wires of g. If g has less than two outgoing wires, then the (one or two) superfluous records are set to be dummy records, which contain only zeroes. The triplets are ordered in M according to the order of the gates, from the first gate to the last.

The matrix N. The records of N are defined similarly to the records of M, with the following differences. For each gate g, the triplet of g in N contains the records corresponding to the *incoming* wires of g rather than the *outgoing* wires of g.

The auxiliary rows. Recall that each proof block corresponds to some vertex of the routing network G, where the matrices M and N correspond to the sources set S and targets set T of G. The correspondence of records to gates and wires in the auxiliary rows will be determined by routing on G, which will be performed using Proposition 3.2.22, restated below.

Proposition (3.2.22, routing of multiple messages, restated). *Let $G = (V, E)$ be a routing network of order n, let $S, T \subseteq V$ be the sets of sources and targets of G respectively, and let $d \in \mathbb{N}$. Let $\sigma \subseteq S \times T$ be a relation such that each $s \in S$ is the first element of at most d pairs in σ, and such that each $t \in T$ is the second element of at most d pairs in σ. We allow σ to be a multi-set, i.e., to contain the same element multiple times. Then, there exists a set $\mathcal{P}$ of paths in G such that the following holds:*

1. *For each $(s,t) \in \sigma$, there exists a path $p \in \mathcal{P}$ that corresponds to (s,t), whose first vertex is s and whose second vertex in t.*

   a) *Every vertex of G participates in at most d paths in $\mathcal{P}$.*

We turn to describing the routing. We construct a relation $\sigma \subseteq S \times T$ (actually, a multiset) as follows: For each gate g of φ, we add σ the pair (s,t), where

1. $s \in S$ is the vertex of G that corresponds to the row of M that contains the record of g in M.
2. $t \in T$ is the vertex of G that corresponds to the row of N that contains the record of g in N.

We do the same for the wires of φ. Now, by Proposition 3.2.22, there exists a collection $\mathcal{P}$of paths such that

1. For each $(s,t) \in \sigma$, there exists a path $p \in \mathcal{P}$ that corresponds to (s,t), whose first vertex is s and whose second vertex in t.
2. Every vertex of G participates in at most w_r paths in $\mathcal{P}$.

We now define the records of the auxiliary rows. For each gate g, we define the following records: let (s,t) be the element of σ that corresponds to g, and let $p \in \mathcal{P}$ be the path that corresponds to (s,t). Now, for each vertex v on the path p, the auxiliary row v contains a record that corresponds to g. The same goes for each wire (g_1, g_2) and the corresponding element (s,t) and path p. If a vertex v of G participates in less than w_r paths, then the remaining records of the auxiliary row v are set to be dummy records.

This concludes the description of the correspondence between the records and the gates/wires, and hence concludes the description of the proof string π.

3.7.2.3 The output circuits of D

In this section we describe the output circuits of the circuit decomposition D and their queries. Fix an input circuit φ of size n and over m inputs. Let V denote the vertex set of the routing network G. We define the outputs' number R_D of D to be $|V| + 2 \cdot \sqrt{n}$. We view the first $|V|$ output circuits as being in one-to-one correspondence with the vertices of G, the next $\sqrt{n}$ output circuits as being in one-to-one correspondence with the rows of M and the last $\sqrt{n}$ output circuits as being in one-to-one correspondence with the rows of N.

We describe for each index $i \in [R_D]$ what the i-th output circuit checks and what blocks it queries. We consider the following cases:

1. $i \leq |V|$: In such case, the i-th output circuit ψ_i corresponds to some vertex v of G, which in turn corresponds to the i-th proof block of D. The goal of the output circuit ψ_i is to check that the routing at the vertex v is valid - that is, that every record that is routed through v has came through one of the incoming edges of v and is sent through one of the outgoing edges of v. To this end, the output circuit ψ_i queries the blocks that correspond to v and its neighbors, and performs the following checks:

   a) Let $u_1, \ldots, u_d$ (for $d \leq 2$) be the vertices of G from which v has incoming edges (if v has no incoming edges, then this check is skipped). Then, ψ_i checks for each record of the i-th proof block that one of the blocks that correspond to $u_1, \ldots, u_d$ contains a record with the same label and the same value.

b) The same check as the first one, but for vertices of G to which v has outgoing edges, instead of vertices of G from which v has incoming edges. Again, if v has no outgoing edges, then this check is skipped.

We note that the output circuit ψ_i can be implemented in size $\tilde{O}(\sqrt{n})$ as follows: For the first check, ψ_i constructs a list of all the records that appear in the blocks that correspond to $u_1, \ldots, u_d$, and sorts those records according to lexicographic order of their label. Then, ψ_i sorts the records that appear in the i-th block according to their label, thus obtaining a second list of records. Finally, ψ_i checks that the second list of records is a sub-sequence of the first list of records. The second check can be implemented similarly. It can be verified that this implementation indeed requires a circuit of size $\tilde{O}(\sqrt{n})$.

2. $|V| + 1 \leq i \leq |V| + \sqrt{n}$: Let $j \stackrel{\text{def}}{=} i - |V|$. In this case, the output circuit ψ_i queries the j-th row of M and checks that it is of the form described in Section 3.7.2.2. That is, ψ_i checks that the j-th row of M consists of triplets of records, where each triplet consists of the record of a gate and of the records of the gate's *outgoing* wires, and where the values of the records of each triplet are equal. More specifically, for every $u \in [w_r]$, the output circuit ψ_i performs the following checks for the u-th triplet of the j-th row:

   a) ψ_i computes the index $h \stackrel{\text{def}}{=} (j-1) \cdot \max\{w_a, \sqrt{n}\} + u$ of the triplet among all the triplets of M. Let us denote the h-th gate of φ by g.

   b) ψ_i checks that the the first record of the triplet corresponds to g, that is, that the first element of the label of the first record contains h and that the second element contains $\perp$.

   c) ψ_i checks that the labels of the two last records of the triplet are the labels of the outgoing wires of the gate g. If g has only one outgoing wire then g checks that the label of the middle record is the label of this unique outgoing wire, and that the label of the last record consists of twice the symbol $\perp$. If g has no outgoing wires, then g checks that the labels of both the last records consist of twice the symbol $\perp$.

   d) ψ_i checks that the values of all the three records in the triplet are equal (recall that those values are supposed to be equal to the value that g outputs when φ is invoked on x).

   If $j \leq \lceil m/w_a \rceil$, then ψ_i also queries the j-th row of the assignment matrix, and checks consistency between this row and the records of M that correspond to input gates. Specifically, observe that for each assignment coordinate in the j-th row of the assignment matrix, the record of the corresponding input gate is found in the j-th row of M. The output circuit φ_i checks for each such assignment coordinate that it is equal to the value of the corresponding record.

3. $|V| + \sqrt{n} + 1 \leq i \leq |V| + 2 \cdot \sqrt{n}$: Let $j \stackrel{\text{def}}{=} i - |V| - \sqrt{n}$. In this case, the output circuit ψ_i queries the j-th row of N and checks that it is of the form described in Section 3.7.2.2. That is, ψ_i checks that the j-th row of N consists of triplets of records where each triplet consists of the record of a gate and of the records of its *incoming* wires, and where the value of each gate record is computed correctly from the values of the records of the incoming wires. To this end, ψ_i performs the same checks as in Case 2, with the following differences:

   a) Instead of checking for each triplet that the last two records contain the labels of the *outgoing* wires, ψ_i checks that last two records contain the labels of the *incoming* wires

   b) Instead of checking for each triplet that the values of all the three records are the same, φ_i checks that the value of the first record, which corresponds to the gate g, contains the value that g outputs when g is given as input the values of the two last records. If g has only one incoming wire, then we take only the value of the middle record, and if g has not incoming wires, then this check is skipped.

c) ψ_i does not check consistency with the tested assignment.

This concludes the description of the output circuits of D. It should be clear that those circuits are of size $\tilde{O}\left(\sqrt{n}\right)$. It is also not hard to see that both the circuit decomposition D and the reverse lister RL can be implemented in size $\operatorname{poly}\log n + O(n^{\mathrm{rep}})$, using the representation ν of G. The construction of the block-access circuit BA is straightforward as well, with one caveat: Recall that the definition of matrix access (Definition 3.6.3) requires that every output circuit ψ_i of D reads the same numbers of assignment blocks and proof blocks, and that the assignment blocks precede the proof blocks in the input of ψ_i. Thus, we modify the foregoing construction of D such that every output circuit ψ_i queries exactly one assignment blocks and five proof blocks in order to meet this requirement. In particular, if $i > |V|$, we modify ψ_i such that it queries the j-th row of M or N five times instead of one. In addition, if ψ_i is not among the circuits that check consistency between M and the assignment matrix, then we modify ψ_i such that it queries the first row of the assignment matrix and ignores it. This concludes the construction.

3.8 Tensor Product Lemma

In this section we prove the general version of the tensor product lemma, restated below.

Lemma (3.6.6, Tensor Product Lemma, restated)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***

  1. *A circuit decomposition D for circuits of size n_D that has b-matrix access, outputs' number R_D, outputs' size s_D, tester size t_D, input representation size n_D^{rep}, and output representation size s_D^{rep}.*
  2. *An assignment tester A for circuits of size n_A that has outputs' number R_A, outputs' size s_A, rejection ratio ρ_A, tester size t_A, input representation size $n_A^{\mathrm{rep}}$and output representation size s_A^{rep}.*
  3. *Reverse listers RL_D and RL_A for D and A of sizes at most t_D and t_A respectively.*
  4. *A block-access circuit BA_D for D of size at most t_D.*
  5. *Furthermore, the following inequalities should hold:*

  $$\begin{aligned} n_A &\geq b \cdot s_D \cdot s_D^{\mathrm{rep}} \cdot \operatorname{poly}\log(b, s_D) + O(R_D \cdot s_A^2) \\ n_A^{\mathrm{rep}} &\geq O\left(t_D\right) + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + s_A \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$

  where the degrees of the polynomials and the constants in the big-O notations are unspecified universal constants.

- ***Output:***

  1. *An assignment tester A' for circuits of size n_D with outputs' number $O(R_A^2)$, outputs' size $O(s_A)$, rejection ratio $\Omega\left(\rho_A^2/(s_A \cdot b)\right)$, tester size*

  $$\begin{aligned} t' &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s_A^{\mathrm{rep}}, \log n_A, \log R_A\right) \\ &\quad + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + b \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$

input representation size n_D^{rep}*, and output representation size* $s_A^{\mathrm{rep}} + \mathrm{poly}\log(s_A)$*.*

2. *An reverse lister* RL' *for* A' *of size at most* t'*.*

This section is organized as follows. In Section 3.8.1 we recall the ideas that underlie the construction of the assignment tester A', which were explained in Section 3.3, and sketch the technical complications that arise when realizing those ideas. In Section 3.8.2, we discuss a robustness property for decompositions, which differs from the notion of expected robustness defined in Section 3.5.4, and which is used in the proof of the tensor product lemma. Next, in Section 3.8.3, we describe the construction of an "intermediate assignment tester", which is the key step in the proof of the tensor product lemma. Finally, in Section 3.8.4, we complete the proof of the tensor product lemma using the intermediate assignment tester.

3.8.1 Proof overview

Let D be a decomposition that has matrix access and A be an assignment tester as in the tensor product lemma. For the purpose of this overview, we assume that the matrix access parameter b and the outputs' size s_A of A are constants. We would like to construct an assignment tester A' for circuits of size n_D, which has outputs' number $\approx R_A^2$, outputs' size s_A, and rejection ratio $\Omega(\rho_A^2/b \cdot s_A) = \Omega(\rho_A^2)$. We begin by recalling the construction of A' that was described in Section 3.3: When given as input a circuit φ of size n_D, the assignment tester A' takes the following steps.

1. A' invokes D on φ, thus obtaining circuits $\psi_1, \ldots, \psi_{R_D}$.
2. For each $i_D \in [R_D]$, the assignment tester A' invokes A on ψ_{i_D}, resulting in circuits $\xi_{i_D,1}, \ldots, \xi_{i_D,R_A}$.
3. For each $i_A \in [R_A]$, the assignment tester A' constructs the circuit $\eta_{i_A} \overset{\mathrm{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D.i_A}$, which corresponds to the i_A-th column of the matrix whose entries are ξ_{i_D,i_A}.
4. For each $i_A \in [R_A]$, the assignment tester A' invokes A on η_{i_A}.

The assignment tester A' finishes by outputting the output circuits of all the invocations of A on the circuits η_{i_A}.

In this section, it is more convenient for us to view of the construction of A' in a slightly different way than the one used in Section 3.3. We first define the "intermediate" assignment tester A_I which on input φ produces the circuits $\eta_1, \ldots, \eta_{R_A}$, and then define A' to be the result of composing A_I and A. It is not hard to see that those two views are equivalent. We note that the view that is used in this section is also the view that is used in the work of [DR06].

It can be verified that A' has the required input size, outputs' number and outputs' size. The non-trivial issues consist of showing that A' has the required rejection ratio, and that it has a super-fast implementation. Below, we discuss those two issues in Sections 3.8.1.1 and 3.8.1.2.

3.8.1.1 The rejection ratio of A'

We show that the rejection ratio of A' is $\Omega\left(\rho_A^2\right)$ in two steps: we first argue that the rejection ratio of the intermediate assignment tester A_I is $\Omega(\rho_A)$, and then we deduce that the rejection ratio of A', which is the composition of A_I with A, is $\Omega(\rho_A^2)$. Both steps are non-trivial and require some work, and we discuss each of them separately.

The rejection ratio of A_I. Consider first the step of showing that the rejection ratio of the intermediate assignment tester A_I is $\Omega(\rho_A)$. Fix an assignment x to φ that is far from any satisfying assignment, and fix a proof string π for A_I. We would like to argue that $x \circ \pi$ is rejected by $\Omega(\rho_A)$ fraction of the circuits $\eta_1, \ldots, \eta_{R_A}$. Observe that since x does not satisfy φ, there exists a circuit ψ_i that rejects $x \circ \pi$. We would now like to argue that due to the rejection ratio of A, it holds that $\Omega(\rho_A)$ fraction of the circuits $\xi_{i,1}, \ldots, \xi_{i,R_A}$ reject $x \circ \pi$. This, in turn, implies that$\Omega(\rho_A)$ fraction of the circuits $\eta_1, \ldots, \eta_{R_A}$ reject x, as required.

However, in order to establish the claim that $\Omega(\rho_A)$ fraction of the circuits $\xi_{i,1}, \ldots, \xi_{i,R_A}$ reject $x \circ \pi$, we need to show that not only that ψ_i rejects $x \circ \pi$, but that $x \circ \pi$ is far from satisfying ψ_i. To this end, we require the decomposition D to have a certain robustness property. We then show how to modify D to satisfy this property, while using the fact that D has matrix access. After robustizing D, the foregoing analysis of the rejection ratio of A_I goes through.

Both the definition of the robustness property of D, and the way to modify D such that it satisfies this property, are described in Section 3.8.2.

The composition of A_I and A. Next, we consider the step of showing that the composition of A_I with A has rejection ratio $\Omega(\rho_A^2)$. To this end, we show that A_I is robust, that is, not only that A_I has rejection ratio $\Omega(\rho_A)$, but it actually has *(expected) robustness* $\Omega(\rho_A)$ (as defined in Section 3.5.4), which implies the required. In order to make A_I robust, we show that A_I has block access, and then apply the robustization technique of Section 3.5.6 to A_I.

It remains to show that A_I has block access. Recall that D has matrix access, and that this means that the tested assignment x and the proof string π_D of D can be arranged in two matrices M_x and M_D, such that each output circuit ψ_i queries only few rows of those matrices. We also define an additional matrix N whose rows are the proof strings of A for each of the invocations of A on a circuit ψ_{i_D}. We now observe that for each of the output circuits η_{i_A} of A_I, the queries of η_{i_A} are contained in a constant number of columns of M_x, M_π, and N. This implies that the assignment tester A_I has $O(1)$-block access, with the blocks being the columns of M_x, M_D, and N, as required.

We still need to show that the queries of each output circuit η_{i_A} are contained in few columns of M_x, M_π, and N. It will be easier to justify this claim after we discuss the efficient implementation of A_I. Thus, we postpone this discussion to Section 3.8.1.3 below.

3.8.1.2 Implementing A_I efficiently

We turn to discuss the efficiency of the implementation of A_I. As was mentioned in Section 3.3.4, the main challenge in coming up with a super-fast implementation of A_I is the need to represent the circuits $\eta_1, \ldots, \eta_{R_A}$ succinctly. Recall that $\eta_{i_A} \overset{\text{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D,i_A}$, and observe that while each of the circuits $\xi_{1,i_A}, \ldots, \xi_{R_D,i_A}$ has a succinct representation, this does not imply that the circuit η_{i_A} has a succinct representation. In particular, a representation of η_{i_A} must represent all the circuits $\xi_{1,i_A}, \ldots, \xi_{R_D,i_A}$, and if those circuits are very different from one another, it may not be possible to have a sufficiently succinct representation that describes all of them simultaneously.

In order to resolve this issue, we use the notion of universal circuit U of Section 3.5.7.2. Recall that a universal circuit U takes as input a representation ζ^{rep} of a circuit ζ, an assignment y to ζ, and verifies that ξ accepts y (using an auxiliary witness z). For simplicity, we ignore the auxiliary witness z for the rest of this overview.

The basic idea of our solution is roughly the following. Fix a tested assignment x and a proof string π_D for D, and let $Q_1^D, \ldots, Q_{R_D}^D$ be the query functions of D that correspond to $\psi_1, \ldots, \psi_{R_D}$ respectively. Now, for every $i_D \in [R_D]$, instead of invoking A on the circuit ψ_{i_D} and on the tested assignment $(x \circ \pi_D)_{|Q_{i_D}^D}$, we invoke A on the circuit U and on tested assignment that consists of $\psi_{i_D}^{\text{rep}}$

and of $(x \circ \pi_D)_{|Q^D_{i_D}}$. Note that the latter invocation of A verifies essentially the same claim as the first invocation of A, namely, that ψ_{i_D} is satisfied by $(x \circ \pi_D)_{|Q^D_{i_D}}$. However, we did gain something: Instead of invoking A on the R_D different circuits $\psi_1, \ldots, \psi_{R_D}$, we now invoke A only on one circuit U, each time with a different tested assignment. Intuitively, the fact that all the invocations of A are made on the same circuit U causes the outputs circuits $\xi_{1,i_A}, \ldots, \xi_{R_D,i_A}$ to be similar to each other, which allows to construct a succinct representation for the circuit $\bigwedge_{i_D=1}^{R_D} \xi_{i_D,i_A}$.

More specifically, this idea is implemented as follows: Let us denote by ξ_{i_A} the i_A-th output circuit of A when invoked on the circuit U. Note that each output circuit ξ_{i_A} may make queries to either ζ^{rep}, y, or to the proof string of A (here we refer to the proof string of the invocation of A on U). We now redefine ξ_{i_D,i_A} to be the circuit that is obtained from ξ_{i_A} by modifying ξ_{i_A} as follows:

1. We hardwire the representation $\psi^{\text{rep}}_{i_D}$ to the inputs of ξ_{i_A} that correspond to queries to ζ^{rep}. That is, if the κ-th query of ξ_{i_A} queries the u-th coordinate of ζ^{rep}, then we hardwire the u-th bit of the description of $\psi^{\text{rep}}_{i_D}$ to the κ-th input gate of ξ_{i_D,i_A}.

2. We redirect the queries of ξ_{i_A} to y to $(x \circ \pi_D)_{|Q^D_{i_D}}$, that is, if the κ-th query of ξ_{i_A} queries the u-th coordinate of y, then the κ-th query of ξ_{i_D,i_A} queries the u-th coordinate of $(x \circ \pi_D)_{|Q^D_{i_D}}$.

3. We redirect the queries of ξ_{i_A} to the proof string of A to the i_D-th row of the matrix N. That is, if the κ-th query of ξ_{i_A} queries the u-th coordinate of the proof string of A, then the κ-th query of ξ_{i_D,i_A} queries the u-th coordinate of the i_D-th row of N.
   Here, we use a slightly *different definition of the matrix* N, and require the i_D-th row of N is contain the proof string that convinces A that the assignment $(\psi^{\text{rep}}_{i_D}, (x \circ \pi_D)_{|Q^D_{i_D}})$ satisfies U.

Next, we construct and output the circuits $\eta_1, \ldots, \eta_{R_A}$, which are defined as before by $\eta_{i_A} \stackrel{\text{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D,i_A}$. It should be clear that this modified version of the circuits η_{i_A} is essentially equivalent to the original construction of those circuits, and hence the previous analysis of the rejection ratio of A_I still applies.

We can now construct a succinct representation $\eta^{\text{rep}}_{i_A}$ of a circuit η_{i_A} as follows. Suppose that the representation $\eta^{\text{rep}}_{i_A}$ is required to retrieve information about a gate g of η_{i_A}. Observe that η_{i_A} consists of circuits $\xi_{1,i_A}, \ldots, \xi_{R_D,i_A}$ that are all identical to ξ_{i_A} except for their input gates, which are determined as listed above. Thus, if g is an internal gate of one of the circuits ξ_{i_D,i_A}, the representation $\eta^{\text{rep}}_{i_A}$ simply invokes the representation $\xi^{\text{rep}}_{i_A}$ of ξ_{i_A} to retrieve the information about the corresponding gate of ξ_{i_A} and outputs it. The only non-trivial case is when g is an input gate one of the circuits ξ_{i_D,i_A}, in which case we consider the following two sub-cases:

1. If g is an input gate that corresponds to a query to ζ^{rep} in the input of U, then we would like to hardwire g to the corresponding bit of the representation $\psi^{\text{rep}}_{i_D}$. To this end, $\eta^{\text{rep}}_{i_A}$ invokes D to compute $\psi^{\text{rep}}_{i_D}$, and hardwires g to the corresponding bit.

2. If g is an input gate that corresponds to a query to y or z in the input of U or to the proof string of A, then we redirect the query to the corresponding coordinate of the matrices M_x, M_D, or N. This redirection can be computed efficiently using the block-access circuit of D.

This concludes the super-fast implementation of A_I.

Remark 3.8.1. The above description ignored the fact that the universal circuit U takes as an additional input an auxiliary witness z. In the actual proof, we will expect the matrix N to contain the auxiliary witnesses in addition to the proof strings of A. In particular, for each i_D, we will require the i_D-th row of N to contain an auxiliary witness for the i_D-th invocation of A. Then, in the construction of ξ_{i_D,i_A}, we will redirect queries of ξ_{i_A} to z the i_D-th row of N.

Remark 3.8.2. We mention that due to technical considerations that were discussed in Section 3.5.7.1, in the actual construction we use the augmented universal circuit $\hat{U}$ instead of the universal circuit U. Recall that $\hat{U}$ is similar to U, but also takes as input multiple copies of the encoding of ζ^{rep} via an error correcting code, as well as multiple copies of y.

3.8.1.3 Showing that the queries of A_I are contained in columns

Recall that in Section 3.8.1.1, we claimed the the queries of every output circuit η_{i_A}queries are contained in a constant number of columns of the matrices M_x, M_D and N, where M_x is the assignment matrix, M_D is the proof matrix of D, and N is the matrix whose rows are the proof strings of all the invocations of A. In this section, we establish this claim and thus conclude this overview.

To this end, recall that the circuits η_{i_A} are defined as $\eta_{i_A} \stackrel{\text{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D.i_A}$, where each circuit $\xi_{i_D.i_A}$ makes at most $s_A = O(1)$ queries to the matrices M_x, M_D, and N. We show that for each circuit $\xi_{i_D.i_A}$, the columns of M_x, M_D, and N to which the queries of $\xi_{i_D.i_A}$ belong depend only on the index i_A and not on the index i_D, and this will imply the required claim. In order to show the latter assertion, recall that the queries of $\xi_{i_D.i_A}$ are obtained from the queries of ξ_{i_A}. Now, for each query of ξ_{i_A} we consider three cases, depending on the part of the input of U at which the query is directed:

- **The query of ξ_{i_A} queries the u-th coordinate of the proof string of A:** In this case, the corresponding query of every circuit ξ_{i_D,i_A} queries the u-th coordinate of the i_D-th row of the matrix N. In other words, the corresponding query of every circuit ξ_{i_D,i_A} always queries the u-th column of the matrix N, regardless of the the index i_D, as required.
- **The query of ξ_{i_A} queries the u-th coordinate of y:** In this case, the corresponding query of each circuit $\xi_{i_D.i_A}$ queries the u-th coordinate of $(x \circ \pi_D)_{|Q_{i_D}}$. Now, since D has matrix access (Definition 3.6.3), it holds that $(x \circ \pi_D)_{|Q_{i_D}}$ consists of few rows of M_x followed by few rows of M_D, where the numbers of rows of M_x and M_D in $(x \circ \pi_D)_{|Q_{i_D}}$ are independent of the index i_D. It can be seen that this implies that the u-th coordinate of $(x \circ \pi_D)_{|Q_{i_D}}$ is always mapped to the same column of M_x or M_D, regardless of the index i_D.
- **The query of ξ_{i_A} queries ζ^{rep}:** In such case, the circuit ξ_{i_D,i_A} does not make any corresponding query, and the corresponding input gate ξ_{i_D,i_A} is hardwired to the corresponding bit of the description of $\psi_{i_D}^{\text{rep}}$. This means that in this case no column of M_x, M_D or N is queried, regardless of the index i_D.

It follows that in all the three cases, the the columns of M_x, M_D, and N to which the queries of $\xi_{i_D.i_A}$ belong depend only on the index i_A and not on the index i_D. We conclude that the queries of every output circuit η_{i_A} are contained in at most $s_A = O(1)$ columns of M_x and M_D, as required.

3.8.2 Robustization of decompositions with matrix access

As discussed in Section 3.8.1.1, in order to establish the rejection ratio of A_I, we require the decomposition D to have a certain robustness property. Specifically, for our argument to go through, D should satisfy the following property: Whenever D is invoked on an assignment x that is *far* from satisfying the input circuit φ and on proof string π, there exists at least one output circuit ψ_{i_A} such that $x \circ \pi$ is *far* from satisfying $x \circ \pi$. This leads to the following definition of "existential robustness" of decompositions.

Definition 3.8.3. We say that a decomposition D has existential robustness ρ if the following holds for every input circuit φ: Let $\psi_1, \ldots, \psi_R$ be the output circuits of D on φ, and let $Q_1, \ldots, Q_R$ be the corresponding query functions. Then, for every assignment x to φ and any proof string π for D, there exists $i \in [R]$ such that

$$\operatorname{dist}\left((x \circ \pi)_{|Q_i}, \mathrm{SAT}(\psi_i)\right) \geq \rho \cdot \operatorname{dist}(x, \mathrm{SAT}(\varphi)). \tag{3.2}$$

Remark 3.8.4. The difference between *existential* robustness and *expected* robustness (of Definition 3.5.6) is that the definition of existential robustness requires Equation (3.2) to hold for some $i \in [R]$, while the definition of expected robustness requires Equation (3.2) to hold for a random $i \in [R]$ in expectation. Note that in general it is unlikely that a decomposition would have expected robustness, since for a decomposition, it is not even guaranteed that a random output circuit will reject, let alone that a random output circuit will be far from being satisfied.

We now observe that we can use the robustization technique of Section 3.5.6 to transform decompositions into existentially robust ones. We actually use the following variant of the procedure of Section 3.5.6, which maintains the matrix access property of the decomposition. This is important since in the construction of the intermediate assignment tester A_I we need D to both be *existentially robust* and have *matrix access*.

Proposition 3.8.5 (Robustization of decompositions with matrix access)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- *Input:*
    1. *A circuit decomposition D for circuits of size n that* has *b-matrix access. Furthermore, we assume that D has outputs' number R, outputs' size s, tester size t, input representation size n^{rep}, and output representation size s^{rep}.*
    2. *A reverse lister RL for D.*
    3. *A block access circuit BA for D.*
- *Output:*
    1. *A circuit decomposition D' for circuits of size n with* existential robustness *$\rho' = \Omega(1/b)$, outputs' number $R' = 2 \cdot R$, outputs' size $s' = O(b \cdot s)$, tester size $t' = O(t) + b \cdot \mathrm{poly}\log(R, s, n, \ell)$, input representation size $n^{\mathrm{rep}\prime} = n^{\mathrm{rep}}$, and output representation size $s^{\mathrm{rep}\prime} = s^{\mathrm{rep}} + b \cdot \mathrm{poly}\log(s)$.*
    2. *A reverse lister RL' for D' of size at most t'.*
    3. *A block access circuit BA' for D'.*

Furthermore, D' has b'-matrix access (for some arbitrarily large b', which in particular may depend on n).

Remark 3.8.6. We stress that the parameter b' of the matrix access of D' may be very large. However, this does not harm our construction, since we only use the decomposition D' of Proposition 3.8.5 in the construction of the intermediate assignment tester A_I (Proposition 3.8.7, stated shortly below), and for this use the value of the parameter b' has no effect.

Proof sketch. Let us denote by ℓ and ℓ' the proof lengths of D and D' respectively. It is not hard to prove that if we apply the procedure of Theorem 3.5.23 to a decomposition D that has b-block access, then the resulting decomposition will have existential robustness $\Omega(1/b)$. This can be done using roughly the same argument used to establish the expected robustness in the proof of Theorem 3.5.23.

For the "furthermore" part, we need to define for D' a partition of the coordinates set $[m+\ell']$ to blocks $B'_1,\ldots,B'_{p'}$, and show that this partition satisfies the requirements of the definition of matrix access (Definition 3.6.3). To this end, let $B_1,\ldots,B_p$ be the partition of $[m+\ell]$ defined by D (that is, by the matrix access of D). We begin the definition of the partition of $[m+\ell']$ for D' by setting the first p blocks $B'_1,\ldots,B'_p$ of D' to be equal to the blocks $B_1,\ldots,B_p$ respectively. It remains to define a partition $B'_{p+1},\ldots,B'_{p'}$ of the set $[m+\ell']\setminus[m+\ell]$ to blocks. Recall that the coordinates in $[m+\ell']\setminus[m+\ell]$ consist of encodings E_j of the blocks B_j of D. The straightforward choice of blocks $B'_{p+1},\ldots,B'_{p'}$ would be to choose the block B'_{p+j} of D' to be the encoding E_j. However, such choice violates the requirement that all the proof blocks would be of the same width, in two ways:

1. The encodings E_j that encode proof blocks of D are wider than the proof blocks of D themselves. This issue can be resolved rather easily, by adding dummy coordinates to the original proof blocks of D such that the resulting blocks will be of the same width as the encodings E_j.

2. The encodings E_j that encode the assignment blocks of D may be much shorter than those that encode the proof blocks of D. This could be the case if the assignment blocks of D are much shorter than the proof blocks of D. In order to resolve this issue, for each encoding E_j that encodes an assignment block, we define the corresponding block B'_{p+j} of D' to consist of many distinct copies of E_j, such that B'_{p+j} has the same width as the encodings of the proof blocks.
   The latter definition of B'_{p+j} can be implemented by redefining the proof string of D' to contain many copies of the encoding E_j. Note that we can not define the block B'_{p+j} to contain multiple queries to the same copy of E_j, because the definition of blocks forbids a block to contain multiple queries to the same coordinate.

After performing the foregoing modifications to the construction of D', as well as few other minor modifications, the decomposition D' can easily be shown to have matrix access. ■

3.8.3 The intermediate assignment tester A_I

In this section we describe the construction of the intermediate assignment tester A_I, which is summarized in the following proposition. We assume that the given circuit decomposition D is existentially robust, and will later obtain this property by applying the robustization technique (Proposition 3.8.5) to D.

For reasons that have to do with the efficient implementation of the reverse lister, we also assume that the assignment tester A is input-uniform (Definition 3.5.11), and we will later obtain this property by applying the generic transformation that was described in Lemma 3.5.13 in Section 3.5.4.

Proposition 3.8.7. *There exists a polynomial time procedure that acts as follows:*

- ***Input:***

  1. *A circuit decomposition D for circuits of size n_D that has outputs' number R_D, outputs' size s_D,* existential robustness *ρ_D, tester size t_D, input representation size n_D^{rep}, and output representation size s_D^{rep}. Furthermore, D is required to have b'-*matrix access *(for arbitrarily large b').*
  2. *An* input-uniform *assignment tester A for circuits of size n_A that has outputs' number R_A, outputs' size s_A, rejection ratio ρ_A, tester size t_A, input representation size n_A^{rep} and output representation size s_A^{rep}.*
  3. *Reverse listers RL_D and RL_A for D and A of sizes at most t_D and t_A respectively.*
  4. *A block-access circuit BA_D for D of size at most t_D.*

5. *Furthermore, the following inequalities should hold:*

$$\begin{aligned} n_A &\geq s_D \cdot s_D^{\text{rep}} \cdot \text{poly}\log(s_D) \\ n_A^{\text{rep}} &\geq \text{poly}\log(s_D) \end{aligned}$$

- ***Output:***

  1. *An assignment tester A_I for circuits of size n_D with outputs' number $R_I \overset{\text{def}}{=} R_A$, outputs' size $s_I \overset{\text{def}}{=} O(R_D \cdot s_A)$, rejection ratio $\Omega(\rho_D \cdot \rho_A)$, tester size*

     $$t_I \overset{\text{def}}{=} O(t_D + s_A \cdot t_A) + s_A \cdot \text{poly}(s_A^{\text{rep}}) + \text{poly}(s_D^{\text{rep}}) + \text{poly}\log(R_D, s_D, n_D, R_A, s_A)$$

     input representation size n_D^{rep}, and output representation size

     $$s_I^{\text{rep}} \overset{\text{def}}{=} O(t_D + s_A \cdot \log s_D) + \text{poly}(s_D^{\text{rep}}) + \text{poly}\log(R_D, s_D, n_D, R_A, s_A).$$

     Furthermore, A_I has s_A-block access.
  2. *An reverse lister RL' for A_I of size at most t_I.*
  3. *A block-access circuit BA_I for A_I of size at most t_I.*

Remark 3.8.8. We note that in the above proposition, we stress that the parameter b' of the matrix access of D does not affect the parameters of A_I. In particular, b' may be *arbitrarily large*, and may depend on n_D. The reason that b' does not affect the parameters of A_I is that for the construction of A_I, we only use the following property from the definition of matrix access:

- he tested assignment and proof string of D can be arranged in matrices, such that every output circuit of ψ_i reads the same number of rows from each matrix, and such that the rows of the assignment matrix precede the rows of the proof matrix in the input of each ψ_i.

The parameter b' of matrix access is only important for purposes of robustization, and in the above proposition, D is already assumed to be robust.

Remark 3.8.9. Throughout this section, we use the family of error correcting codes $\{C_k\}_{k=1}^{\infty}$ whose existence was stated in Fact 3.5.14 in Section 3.5.5. Recall that for each $k \in \mathbb{N}$, the code C_k has message length k. With a slight abuse of notation, for every string $x \in \{0,1\}^*$ we denote $C(x) = C_{|x|}(x)$, and in general, we drop k whenever k is clear from the context.

Recall furthermore that all the codes in the family has relative distance that is lower bounded by a universal constant δ_C, and that for each $k \in \mathbb{N}$, the block length of C_k is denoted by $l_k = O(k)$.

The rest of this section is dedicated to the proof of Proposition 3.8.7. Let D, A, RL_D, RL_A, BA_D be as in the proposition, and let ℓ_D and ℓ_A be the proof lengths of D and A respectively. Observe that since D has matrix access, all its output circuits have the same input length (i.e. queries number), let us denote this length by q_D.

Let $\hat{U} = \hat{U}_{s_D, s_D^{\text{rep}}, q_D}$ be the augmented universal circuit of Corollary 3.5.28, and recall that $\hat{U}$ has size $s_D \cdot s_D^{\text{rep}} \cdot \text{poly}\log(s_D) \leq n_A$ and has representation $\hat{U}^{\text{rep}}$ of size $\text{poly}\log(s_D) \leq n_A^{\text{rep}}$. Furthermore, recall that $\hat{U}$ takes as input a representation ζ^{rep} of a circuit ζ over q_D inputs, strings $c^1, \ldots, c^{\alpha}$, which are supposed to be the encoding $C(\zeta^{\text{rep}})$ of ζ^{rep}, strings $y^1, \ldots, y^{\beta} \in \{0,1\}^{q_D}$ that are supposed to be equal to each other and to be an assignment to ζ, and string z of length $\ell_U \overset{\text{def}}{=} O(s_D)$ that is supposed to "convince" $\hat{U}$ that ζ accepts y^1. More formally, we require that if $y^1 = \ldots = y^{\beta}$ is a satisfying assignment of ζ, then $\hat{U}$ accepts for some choice of z, and otherwise $\hat{U}$ rejects for every choice of z.

In the rest of this section, we describe the action of D on a fixed input circuit φ of size n_D over m inputs that has representation φ^{rep} of size n_D^{rep}. Let $\psi_1, \ldots, \psi_{R_D}$ be the output circuits of D when invoked on φ^{rep}, and let $\psi_1^{\text{rep}}, \ldots, \psi_{R_D}^{\text{rep}}$ and $Q_1^D, \ldots, Q_{R_D}^D$ be the corresponding representations and query functions. Furthermore, let $\xi_1, \ldots, \xi_{R_A}$ be the output circuits of A when invoked on $\hat{U}^{\text{rep}}$, and let $\xi_1^{\text{rep}}, \ldots, \xi_{R_A}^{\text{rep}}$ and $Q_1^A, \ldots, Q_{R_A}^A$ be the corresponding representations and query functions. Note that by our assumption on the input size and input representation size of A it is indeed possible to invoke A on $\hat{U}$.

3.8.3.1 The proof strings of A_I

Fix a satisfying assignment x of φ. We describe the proof string π_I that convinces A_I to accept x. Recall that ℓ_D and ℓ_A denote the proof lengths of D and A respectively, and that ℓ_U is the length of the witnesses of $\hat{U}$. The proof string π_I is of length $\ell_I \stackrel{\text{def}}{=} \ell_D + R_D \cdot (\ell_U + \ell_A)$ and consists of the following parts:

1. π_I contains a proof string π_D that convinces D that x satisfies φ.
2. For each $i_D \in [R_D]$, the proof string π_I contains a witness z^{i_D} that convinces $\hat{U}$ that $(x \circ \pi_D)_{|Q_{i_D}^D}$ satisfies ψ_i.
3. For each $i_D \in [R_D]$, the proof string π_I contains a string $\pi_A^{i_D}$ defined as follows. Let $c = C(\psi_i^{\text{rep}})$ be the encoding of the binary description of $\psi_{i_D}^{\text{rep}}$ via the code C. Then, π_A^i is the string that convinces A that $\hat{U}$ accepts the input which consists of the binary description of $\psi_{i_D}^{\text{rep}}$, of α copies of c, of β copies of $(x \circ \pi_D)_{|Q_{i_D}^D}$, and of z^{i_D}.

We denote by M_x and M_D the assignment matrix and proof matrix in which x and π_D can be arranged due to the fact that D has matrix access, and denote by N the $R_D \times (\ell_U + \ell_A)$ matrix whose i_D-th row is the string $z^{i_D} \circ \pi_A^{i_D}$.

3.8.3.2 The block access circuit BA_I

We describe the behavior of the block access circuit BA_I. Let us denote by a and w_a the number and width of the assignment blocks of D respectively, let us denote by w_D the width of the proof blocks of D, and observe that a, w_a, and w_D can be computed using BA_D. As discussed in the the proof overview, the blocks of A_I consist of the w_a columns of the matrix M_x, the w_D columns of the matrix M_D, and the $\ell_U + \ell_A$ columns of the matrix N.

Recall that BA_I has five modes of operation. It is easy to implement efficiently the first four modes of BA_I, namely, the Number of Blocks mode, the Block to Coordinate mode, the Coordinate to Block mode, and the Number of Assignment Blocks mode, and we do not elaborate on their implementation. It remains to describe the implementation of the Circuit to Blocks mode, in which BA_I is given an index $i_A \in [R_A]$, and is required to output the indices of the blocks that are queried by η_{i_A}, the i_A-th output circuit of A_I.

In the rest of this section, we describe the action of BA_I in Circuit to Blocks mode on a fixed index $i_A \in [R_A]$. As was explained in the overview in Section 3.8.1.3, the output circuit η_{i_A} is defined by $\eta_{i_A} \stackrel{\text{def}}{=} \bigwedge_{i_D \in [R_D]} \xi_{i_D, i_A}$, where the circuits ξ_{i_D, i_A} are obtained by redirecting the queries of ξ_{i_A}. In particular, the columns of M_x, M_D and N that are queried by η_{i_A} are determined by the queries of the circuit ξ_{i_A}, where each column that is queried by η_{i_A} corresponds to one query of ξ_{i_A}. The block access circuit BA_I thus begins its action by computing the queries $\sigma_1, \ldots, \sigma_{q_A} \in [q_D + \ell_A]$ of ξ_{i_A} to the input of $\hat{U}$ and to the proof string of A.

We now explain, for each query σ_h, what is the column of M_x, M_D or N that corresponds to σ_h that is queried by η_{i_A}. The block access circuit BA_I computes the indices of those columns for each of the queries σ_h, and outputs the indices of all of those columns. Fix a query σ_h, and recall that the input of $\hat{U}$ consists of four parts: the description of ζ^{rep} in binary, α strings which are supposed to be equal to the encoding $C(\zeta^{\text{rep}})$ of ζ^{rep}, β supposed copies of the assignment y to ζ, and an auxiliary witness z. The query σ_h may be directed at any of those parts, or to the proof string of A for its invocation on $\hat{U}$. We consider each case separately:

1. **σ_h is directed at z or at the proof string of A:** In this case, if σ_h is directed at the u-th coordinate of z, then η_{i_A} queries the u-th column of N. Similarly, if σ_h is directed at the u-th coordinate of z, then η_{i_A} queries the $(\ell_U + u)$-th column of N.

2. **σ_h is directed at one of the supposed copies of y:** In this case, the circuit η_{i_A} queries one of the columns of M_x or M_D. As explained in the overview, when constructing the circuit ξ_{i_D,i_A}, the queries of ξ_{i_A} to the supposed copies of y are redirected to $(x \circ \pi_D)_{|Q^D_{i_D}}$. Recall that since D has matrix access, the string $(x \circ \pi_D)_{|Q^D_{i_D}}$ consists of rows of M_x followed by rows of M_D, where the numbers of rows of M_x and M_D do not depend on i_D. Let us denote by r_a and r_D the numbers of rows of M_x and M_D respectively that are queried by every output circuit ψ_{i_D} (again, r_a and r_D are independent of i_D). In addition, recall that we denote by w_a and w_D the widths of M_x and M_D.
   Let us view the assignment y as consisting of r_a blocks of length w_a followed by r_D blocks of length w_D. It can be seen that queries of ξ_{i_A} to the first r_a blocks are redirected by ξ_{i_D,i_A} to the corresponding rows of M_x and that the following r_D blocks are always mapped to the rows of M_D. Now, suppose that the query σ_h is directed at the v-th coordinate of one of the supposed copies of y. We consider two sub-cases:

   a) If v is the u-th coordinate of one of the first r_a blocks of y, then the output circuit η_{i_A} queries the u-th column of M_x.

   b) On the other hand, if v is the u-th coordinate of one of the last r_D blocks of y, then the output circuit η_{i_A} queries the u-th column of M_D.

3. **σ_h is directed at the description of ζ^{rep} or at one of its supposed encodings:** In this case, the circuit η_{i_A} does not query any column that corresponds to σ_h, since, as explained in the overview, the queries of ξ_{i_A} to ζ^{rep} and its supposed encodings are hardwired to the descriptions and encodings of the corresponding representations $\psi^{\text{rep}}_{i_D}$. This means that the h-th input gate of each circuit ξ_{i_D,i_A} is hardwired to constants, and hence η_{i_A} does not make any query that corresponds to σ_h.

This concludes the description of the columns that η_{i_A} queries, and the description of BA_I. Observe that A_I indeed satisfies all the requirements for having s_A-block access: every output circuit η_{i_A} queries at most s_A columns since s_A is an upper bound on the number of queries of ξ_{i_A}. Furthermore, all the assignment blocks of A_I are of the same width a. Finally, since D has matrix access, and by the definition of matrix access, it holds that every assignment block of A_I (i.e., column of M_x) contains $(1/3)$ fraction of non-dummy coordinates. It follows that A_I has s_A-block access, as required.

3.8.3.3 The implementation of A_I

We proceed to describe the assignment tester A_I itself. It suffices to describe the circuit mode of A_I, since the query mode of A_I is determined by the Circuit-to-Blocks mode of BA_I (see Section 3.8.3.2), and can be implemented by using BA_I. Thus, it suffices to describe, for every index i_A, what is the

i_A-th output circuit η_{i_A} of A_I, how its representation $\eta_{i_A}^{\text{rep}}$ is implemented, and how the representation $\eta_{i_A}^{\text{rep}}$ is computed by A_I. For the rest of this section, fix an index $i_A \in [R_A]$. We focus on the descriptions of η_{i_A} and of $\eta_{i_A}^{\text{rep}}$, and skip the description of how A_I computes $\eta_{i_A}^{\text{rep}}$, which is straightforward.

The output circuit η_{i_A}. We begin by describing the output circuit η_{i_A}. As explained in the overview, the basic idea that underlies the definition of η_{i_A} is the following: Recall that ξ_{i_A} is the i_A-th output circuit of A when invoked on $\hat{U}$. For every $i_D \in [R_D]$, we obtain a circuit ξ_{i_D,i_A} from ξ_{i_A} by fixing some of the queries of ξ_{i_A} to the description of $\psi_{i_D}^{\text{rep}}$ and its encoding, and by redirecting the other queries of ξ_{i_A} into the matrices M_x, M_D, and N. Next, we observe that all the queries of the circuits ξ_{i_D,i_A} are contained in few columns of M_x, M_D, and N. Finally, we define η_{i_A} to be the circuit that queries the aforementioned columns of M_x, M_D, and N and checks that that all the circuits ξ_{i_D,i_A} are satisfied.

More formally, the output circuit η_{i_A} is defined as follows. The circuit η_{i_A} consists of three parts:

1. Input gates.
2. An output gate which is an AND gate.
3. A collection of R_D circuits, such that the i_D-th circuit ξ_{i_D,i_A} is obtained from ξ_{i_A} by modifying ξ_{i_A} as follows:

    a) Modifying the input gates of ξ_{i_D,i_A} that correspond to queries to the ζ^{rep} part in the input of $\hat{U}$ to be constant gates that contain the description of the output circuit $\psi_{i_D}^{\text{rep}}$.

    b) Modifying the input gates of ξ_{i_D,i_A} that correspond to queries to the supposed encodings of ζ^{rep} in the input of $\hat{U}$ to be constant gates that contain the encoding of the description of $\psi_{i_D}^{\text{rep}}$.

    c) Connecting the output gate of ξ_{i_D,i_A} to the output gate of η_{i_A}.

    d) Connecting each input gate of ξ_{i_D,i_A} to the corresponding input gate of η_{i_A}: As explained above, every query of ξ_{i_A} should be redirected to some coordinate σ of the matrices M_x, M_D, and N, and η_{i_A} queries the column inside which the coordinate σ is found. We thus connect each input gate of ξ_{i_D,i_A} that should be directed to a coordinate σ to the corresponding input gate of η_{i_A} that queries σ.
    More specifically, recall that the assignment to $\hat{U}$ consists of ζ^{rep} and its encodings, supposed copies of an assignment y to ζ, and of an auxiliary witness z. Furthermore, we view y as consisting of r_a blocks of length w_a followed by r_D blocks of length w_D (see Section 3.8.3.2). Let q_{i_A} be the number of input gates o ξ_{i_A}. For each $j \in [q_{i_A}]$, we consider the following cases for the j-th input gate of ξ_{i_A}:

        i. Suppose that the j-th input gate of ξ_{i_A} corresponds to a query to the u-th coordinate of the h-th block of y (for one of the supposed copies of y). Suppose furthermore and the h-th block that is queried by the output circuit ψ_{i_D} is the v-th row of M_x or M_D. Then, we connect the j-th input gate of ξ_{i_D,i_A} to the input gate of η_{i_A} that corresponds to the v-th coordinate of the j-th column that is queried by η_{i_A} (which is the u-th column of M_x or M_D).

        ii. Suppose that the j-th input gate of ξ_{i_A} corresponds to a query to the u-th coordinate of z or to the u-th coordinate of the proof string of A. In this case, we connect the j-th input gate of ξ_{i_A} to the input gate of η_{i_A} that corresponds to the i_D-th coordinate of the j-th column that is queried η_{i_A} (which is the u-th column of N).

This concludes the description of η_{i_A}.

The representation $\eta_{i_A}^{\mathrm{rep}}$. Recall that $\eta_{i_A}^{\mathrm{rep}}$ computes the following functionality: the representation $\eta_{i_A}^{\mathrm{rep}}$ takes as input the index of a gate g of η_{i_A} and an index h, and $\eta_{i_A}^{\mathrm{rep}}$ is required to retrieve the function of g (one of AND, OR, NOT, or one of the constants 0 and 1), the index of the gate from which the h-th incoming wire of g comes, and the index of the gate to which the h-th outgoing wire of g goes. This functionality is straightforward to compute for most of the gates and wires of η_{i_A}, but is non-trivial in the following cases:

1. Suppose that g is one of the constant gates of a circuit ξ_{i_D,i_A} that should be fixed to the description of $\psi_{i_D}^{\mathrm{rep}}$ or its encodings. In this case, the representation $\eta_{i_A}^{\mathrm{rep}}$ should determine whether this gate is the constant 0 or the constant 1. The straightforward way to implement this functionality is to hardwire to the representation $\eta_{i_A}^{\mathrm{rep}}$ the descriptions of all the representations $\psi_{i_D}^{\mathrm{rep}}$ for every $i_D \in [R_D]$. However, this would cause the representation $\eta_{i_A}^{\mathrm{rep}}$ to be of size at least R_D, which is too large.
   We therefore use the following alternative solution: We hardwire into $\eta_{i_A}^{\mathrm{rep}}$ the input representation φ^{rep} and the decomposition D itself. Then, whenever $\eta_{i_A}^{\mathrm{rep}}$ needs the description of $\psi_{i_D}^{\mathrm{rep}}$ for any $i_D \in [R_D]$, the representation $\eta_{i_A}^{\mathrm{rep}}$ invokes D on φ^{rep} in order to generate $\psi_{i_D}^{\mathrm{rep}}$.
   We stress that the fact that $\eta_{i_A}^{\mathrm{rep}}$ can compute the description of $\psi_{i_D}^{\mathrm{rep}}$ using D is a key point in our construction of A_I, and is one of the central ideas of this chapter. What is actually happening here is that we use the fact that the circuits $\psi_1^{\mathrm{rep}}, \ldots, \psi_{i_D}^{\mathrm{rep}}$ are in a way "similar" and "uniform", in the sense that they can all be generated using D.
   We note that when $\eta_{i_A}^{\mathrm{rep}}$ needs the *encoding* of the description of $\psi_{i_D}^{\mathrm{rep}}$, it computes the description of $\psi_{i_D}^{\mathrm{rep}}$ and encodes it via C, which can be done using a circuit of size $\mathrm{poly}\left(\left|\psi_{i_D}^{\mathrm{rep}}\right|\right) = \mathrm{poly}\left(s_D^{\mathrm{rep}}\right)$.

2. Suppose that g is an "input gate" of a circuit ξ_{i_D,i_A}, that is, g is one of the gates of ξ_{i_D,i_A} that are obtained redirecting a query of ξ_{i_A} to an input gate of η_{i_A}. The representation $\eta_{i_A}^{\mathrm{rep}}$ should determine the gates of η_{i_A} from which g has incoming wires, which are all input gates of η_{i_A}. The key issue here is that if g corresponds to a query of ξ_{i_A} that is redirected to M_x or M_D, then in order to determine the relevant input gates of η_{i_A}, we should determine the queries of the circuit ψ_{i_D}. This can be done by invoking the decomposition D on the input representation φ^{rep} in query mode to obtain the queries of ψ_{i_D}, using again the fact that the descriptions of D and φ^{rep} are hardwired into $\eta_{i_A}^{\mathrm{rep}}$.

3. Suppose that g is one of the input gates of η_{i_A}. The representation $\eta_{i_A}^{\mathrm{rep}}$ should determine the gates of η_{i_A} to which g has outgoing wires. All of those gates are "input gates" of circuits ξ_{i_D,i_A}, that is, gates of ξ_{i_D,i_A} that are obtained redirecting a query of ξ_{i_A} to an input gate of η_{i_A}. The challenge is to determine which are the relevant circuits ξ_{i_D,i_A}, that is, for which indices i_D the gate g has an outgoing wire to a gate of ξ_{i_D,i_A}.
   The key issue here is that if g corresponds to a query to a coordinate σ of M_x or M_D, then we need to determine for which indices i_D the output circuit ψ_{i_D} queries the coordinate σ. Fortunately, this can be done easily by invoking the reverse lister RL_D, and therefore we hardwire the description of RL_D into $\eta_{i_A}^{\mathrm{rep}}$ as well. Once we determined those indices i_D, the rest of the implementation is straightforward.
   Interestingly, we note that, except for the proof of Dinur's amplification theorem, this is the only place in this chapter where we use the fact that our assignment testers have super-fast reverse listers[9].

We mention that we also hardwire the description of the circuit ξ_{i_A} and its queries into the representation $\eta_{i_A}^{\mathrm{rep}}$. The representation $\eta_{i_A}^{\mathrm{rep}}$ uses this information to compute the descriptions of the internal gates

[9]We also use this fact in order to upper bound the proof length, as in Section 3.5.2, but this is done only for convenience and can be avoided.

of the circuits ξ_{i_D,i_A}. This concludes the implementation of $\eta_{i_A}^{\mathrm{rep}}$, and the description of the circuit mode of A_I.

Remark 3.8.10. We note that instead of hardwiring the circuit ξ_{i_A} and its queries into the representation $\eta_{i_A}^{\mathrm{rep}}$, we could have also hardwired the assignment tester A into $\eta_{i_A}^{\mathrm{rep}}$ and use it to compute the queries of ξ_{i_A}. However, hardwiring A into $\eta_{i_A}^{\mathrm{rep}}$ would cause the output representation size of A_I to be at least t_A, which in turn would have caused problems in the proof of the tensor product lemma (see Remark 3.8.11).

3.8.3.4 The reverse lister of A_I

We turn to describe the construction of the reverse lister RL_I of A_I. This construction is not central to the understanding of the proof of the tensor product lemma, and can be skipped at first reading. We also note that the construction is straightforward except for one subtle point in which we use the assumption that the assignment tester A is input-uniform.

Fix a coordinate $k \in [m + \ell_I]$, and consider the action of RL_I on input representation φ^{rep} and coordinate k. For simplicity, we assume that k belongs to the matrix M_x, while the cases where k belongs to one of the matrices M_D and N can be handled similarly.

Recall that the columns of M_x and M_D that an output circuit η_{i_A} queries are determined as follows (see also Section 3.8.3.2): for each query of ξ_{i_A} to one of the β supposed copies of y in the input of $\hat{U}$, the output circuit η_{i_A} queries either a column of M_x or a column of M_D. The exact column of M_x or M_D that is queried depends on the place of the query within the supposed copy of y. In particular if the query belongs to one of the first r_a blocks of y (each of length w_a), then η_{i_A} queries a column of M_x, and if the query belongs to one of the last r_D blocks of y (each of length w_D), then η_{i_A} queries a column of M_D.

Now, let u be the index of the column of M_x to which the coordinate k belongs. It can be seen that there exist $\beta \cdot r_a$ coordinates $\sigma_1, \ldots, \sigma_{\beta \cdot r_a}$ in the input of $\hat{U}$ such that an output circuit η_{i_A} queries the u-th column of M_x exactly once for each query of ξ_{i_A} to a coordinate σ_i. Observe that the coordinates $\sigma_1, \ldots, \sigma_{\beta \cdot r_a}$ can be computed efficiently from u, r_a, w_a, r_D, w_D, and $q_D \overset{\mathrm{def}}{=} r_a \cdot w_a + r_D \cdot w_D$. We turn to describe the action of RL_I on the coordinate k in each of its modes:

1. **Counting mode:** In this mode, RL_I is given as input φ^{rep} and k, and is required to output $|\mathbf{RevList}_{A_I,\varphi}(k)|$. By the above discussion, it can be seen that

$$|\mathbf{RevList}_{A_I,\varphi}(k)| = \sum_{h=1}^{\beta \cdot r_a} \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_h)\right|$$

   Now, since the assignment tester A is assumed to be input-uniform, it holds that

$$\left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right| = \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_2)\right| = \ldots \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_{\beta \cdot r_a})\right|$$

   Thus, in order to compute $|\mathbf{RevList}_{A_I,\varphi}(k)|$, the reverse lister RL_I invokes RL_A on $\hat{U}$ and σ_1 to determine $\left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right|$, and outputs $\beta \cdot r_a \cdot \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right|$.

2. **Retrieval mode:** In this mode, RL_I is given as input φ^{rep}, k, and $v \in [|\mathbf{RevList}_{A_I,\varphi}(k)|]$, and is required to output the v-th element (i, κ) of $\mathbf{RevList}_{A_I,\varphi}(k)$. Recall that $|\mathbf{RevList}_{A_I,\varphi}(k)| = \beta \cdot r_a \cdot \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right|$, so the index v can be viewed as a pair of indices (h, v') where $h \in [\beta \cdot r_a]$ and $v' \in \left[\left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right|\right]$. The reverse lister RL_I begins by invoking RL_A in retrieval mode

to compute the v'-th element (i_A, κ_A) of $\mathbf{RevList}_{A,\hat{U}}(\sigma_h)$.
Now, observe that the desired value of i is i_A. Furthermore, observe that the place κ of the query to the coordinate k within the input of η_i can be computed efficiently using BA_I. Hence, the reverse lister RL_I computes this value of κ and outputs (i, κ) (where $i \stackrel{\text{def}}{=} i_A$).

3. **Reverse retrieval mode:** In this mode, RL_I is given as input φ^{rep}, k, and a pair $(i, \kappa) \in \mathbf{RevList}_{A_I,\varphi^{\text{rep}}}(k)$, and is required to output the index v such that (i, κ) is the v-th element of $\mathbf{RevList}_{A_I,\varphi}(k)$. Recall that κ is a coordinate in the input of η_i, and suppose that it belongs to the j-th column that is queried by η_i. Thus, the query of ξ_i that corresponds to this column is the j-th query of ξ_i that is not directed at the representation ζ^{rep} or to its supposed encodings in the input of $\hat{U}$ - suppose that this is the κ_A-th query of ξ_i. Now, the reverse lister RL_I first invokes the assignment tester A in query mode on κ_A to determine the coordinate σ_h at which the κ_A-th query of ξ_i is directed. Then, RL_I invokes RL_A in reverse retrieval mode to find the index v' such that (i, σ_h) is the v'-th element of $\mathbf{RevList}_{A,\hat{U}}(\sigma_h)$. Finally, RL_I outputs $v \stackrel{\text{def}}{=} (h-1) \cdot \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right| + v'$.

This concludes the description of the action of RL_I on a coordinate k of M_x. The cases where k belongs to the matrix M_D or to the matrix N are handled similarly. We note that if k belongs to the matrix N, instead of M_x or M_D, then the implementation is actually simpler, since the coordinates $\sigma_1, \ldots, \sigma_{\beta \cdot r_a}$ in the input of $\hat{U}$ are replaced with only one coordinate σ.

3.8.3.5 The parameters of A_I

It is easy to verify that A_I has the input size, input representation size, outputs' number, and outputs' size that are stated in the lemma. In addition, as noted in Section 3.8.3.2, the assignment tester A_I indeed has s_A-block access. We now argue that A_I has the correct output representation size

$$s_I^{\text{rep}} \stackrel{\text{def}}{=} O\left(t_D + s_A \cdot \log s_D\right) + \operatorname{poly}\left(s_D^{\text{rep}}\right) + \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A\right).$$

To see it, note that the actions of a representation $\eta_{i_A}^{\text{rep}}$ consist of invoking the decomposition D to compute a representation $\psi_{i_D}^{\text{rep}}$ and its queries (which is a reason for the $O(t_D)$ term), computing the encoding of $\psi_{i_D}^{\text{rep}}$ (which is the reason for the $\operatorname{poly}(s_D^{\text{rep}})$ term), invoking the reverse lister RL_D and the block access circuit BA_D (which is another reason for the $O(t_D)$ term), and performing calculations on numbers in the sets $[R_D]$, $[s_D]$, $[n_D]$, $[\ell_D]$, $[\ell_A]$ and $[s_A]$ (which is the reason for the term $\operatorname{poly}\log(R_D, s_D, n_D, \ell_D, \ell_A, s_A)$). Furthermore, the representation $\eta_{i_A}^{\text{rep}}$ contains the description of ξ_{i_A} and all of its queries into the input of $\hat{U}$, which is the reason for the $O\left(s_A \cdot \log s_D\right)$ term.

Next, we claim that A_I indeed has tester size

$$\begin{aligned} t_I &= s_I^{\text{rep}} + O\left(s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s_A^{\text{rep}}\right) \\ &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s_A^{\text{rep}}\right) + \operatorname{poly}\left(s_D^{\text{rep}}\right) + \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A\right) \end{aligned}$$

To see it, note that the actions of A_I on index i_A consist of computing the representation $\xi_{i_A}^{\text{rep}}$(which accounts for a term of $O(t_A)$), computing the description of ξ_{i_A} from $\xi_{i_A}^{\text{rep}}$ (which is the reason for the $s_A \cdot \operatorname{poly}\left(s_A^{\text{rep}}\right)$ term), computing the queries of ξ_{i_A} (which accounts for a $O(s_A \cdot t_A)$ term), and outputting the resulting representation η_{i_A} (which accounts for the s_I^{rep} term).

It remains to show that the rejection ratio of A_I is $\Omega\left(\rho_D \cdot \rho_A\right)$. Let x be an assignment to φ that is ε-far from any satisfying assignment, and let π_I be a proof string for A_I. As before, we view π_I as consisting of a proof string π_D for D, of a collection of witnesses z^i for $\hat{U}$, and of a collection of proof strings π_A^i for A. By the existential robustness of D, there exists $i_D \in [R_D]$ such that $(x \circ \pi_D)_{|Q_{i_D}^{D,\varphi}}$ is

$(\rho_D \cdot \varepsilon)$-far from satisfying ψ_{i_D}. Now, let $c = C(\psi^{\text{rep}}_{i_D})$ be encoding of the binary description of $\psi^{\text{rep}}_{i_D}$, and consider the following assignment y to $\hat{U}$:

$$y \stackrel{\text{def}}{=} \psi^{\text{rep}}_{i_D} \circ \underbrace{c \circ \cdots \circ c}_{\alpha} \circ \underbrace{(x \circ \pi_D)_{|Q^{D,\varphi}_{i_D}} \circ \ldots \circ (x \circ \pi_D)_{|Q^{D,\varphi}_{i_D}}}_{\beta} \circ z^{i_D},$$

It is not hard to see that y is $\Omega\left(\rho_D \cdot \varepsilon\right)$-far from satisfying $\hat{U}$, and therefore for at least $\Omega\left(\rho_A \cdot \rho_D \cdot \varepsilon\right)$ fraction of the circuits ξ_{i_A} reject $y \circ \pi^{i_D}_A$. It follows that at least $\Omega\left(\rho_A \cdot \rho_D \cdot \varepsilon\right)$ fraction of the circuits ξ_{i_D,i_A} reject their corresponding assignment, and this, in turn, implies that at least $\Omega\left(\rho_A \cdot \rho_D \cdot \varepsilon\right)$ fraction of the circuits η_{i_A} reject $x \circ \pi_I$, as required.

3.8.4 Proof of the Tensor Product Lemma

In this section, we complete the proof of the tensor product lemma, by describing the procedure of the tensor product lemma that constructs the assignment tester A from the decomposition D and the assignment tester A'. We note that the description of the procedure and the analysis of the parameters is technical, and contains no new ideas.

When given as input a circuit decomposition D and an assignment tester A as in the lemma, as well as circuits RL_D, BA_D, and RL_A, the procedure takes the following steps:

1. The procedure applies the robustization technique of Proposition 3.8.5 to D, thus obtaining an existentially robust decomposition D^{rob}.
2. The procedure applies the transformation of Lemma 3.5.13 to A, thus obtaining an input-uniform assignment tester A^{uni}.
3. The procedure applies Proposition 3.8.7 to D^{rob} and A^{uni} to construct the intermediate assignment tester A_I.
4. The procedure applies Theorem 3.5.23 (the robustization theorem) to A_I, yielding a robust assignment tester $A_{I'}$.
5. the procedure composes $A_{I'}$ with A using the composition theorem (Theorem 3.5.7), thus obtaining the assignment tester A'.
6. The procedure outputs A' and the corresponding reverse lister RL', which is obtained in the process of constructing A'.

We turn to analyze the parameters of A' by analyzing the parameters obtained in each of the foregoing steps:

1. By Proposition 3.8.5, it holds that D^{rob} is a circuit decomposition for circuits of size n_D with outputs' number $R_{D^{\text{rob}}} \stackrel{\text{def}}{=} 2 \cdot R_D$, outputs' size $s_{D^{\text{rob}}} \stackrel{\text{def}}{=} O(b \cdot s_D)$, existential robustness $\rho_{D^{\text{rob}}} \stackrel{\text{def}}{=} \Omega\left(1/b\right)$, tester size $t_{D^{\text{rob}}} \stackrel{\text{def}}{=} O\left(t_D\right) + b \cdot \text{poly}\log\left(R_D, s_D, n_D\right)$, input representation size n^{rep}_D, and output representation size $s^{\text{rep}}_{D^{\text{rob}}} \stackrel{\text{def}}{=} s^{\text{rep}}_D + b \cdot \text{poly}\log\left(s_D\right)$. Furthermore, D has b'-matrix access for some arbitrary b'.
2. By Lemma 3.5.13, it holds that A^{uni} is an assignment tester for circuits of size n_A with with outputs' number $R_{A^{\text{uni}}} = 2 \cdot R_A$, outputs' size $s_{A^{\text{uni}}} \stackrel{\text{def}}{=} O(s_A)$, rejection ratio $\frac{1}{4} \cdot \rho_A$, tester size $t_{A^{\text{uni}}} = t_A + \text{poly}\log\left(n_A, R_A\right)$, input representation size n^{rep}_A, and output representation size $s^{\text{rep}}_{A^{\text{uni}}} = s^{\text{rep}}_A + \text{poly}\log(n_A)$.

3. By Proposition 3.8.7, it holds that A_I is an assignment tester for circuits of size n_D with outputs' number $R_I \overset{\text{def}}{=} R_{A^{\text{uni}}} = O(R_A)$, outputs' size $s_I \overset{\text{def}}{=} O(R_{D^{\text{rob}}} \cdot s_{A^{\text{uni}}}) = O(R_D \cdot s_A)$, rejection ratio $\Omega\left(\rho_{D^{\text{rob}}} \cdot \rho_{A^{\text{uni}}}\right) = \Omega(\rho_A/b)$, tester size

$$\begin{aligned} t_I &\overset{\text{def}}{=} O\left(t_{D^{\text{rob}}} + s_{A^{\text{uni}}} \cdot t_{A^{\text{uni}}}\right) + s_{A^{\text{uni}}} \cdot \operatorname{poly}\left(s^{\text{rep}}_{A^{\text{uni}}}\right) \\ &\quad + \operatorname{poly}\left(s^{\text{rep}}_{D^{\text{rob}}}\right) + \operatorname{poly}\log\left(R_{D^{\text{rob}}}, s_{D^{\text{rob}}}, n_{D^{\text{rob}}}, R_{A^{\text{uni}}}, s_{A^{\text{uni}}}\right) \\ &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s^{\text{rep}}_A, \log n_A, \log R_A\right) \\ &\quad + \operatorname{poly}\left(s^{\text{rep}}_D, b, \log s_D\right) + b \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$

input representation size n^{rep}_D, and output representation size

$$\begin{aligned} s^{\text{rep}}_I &\overset{\text{def}}{=} O\left(t_{D^{\text{rob}}} + s_{A^{\text{uni}}} \cdot \log s_{D^{\text{rob}}}\right) + \operatorname{poly}\left(s^{\text{rep}}_{D^{\text{rob}}}\right) + \operatorname{poly}\log\left(R_{D^{\text{rob}}}, s_{D^{\text{rob}}}, n_D, R_{A^{\text{uni}}}, s_{A^{\text{uni}}}\right) \\ &= O\left(t_D + s_A \cdot \log\left(s_D \cdot b\right)\right) + \operatorname{poly}\left(s^{\text{rep}}_D, b, \log s_D\right) + \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right). \end{aligned}$$

Furthermore, A_I has $s_{A^{\text{uni}}}$-block access. We also note that by our assumption on the input size and input representation size of A, it is indeed legal to apply Proposition 3.8.7 to D^{rob} and A^{uni} (recall that this proposition requires a lower bound on the input size and input representation size of A).

4. By Theorem 3.5.23, it holds that $A_{I'}$ is an assignment tester for circuits of size n_D with outputs' number $R_{I'} \overset{\text{def}}{=} 2 \cdot R_I = O(R_A)$, outputs' size $s_{I'} \overset{\text{def}}{=} O(s_A \cdot s_I) = O\left(R_D \cdot s_A^2\right)$, robustness $\rho_{I'} \overset{\text{def}}{=} \Omega\left(\rho_I / s_{A^{\text{uni}}}\right) = \Omega\left(\rho_A / b \cdot s_A\right)$, tester size

$$\begin{aligned} t_{I'} &\overset{\text{def}}{=} O\left(s_{A^{\text{uni}}} \cdot t_I\right) + s_{A^{\text{uni}}} \cdot \operatorname{poly}\log\left(R_I, s_I, n_D\right) \\ &= O\left(s_A \cdot t_D + s_A^2 \cdot t_A\right) + \operatorname{poly}\left(s_A, s^{\text{rep}}_A, s^{\text{rep}}_D, b\right) \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, n_A\right), \end{aligned}$$

input representation size n^{rep}_D, and output representation size

$$\begin{aligned} s^{\text{rep}}_{I'} &\overset{\text{def}}{=} s^{\text{rep}}_I + s_{A^{\text{uni}}} \cdot \operatorname{poly}\log\left(s_I\right) \\ &= O\left(t_D\right) + \operatorname{poly}\left(s^{\text{rep}}_D, b, \log s_D\right) + s_A \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right). \end{aligned}$$

5. Finally, by the composition theorem (Theorem 3.5.7), it holds that A' is an assignment tester for circuits of size n_D with outputs' number $R' \overset{\text{def}}{=} 2 \cdot R_I \cdot R_A = O\left(R_A^2\right)$, outputs' size $s' \overset{\text{def}}{=} O(s_A)$, rejection ratio $\rho' \overset{\text{def}}{=} \frac{1}{4} \cdot \rho_{I'} \cdot \rho_A = \Omega\left(\rho_A^2 / b \cdot s_A\right)$, tester size

$$\begin{aligned} t' &\overset{\text{def}}{=} O\left(t_{I'} + t_A\right) + \operatorname{poly}\log\left(n_D, R_{I'}, \ell_{I'}, R_A, \ell_A\right) \\ &= O\left(s_A \cdot t_D + s_A^2 \cdot t_A\right) + \operatorname{poly}\left(s_A, s^{\text{rep}}_A, s^{\text{rep}}_D, b\right) \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, n_A\right), \end{aligned}$$

input representation size n^{rep}_D, and output representation size $s^{\text{rep}\prime} \overset{\text{def}}{=} s^{\text{rep}}_A + \operatorname{poly}\log(s_A)$. Furthermore, RL' is of size at most t'. We also note that applying the composition theorem to $A_{I'}$ and A is legal, since by our assumption on A, its input size is larger than $s_{I'}$, and its input representation size is larger than $s^{\text{rep}}_{I'}$.

The required result follows.

Remark 3.8.11. We would like to highlight the fact that the output representation size s^{rep}_I of A_I does not depend on the tester size t_A of A, even though the tester size t_I of A_I does depend on t_A. This is an important and non-trivial fact that results from our particular implementation of A_I. To see why this fact is important, observe that had s_I^{rep} depended on t_A, we would not have been able to perform the last composition step, since the output representation size $s_{I'}^{\mathrm{rep}}$ of $A_{I'}$ would have been greater than the input representation size n_A^{rep} of A.

While it is tempting to try to solve this problem using the input representation lemma (Lemma 3.5.13), it is not clear that this solution would have worked, since this would have introduced a non-trivial dependency into our iterative construction in Section 3.6.

Chapter 4

Combinatorial PCPs with Short Proofs

4.1 Introduction

4.1.1 Background and Our Results

A PCP (Probabilistically Checkable Proof) is a proof system that allows checking the validity of a claim by reading only a constant number of bits of the proof. The PCP theorem asserts the existence of PCPs of polynomial length for any claim that can be stated as membership in an **NP** language. In this chapter, we consider the length of the proofs, and provide a combinatorial construction of PCPs that are almost as short as the ones obtained from the algebraic constructions.

Let L be a language in **NP**, and recall that there is a polynomial-time algorithm V that verifies the membership of a string x in L when given an additional **NP**-witness. Let $t : \mathbb{N} \to \mathbb{N}$ denote the running time of V. The original PCP theorem asserts that in order to verify that $x \in L$, the PCP verifier needs to use a proof of length $\text{poly}\,(t(|x|))$. However, using algebraic techniques, one can construct PCP verifiers that use a proof of length only $t \cdot \text{poly}\log(t)$ [BSS08, Din07]. It is not known whether one can construct such PCPs using a combinatorial approach such as Dinur's[1].

In this chapter, we present an (almost) combinatorial construction of PCPs that use proofs of length $t \cdot (\log t)^{O(\log\log t)}$, thus coming very close to the state of the art algebraic constructions. Namely, our main result is the following

Theorem 4.1.1 (Main theorem)**.** *For every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in$ **NTIME**(t), there exists a PCP verifier for L with proof length $t(n) \cdot (\log(t(n)))^{O(\log\log t(n))}$, query complexity $O(1)$, and rejection probability $\Omega(1)$.*

In order to prove Theorem 4.1.1, we develop a few generic PCP techniques that may be of independent interest, and are discussed in Section 4.1.2.

Our use of polynomials. As mentioned above, our construction is only "almost" combinatorial. The only exception is that our construction does use low degree polynomials at one point. However, our use of polynomials is a very restricted one, and is confined to the construction of error correcting codes with a certain simple property. Specifically, we only use polynomials to construct a triplet (C_A, C_B, C_M) of linear codes that have the following "multiplication property":

- For every two codewords $c_A \in C_A$ and $c_B \in C_B$, it holds that $c_A \cdot c_B$ is a codeword of C_M, where $c_A \cdot c_B$ is obtained by coordinate-wise multiplication of c_A and c_B. The multiplication is done in the finite field over which the codes are linear.

[1]We mention that the construction of PCPs that have proof length $t \cdot \text{poly}\log(t)$ uses Dinur's combinatorial techniques in addition to the algebraic techniques. Still, the main part of this construction is algebraic.

Such a triplet (C_A, C_B, C_M) can easily be constructed using low-degree polynomials. Moreover, if the codes C_A, C_B, and C_M are allowed to have quadratic length, then they can also be constructed without using polynomials (see Section 2.3.2 of Chapter 2). However, for our purposes we need the codes C_A, C_B, and C_M to have quasi-linear length, and we do not know how to construct such codes without using polynomials. Still, a combinatorial construction of such codes is conceivable.

Extensions of our result. It can be shown that our PCPs have randomness complexity of $\log t + O(\log^2 \log t)$, which matches the length of the proofs. In addition, as in previous works in the area, our construction of PCPs can be extended to yield the stronger notion of PCPPs [BSGH+06, DR06]. We do not elaborate on those claims in this preliminary version.

4.1.2 Our techniques

Below, we sketch the main steps of our construction and the main techniques that we use.

Constructing PCPs from linear PCPPs. Our first step is reducing the construction of PCPs to the construction of a simpler object, called linear PCPPs [BSHLM09]. Informally, a linear PCPP is a verifier that, when given a linear subspace $W \subseteq \mathbb{F}^n$ and oracle access to a vector $w \in \mathbb{F}^n$, verifies that $w \in W$ by making few queries to w and to an alleged proof. In other words, a linear PCPP is the restriction of the notion of a PCPP [BSGH+06, DR06] to the verification of membership in linear subspaces.

We show that *any* construction of a linear PCPP implies a construction of a full-fledged PCP, with a poly-logarithmic loss in the parameters. The construction of the full-fledged PCP is generic, and uses the linear PCPP as a black box. We believe that this construction is interesting in its own right, and may be useful for future works[2].

Our construction of PCPs from linear PCPPs is performed by combining the linear PCPP with the multiplication codes that were discussed in Section 4.1.1. Intuitively, the multiplication codes allow us to go from verifying linear claims to verifying non-linear claims.

Robustization via tensor product codes. Our next step is to note, following [BSGH+06, DR06, BSS08, Din07], that it suffices to construct a linear PCPP that makes $\tilde{O}(\sqrt{n})$ queries to its oracle and has proof length $\tilde{O}(n)$. Such a linear PCPP can then be composed with itself for $O(\log \log n)$ times to yield[3] a linear PCPP with a constant number of queries and proof length $n \cdot (\log n)^{O(\log \log n)}$.

In order for us to be able to apply such composition, the linear PCPP is required to have a property called robustness [BSGH+06, DR06]. The robustness property was achieved in several previous works by a technique called "robustization" [BSGH+06, DR06] (also "alphabet reduction" or "parallelization"). The robustization technique allows one to transform every PCP with a "block-access" property into a robust PCP. Specifically, the latter property may be viewed as follows: It is required that the PCP proof can be partitioned to blocks, such that the PCP verifier always queries only a constant number of the blocks.

[2]We note that the work of [BSS08] has shown a stronger result, namely that one can construct a full-feldged PCP from a linear PCPP that *can only verify membership in a Reed-Solomon code* (rather than a general linear subspace). However, their construction is signicantly more complicated than ours, and relies heavily on algebraic machinery.

We also note that other works in this area (e.g. [AS98, ALM+98, BSGH+06]) start by reducing the construction of a full-edged PCP to the construction of a PCP for a specic algebraic problem. However, in all of those works, the latter algebraic problem is **NP**-complete, while the problem we consider (checking membership in a linear subspace) is in **P**. Hence, there is a fundamental dierence between those reductions and ours.

[3]We mention that after the composition one also needs to apply a query reduction technique and the gap amplification theorem of Dinur.

In our context, we do not know how to construct a PCP that satisfies the foregoing block-access property. We therefore generalize the robustization technique such that it can be applied to PCPs which satisfy a weaker requirement.

To this end, instead of partitioning the proof to blocks, we arrange the proof coordinates in a matrix. From this point of view, the foregoing block-access property may be viewed as restricting the PCP verifier to reading a constant number of rows of the matrix. We now generalize the robustization technique by allowing the PCP verifier to query *both rows and columns* of the aforementioned matrix, as long as it queries a *constant number* of rows and columns.

Both the standard robustization technique and our generalization use error correcting codes. The standard robustization technique transforms a "block-access PCP" into a robust one by encoding each of the blocks by an error correcting code. In our a generalization, we transform the "row/column-access PCP" into a robust one by encoding the corresponding matrix via a robust tensor product code [BSS06]. This means, roughly, that we first encode the rows of the matrix by an error correcting code, and then encode the columns of the new matrix by an error correcting code.

We stress that this generalization of the robustization method is generic, and may be useful for future constructions of PCPs.

Constructing linear PCPPs that make $\tilde{O}(\sqrt{n})$ queries. It remains to construct a linear PCPP that makes $\tilde{O}(\sqrt{n})$ queries to its oracle, has proof length $\tilde{O}(n)$, and satisfies the relaxed robustization requirement discussed above. To this end, we first define a notion called "Simultaneous Linear Verifier" (abbreviated SLV). Informally, an SLV is a verifier V that when given as input $\sqrt{n}$ subspaces $W^1, \ldots, W^{\sqrt{n}} \subseteq \mathbb{F}^{\sqrt{n}}$ and oracle access to $\sqrt{n}$ vectors $w^1, \ldots, w^{\sqrt{n}} \in \mathbb{F}^{\sqrt{n}}$, verifies that all the claims of the form $w^i \in W^i$ hold *simultaneously.* Here, "simultaneously" means that *if for at least one index* $i \in [m]$ it holds that w^i is far from W^i, the verifier V rejects with significant probability[4].

We proceed by showing how to construct the required linear PCPP while assuming the existence of an SLV that uses $\tilde{O}(\sqrt{n})$ queries and proofs of length $\tilde{O}(n)$. This is done by transforming a claim of the form $w \in W$ (for $w \in \mathbb{F}^n$) to an equivalent collection of claims of the form $w^i \in W^i$ (for $w^i \in \mathbb{F}^{\sqrt{n}}$). The latter transformation is performed by embedding the claim $w \in W$ on a structured routing network, following [BFLS91, PS94] and in particular Chapter 3.

Finally, we show how to construct the required SLV. We begin by considering the special case in which all the subspaces W^i are equal (say, to a subspace W), and show how to handle this case using error correcting codes. Finally, we show how to decompose the general case to a constant number of instances of the foregoing special case, and handle those instances as before. The latter decomposition is performed via a novel application of routing networks and of the multiplication codes discussed in Section 4.1.1.

In the author's opinion, the introduction of SLVs and their construction is the most interesting part of this chapter.

Organization of this chapter. In Section 4.2 we recall the preliminaries that are required for this chapter. In Section 4.3, we define the notion of linear PCPPs, and show to construct a PCP based on a linear PCP. In Section 4.4, we show our generalization of the robustization technique. In Section 4.5, we show the construction of linear PCPPs with $\tilde{O}(\sqrt{n})$ queries. Finally, in Section 4.6, we show how to use the foregoing tools to construct the required PCPs and prove the main theorem (Theorem 4.1.1).

[4]Note that the notion of SLV differs from a linear PCPP: if we used a linear PCPP verifier to verify all of the claims $w^i \in W^i$ simultaneously, then the verifier would have rejected only if the vectors $w^1, \ldots, w^{\sqrt{n}}$ were far from $W^1, \ldots, W^{\sqrt{n}}$ *on average.* On the other hand, a linear PCPP is capable of handling a a subspace of $\mathbb{F}^n$ and not just a collection of subspaces of $\mathbb{F}^{\sqrt{n}}$.

4.2 Preliminaries

For any $n \in \mathbb{N}$ we denote $[n] \overset{\text{def}}{=} \{1 \ldots, n\}$. For a string $x \in \{0,1\}^n$ and a set $S \subseteq [n]$, we denote by $x_{|S}$ the projection of x to the coordinates in S.

4.2.1 PCPs

Below we give the formal definition of PCPs.

Definition 4.2.1. Let $L \subseteq \{0,1\}^*$ be a language, and let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0,1)$. A PCP verifier V for L with query complexity q, proof length ℓ, and rejection probability ρ, is a probabilistic oracle machine that satisfies the following requirements:

1. On every input $x \in \{0,1\}^*$ and every sequence of coin tosses, V makes at most $q(|x|)$ queries to its oracle.
2. **Completeness:** For every $x \in L$, there exists a string $\pi \in \{0,1\}^{\ell(|x|)}$ such that $\Pr[V^\pi(x) \text{ accepts}] = 1$.
3. **Soundness:** For every $x \notin L$ and every a string $\pi \in \{0,1\}^{\ell(|x|)}$, it holds that $\Pr[V^\pi(x) \text{ rejects}] \geq \rho(|x|)$.

4.2.2 Error Correcting Codes

All the error correcting codes that we consider in this extended abstract are *binary linear codes*, to be defined next. A (linear) code C with message length k and block length ℓ is an injective linear function from $\{0,1\}^k$ to $\{0,1\}^\ell$ (where linearity is over GF(2)). We will sometimes identify C with its image $C(\{0,1\}^k)$. Specifically, we will write $c \in C$ to indicate the fact that there exists $x \in \{0,1\}^k$ such that $c = C(x)$. In such case, we also say that c is a codeword of C. The rate R_C of the code C is the ratio k/ℓ.

For any two strings $x, y \in \{0,1\}^\ell$, the relative Hamming distance between x and y is the fraction of coordinates on which x and y differ, and is denoted by $\delta(x,y) \overset{\text{def}}{=} |\{x_i \neq y_i : i \in [\ell]\}|/\ell$. The relative distance of a code C is defined as $\delta_C \overset{\text{def}}{=} \min_{c_1 \neq c_2 \in C}\{\delta(c_1, c_2)\}$. For every code $C : \{0,1\}^k \to \{0,1\}^\ell$ and a string $w \in \{0,1\}^\ell$ we denote $\delta(w, C) \overset{\text{def}}{=} \min_{c \in C}\{\delta(w,c)\}$. We say that w is ε-far from C (resp. ε-close to C) if $\delta(w,C) > \varepsilon$ (resp. $\delta(w,C) \leq \varepsilon$).

A code $C : \{0,1\}^k \to \{0,1\}^\ell$ is said to be systematic if for every $x \in \{0,1\}^k$ it holds that $C(x)_{|[k]} = x$. By Gaussian elimination, every linear code may be assumed to be systematic without loss of generality. All the codes in this chapter are assumed to be systematic.

4.2.3 Routing networks

In our construction of PCPs, we use a special kind of graphs called permutation routing networks (see, e.g., [Lei92]). In order to motivate this notion, let us think of the vertices of the graph as computers in a network, such that two computers can communicate if and only if they are connected by an edge. Suppose that there is some set S of computers in the network such that each computer in S needs to send a message to some other computer in S, and furthermore that each computer in S needs to receive a message from exactly one computer in S (in other words, the mapping from source computers to target computers is a permutation). Then, the property of the routing network says that we can route the messages in the network such that each computer in the network forwards exactly one message. Formally, we use the following definition of routing networks.

Definition 4.2.2. A routing network of order n is a graph $G = (V, E)$ along with a special set of vertices $S \subseteq V$ of size n, such that the following requirement holds: For every permuation σ on S, there exists a set $\mathcal{P}$ of vertex-disjoint paths in G that connect each $v \in S$ to $\sigma(v) \in s$.

Routing networks were studied extensively in the literature of distributed computing, and several constructions of efficient routing networks are known. In particular, we use the following fact on routing networks, whose requirements are satisfied by several constructions.

Fact 4.2.3 (see, e.g, [Lei92]). *There exists an infinite family of routing networks $\{G_n\}_{n=1}^{\infty}$, the network G_n being of order n such that the following properties hold.*

1. *G_n has $\tilde{O}(n)$ vertices.*
2. *The edges of G_n can be colored using 4 colors such that no two edges of the same color share a vertex.*
3. *There exists an algorithm that on input n, runs in time* $\mathrm{poly}(n)$ *and outputs G_n.*
4. *There exists a polynomial time algorithm that when given as input G_n and a bijection $\sigma : S \to T$ outputs a set $\mathcal{P}$ of vertex-disjoint paths that connect each $v \in S$ to $\sigma(v) \in T$.*

4.3 PCPs and Linear PCPPs.

In this section we define the notion of linear PCPPs, and show how to construct a full-fledged PCP using a linear PCPP as a building block.

4.3.1 Linear PCPPs

We begin by defining the notion of linear PCPPs (originally defined in [BSHLM09]). For simplicity, in this overview we restrict our attention to linear PCPPs over GF(2), but it is possible to generalize this definition to larger fields, and in fact we do use such a generalization in the actual proof.

Informally, a linear PCPP is a verifier that checks that a vector w satisfies a linear assertion by reading a small part of w, and of an alledged proof. Little more specifically, a linear PCPP is an oracle machine that takes as explicit input a linear subspace $W \subseteq \{0,1\}^m$, gets oracle access to a vector $w \in \{0,1\}^m$ and to an additional string $\pi \in \{0,1\}^*$, accepts with probability 1 if w belongs to W, and rejects with significant probability if w is far from W. The subspace W is represented by a *linear circuit* [Val77], to be defined next.

A linear circuit (over GF(2)) is a circuit that contains only gates that compute the XOR of their inputs[5]. The size of the circuit is defined to be the number of wires in the circuit. Note that every output of the circuit is a linear *non-affine* function[6] of the inputs, and that every linear function over GF(2) can be computed by such circuits. We note that in some previous works the definition of linear circuits is little different, and allows the circuits to compute affine functions. The reason we choose not to allow affine functions is that it allows us to state a stronger result (Theorem 4.3.6).

We will usually consider linear circuits that have multiple outputs. We say that a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ accepts an input $w \in \{0,1\}^m$ if $\varphi(w) = \bar{0} \in \{0,1\}^t$. Observe that the set of inputs accepted by a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ is a linear subspace of $\{0,1\}^m$ of dimension at least $m - t$. We denote the latter subspace by W_φ, and say that φ accepts W_φ.

Definition 4.3.1 (Variant of [BSHLM09]). Let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0,1)$. A linear PCPP verifier V with query complexity q, proof length ℓ, and rejection ratio ρ, is a probabilistic oracle machine that satisfies the following requirements:

[5]In the generalization of this definition to larger fields, every gate computes a linear combination of its inputs.

[6]The function is non-affine because we did not allow constant gates.

1. V takes as input a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ of size n, and gets oracle access to a vector $w \in \{0,1\}^m$ as well as to an additional string $\pi \in \{0,1\}^{\ell(n)}$.
2. On every input circuit φ and every sequence of coin tosses, V makes at total number of at most $q(n)$ queries to its oracle. The queries are made non-adaptively.
3. **Completeness:** For every $x \in W_\varphi$, there exists a string $\pi \in \{0,1\}^{\ell(n)}$ such that $\Pr\left[V^{x,\pi}(\varphi) \text{ accepts}\right] = 1$.
4. **Soundness:** For every $x \in \{0,1\}^m$ that is ε-far from W_φ and every string $\pi \in \{0,1\}^{\ell(n)}$, it holds that $\Pr\left[V^{x,\pi}(\varphi) \text{ rejects}\right] \geq \rho(n) \cdot \varepsilon$.

Remark 4.3.2. In order to be able to compose linear PCPPs, we also need to require that after the linear PCPP gets the answers it gets to its queries, it may only apply *linear* predicates to those answers. However, we ignore this issue in this extended abstract.

Remark 4.3.3. Our notion of linear PCPP is a special case of the notion of PCPP of [BSGH$^+$06, DR06] (a.k.a assignment tester). It is also related to the notion of linear inner verifier of [GS06].

4.3.2 Constructing PCPs from linear PCPPs

We turn to show a construction of PCPs that uses linear PCPPs as a building block. This is formalized in Theorem 4.3.6 below.

Before getting to the theorem and its proof, we note that the proof relies on the existence of "multiplication codes", which are codes with a certain multiplication property. For simplicity, in this overview we assume the existence of multiplication codes over $\{0,1\}$, while in the actual proof we use such codes over a larger field, and construct them using the Reed-Solomon code. More specifically, we assume the following:

Assumption 4.3.4. *For every $k \in \mathbb{N}$, there exists a triplet of systematic codes (C_A, C_B, C_M) of constant rate and relative distance, such that C_A and C_B have message length k, and such that the following holds: For every $c_A \in C_A$ and $c_B \in C_B$, it holds that $c_A \cdot c_B \in C_M$, where $c_A \cdot c_B$ is the coordinate-wise multiplication of c_A and c_B.*

Remark 4.3.5. In fact, our construction would work even if the rate and distance of C_A, C_B, and C_M were only $1/\text{poly}\log k$. However, we preferred to use constants for simplicity and because it can be achieved (over larger fields). While we only know how to construct such codes by using polynomials, it is plausible that such codes can be constructed in other ways.

In the rest of this section, we sketch the proof of the following theorem, based on Assumption 4.3.4.

Theorem 4.3.6. *Suppose that there exists a linear PCPP verifier V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$. Then, for every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in$ **NTIME**(t), there exists a PCP verifier V' for L with proof length $O\left(t + \ell\left(O(t)\right)\right)$ query complexity $O\left(q\left(O(t)\right) \cdot \log n\right)$,, and rejection probability $\Omega\left(\rho\left(O(t)\right)\right)$.*

Remark 4.3.7. The extra $\log n$ factor in the query complexity comes from the use of large fields in the construction of the multiplication codes, and is not apparent in this extended abstract. In our final construction of PCPs this $\log n$ factor is reduced in a later stage using known query reduction techniques.

Intuitively, the idea that underlies the proof of Theorem 4.3.6 is that the multiplication codes allow us to go from verifying linear claims to verifying non-linear claims. Little more specifically, we begin by noting that it suffices to construct a PCP for the language of satisfiable systems of quadratic equations. Given a system of quadratic equations E, the PCP proof will be expected to contain a satisfying assignment x to the system, as well as a string y that should contain the value under x of each quadratic term that appears in E. Now, E may be viewed as a system of *linear* equations over x and y, so we can use the linear PCPP verifier V on x and y to check that it is satisfied. It remains to verify that y is indeed consistent with x, and the point is that this consistency can be verified by using the multiplication codes together with V.

To be more concrete, the verifier V' acts as follows. Let C_A, C_B, and C_M be as in Assumption 4.3.4. The verifier V' constructs two projections of the string x, denoted a and b, such that $y = a \cdot b$ (where the multiplication is coordinate-wise). The verifier V' expects the prover to provide the encodings $a' \stackrel{\text{def}}{=} C_A(a)$ and $b' \stackrel{\text{def}}{=} C_B(b)$, as well as the vector $c' \stackrel{\text{def}}{=} C_A(a) \cdot C_B(b) \in C_M$. Note that the string y is a substring of c', since C_A, C_B, and C_M are systematic. Now, V' invokes V to verify that the system E is satisfied as a linear system over x and y, where y is retrieved from c'.

Finally, the verifier V' checks that the vectors a', b', and c' provided by the prover are constructed as expected. To this end, observe that $C_A(a)$ and $C_B(b)$ are obtained from x via a linear transformation, and hence V' verifies the consistency of a' and b' with x simply by invoking V. In order to verify the consistency of c', the verifier V' checks that the vectors c' and $a' \cdot b'$ agree on a random coordinate. The soundness of the latter check is proved using the relative distance of C_M.

Remark 4.3.8. One may argue that using the language of quadratic equations is an "algebraic step". However, it is not hard to adjust the proof to work with the language CIRCUITSAT instead of quadratic equations. We chose the language of quadratic equations for technical convenience.

We now turn to give a somewhat more rigorous proof description of We begin by recalling that it suffices to construct a PCP for proving that a system of quadratic equations is satisfiable, due to the **NP**-completeness of the quadratic equations problem, and via the efficient reduction of Turing machines to circuits of [PF79]. We turn to construct a PCP verifier V' for quadratic equations. For simplicity, we consider only equations that do not contain constant terms - it is not hard to extend the proof to the general case.

The proof strings of V'. Let E denote a satisfiable system of quadratic equations, and let x be a satisfying assignment of E. We describe how to construct the proof string of V' that corresponds to E and x.

We first arrange the quadratic terms that appear in E in an arbitrary fixed order, and define y to be a string whose i-th bit is the value of the i-th quadratic term (of E) under x. Observe that E induces a system of linear equations over x and y (since by assumption E does not contain constant terms).

Let (C_A, C_B, C_M) be the multiplication codes of Assumption 4.3.4. We now define two strings a and b as follows: The i-th bit of a is the value under x of the first factor of the i-th quadratic term in E (according to some arbitrary order of the factors), and b is defined similarly for the second factors. We define c^a and c^b to be the encodings of a and b via C_A and C_B respectively, and define $c^m = c^a \cdot c^b$. Note that since C_A, C_B, and C_M are systematic, for each $i \in [|y|]$ it should hold that $y_i = (c^m)_i$.

Finally, we define c^x to be the encoding of x via C_A. The proof string of V' that corresponds to E and x is now defined to be the concatenation of c^x, c^a, c^b, c^m, and an additional string π to be described next. The string π is a proof string of the linear PCPP verifier V that proves that:

1. c^x, c^a, c^b, and c^m are all legal codewords of the corresponding codes.

2. x and y satisfy the system of linear equations induced over them by E, where x and y are retrieved from c^x and c^m, respectively.

3. The strings a and b are consistent with the string x, where x, a, and b are retrieved from c^x, c^a, and c^b, respectively. Here, consistency means that the occurences of the value of each x_i in a and b are indeed consistent with x_i.

The behavior of V'. Fix a system of quadratic equations E. We describe the action of V' on input E when given access to a purported proof π' of the form $c^x \circ c^a \circ c^b \circ c^m \circ \pi$. The verifier V' performs the following checks:

1. V' invokes V to perform the linear checks listed above.
2. V' chooses uniformly at random $i \in [|c^a|]$ and checks that $(c^a)_i \cdot \left(c^b\right)_i = (c^m)_i$.

The completeness of V' is clear. Proving the soundness of V' and analyzing its other parameters is not hard and is deferred to the full version of this work.

4.4 A Generalization of the Robustization Technique

Our construction of PCPs uses the composition technique in order to reduce the query complexity of our linear PCPPs (see details in Section 4.6). In order to apply composition, we need our linear PCPPs to have a property called robustness [BSGH+06, DR06]. As discussed in Section 4.1.2, this property was achieved in previous works by a technique called "robustization", which can not be applied to our linear PCPPs. In order tor resolve this issue, we generalize the robustization technique so it can be applied to our linear PCPPs.

In Section 4.4.1 below, we define the notion of robustness and describe the standard robustization method. Then, in Section 4.4.2, we describe our generalization of the robustization method.

4.4.1 Background on robustness and robustization

Below we provide some background on the robustness property, and a more detailed information may be found in [BSGH+06, DR06]. All the definitions and results discussed here are stated in terms of linear PCPPs, although they were originally developed for PCPs. Informally, a linear PCPP verifier V is robust if, when V is given oracle access to a vector x that is far from satisfying the linear assertion being checked, the answers to V's queries are far from making V accept.

Definition 4.4.1 (Views and accepting views)**.** Let V be a linear PCPP verifier with proof length ℓ. Fix an input circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ of size n, a vector $x \in \{0,1\}^m$, and a proof string $\pi \in \{0,1\}^{\ell(n)}$. For every possible invocation of V, we refer to the answers that V gets to its queries as the view of V. If V accepts, then the corresponding view is said to be an accepted view.

Definition 4.4.2 (Robustness)**.** Let $\rho : \mathbb{N} \to (0,1)$, and let. V be a linear PCPP verifier. The verifier V is said to have robustness ρ if whenever x is ε-far from W_φ, the expected relative distance of the view of V from the closest accepted view is at least $\rho(n) \cdot \varepsilon$.

Remark 4.4.3. Observe that if a linear PCPP has robustness ρ then it in particular has rejection ratio ρ. Thus, when discussing robust linear PCPPs we do not mention their rejection ratio.

A common technique in the PCP literature, called "robustization" (also "alphabet reduction" or "parallelization"), allows transforming every PCP with a certain query structure into a robust PCP. Specifically, the technique requires the following property:

Definition 4.4.4. We say that a linear PCPP verifier V has κ-block access if for every circuit φ, the coordiantes of the oracle $x \circ \pi$ can be partitioned into blocks, such that V always queries at most κ blocks.

The robustization technique yields the following result.

Theorem 4.4.5 (Robustization, special case proved in [BSGH+06, DR06])**.** *Suppose that there exists a linear PCPP verifier V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$, which has κ-block access. Then, there exists a linear PCPP verifier V' with query complexity $O(q)$, proof length $O(\ell)$, and* robustness $\Omega(\rho/\kappa)$.

The verifier V' is constructed roughly as follows. The prover is expected to provide V' with the encoding of each of the blocks via an error correcting code. The verifier V' emulates the verifier V, but whenever V queries a block, the verifier V' also queries the purported encoding of the block, and verifies that it is indeed the legal encoding of the block.

To see that V' is robust, observe that whenever V rejects, at least one of the blocks that V queries must be modified in order to make V accept. Hence, in order to make V' accept, the prover must modify both the aforementioned block *and its encoding*. However, modifying the latter encoding to another legal encoding requires flipping many coordinates, and therefore the view of V' is far from any accepting view.

4.4.2 Our generalized robustization

Unfortunately, we do not know how to make our linear PCPPs to have κ-block access. In order to resolve this issue, we define a relaxation of the block access property, which we call row/column access, and prove a more general robustization theorem that applies to PCPs with the latter property.

Definition 4.4.6. We say that a linear PCPP verifier V has κ-row/column access if for every circuit φ and every strings x, π, the string $x \circ \pi$ can be arranged in a matrix M, such that V queries at most κ rows and columns of M.

We now have the following result.

Theorem 4.4.7 (Generalized robustization)**.** *Suppose that there exists a linear PCPP verifier V with query complexity q, proof length ℓ, and rejection ratio ρ, which has κ-row/column access. Then, there exists a linear PCPP verifier V' with query complexity $O(q)$, proof length $O(\ell)$, and* robustness $\Omega(\rho/\kappa)$.

It is tempting to try to prove Theorem 4.4.7 using the same argument as for Theorem 4.4.5. Such a construction would ask the prover to provide the encoding of every row and column of M via an eror correcting code. Then, the verifier V' would read the encoding of every row and column that are queried by V and check that it is a legal encoding.

However, the foregoing argument fails. The reason is that the purported encodings of the rows and columns of M may be *inconsistent*. That is, the prover might provide us with encodings whose encoded messages do not agree on the intersections of the rows and columns. Such an inconsistency may fail the soundness of V'.

In order to resolve this issue, we use tensor codes, to be defined next.

Definition 4.4.8 (Tensor codes, see, e.g., [Sud01, Lect. 6 (2.4)])**.** Let $C : \{0,1\}^k \to \{0,1\}^n$ be a code. The tensor code C^2 is a code with block length n^2, whose codewords are exactly the $n \times n$ matrices N such that every row and every column of N is a codeword of C.

Fact 4.4.9. *Let $C : \{0,1\}^k \to \{0,1\}^n$ be a code of relative distance δ. Then, C^2 has message length k^2 and relative distance δ^2. Furthermore, if C is systematic, then the message encoded by a codeword N of C^2 is the top-left $k \times k$ submatrix of N.*

The critical property of tensor codes that we use is the following: Let N' be a matrix that is close to a codeword N of C^2. Then, for most of the rows and columns of N', the closest codeword of C is the corresponding row or column of N. In particular, this means that the messages encoded by most rows and columns of N' are consistent with each other.

Now, we construct the verifier V' such that the prover is required to provide the encoding of the matrix M via tensor code C^2. The soundness analysis goes roughly as follows. Let N' be the purported encoding of M provided by the prover. We first note that if N' is close to C^2, then we can use the foregoing property of C^2 to argue that the rows and columns are mostly consistent, and perform roughly the same analysis as in Theorem 4.4.5.

It remains to verify that N' is close to C^2. To this end, we require C^2 to have a *robust tester*. This means, roughly, that there exists a *robust* linear PCPP verifier that is only capable of verifying the assertion that a matrix is close to C^2 (rahter than verifying any linear assertion). Tensor codes satisfying this requirement can be constructed combinatorially using a few methods (see [BSS06, DSW06, BSV09b, BSV09a]). The verifier V' will invoke the robust tester of C^2 to verify that N' is close to C^2, and then proceed as before.

Remark 4.4.10. The foregoing description oversimplifies things a little. In particular, note that the property of C^2 only guarantees that *most of* the rows and columns are consistent, and not all of them. Thus, in general it could be the case that V always queries the inconsistent rows and columns. In order to handle this issue, we begin the construction by invoking the expander-replacement technique of [PY91], which guarantees that V queries rows and columns according to the uniform distribution, and then proceed as before.

4.5 Construction of Linear PCPPs with $\sqrt{n}$ Queries

In this section, we explain how to construct a linear PCPP that have proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\text{poly}\log n$, and that has $O(1)$-row/column access. Later, in Section 4.6, we will compose this linear PCPP with itself for $O(\log\log n)$ times to obtain a linear PCP with constant query complexity.

This section is organized as follows: In Section 4.5.1 below, we define an auxiliary object called simultaneous linear verifier (SLV). Then, in Section 4.5.2, we describe how to construct linear PCPPs based on the existence of SLVs. Finally, in Section 4.5.3, we show a construction of SLVs, which in turn yield linear PCPPs.

In the author's opinion, the introduction of SLVs, as well as the ways they are constructed and used for constructing linear PCPPs, are the most interesting part in this chapter.

4.5.1 Simultaneous linear verifiers

Intuitively, an SLV is a variant of a linear PCPP verifier, which verifies $\sqrt{n}$ linear assertions of size $\sqrt{n}$ instead of verifying one linear claim of size n. The verification is *simultaneous*, in the sense that if *at least one* of the $\sqrt{n}$ linear assertions is far from being correct, then the SLV rejects with noticeable probability.

To see the difference between SLVs and linear PCPPs, observe that if a linear PCPP verifier is invoked on a linear assertion of size n that consists of $\sqrt{n}$ concatenated linear assertions of size $\sqrt{n}$, then the verifier is required to reject only if the $\sqrt{n}$ linear assertions are far from being correct *on average* (i.e., a *random* assertion is far from being correct).

Definition 4.5.1. Let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0, 1)$. A simultaneous linear verifier (SLV) V with query complexity q, proof length ℓ, and rejection ratio ρ, is a probabilistic oracle machine that satisfies the following requirements:

1. V takes as input $\sqrt{n}$ linear circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}} : \{0,1\}^m \to \{0,1\}^t$ of size at most $\sqrt{n}$, and gets oracle access to $\sqrt{n}$ vectors $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$ as well as to an additional string $\pi \in \{0,1\}^{\ell(n)}$.

2. For every input circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ and every sequence of coin tosses, V makes at most $q(n)$ non-adaptive queries to its oracle.

3. **Completeness:** For every $x_1 \in W_{\varphi_1}, \ldots, x_{\sqrt{n}} \in W_{\varphi_{\sqrt{n}}}$, there exists a string $\pi \in \{0,1\}^{\ell(n)}$ such that $\Pr\left[V^{x_1,\ldots,x_{\sqrt{n}},\pi} \text{ accepts}\right] = 1$.

4. **Soundness:** For every $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$ such that for some $i \in [\sqrt{n}]$ it holds that x_i is ε-far from W_{φ_i}, and for every string $\pi \in \{0,1\}^{\ell(n)}$, it holds that $\Pr\left[V^{x_1,\ldots,x_{\sqrt{n}},\pi} \text{ rejects}\right] \geq \rho(n) \cdot \varepsilon$.

Recall that our goal is to construct linear PCPPs that have the *row/column access* property (Definition 4.4.6), which is important since we use it later to make the linear PCPP robust. To this end, we will also want our SLVs to have row/column access as well. Recall that a linear PCPP verifier is said to have κ-row/column access if the coordinates of its oracle can be arranged in a matrix M, such that the verifier always queries at most κ rows and columns of M. The same definition can be applied to SLVs. Actually, we will want our SLVs to satisfy the following stronger property, which will be used shortly below in Theorem 4.5.2.

Definition 4.5.2. An SLV V is said to have κ-row/column access with assignments sub-matrix if it has $O(1)$-row/column access, and furthermore satisfies the following requirement: Let M be the matrix with respect to which the SLV has row/column access. Then, the matrix whose rows are the assignments $x_1, \ldots, x_{\sqrt{n}}$ is a sub-matrix of the matrix M, where $x_1, \ldots, x_{\sqrt{n}}$ are as in Definition 4.5.1.

4.5.2 Linear PCPPs from simultaneous linear verifiers

As discussed above, the importance of SLVs is that we are able to construct a linear PCPP using an SLV as a building block. More specifically, we have the following result.

Theorem 4.5.3. *Suppose that there exists an SLV V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$. Then, there exists a linear PCPP verifier V' with query complexity $O\left(q\left(\tilde{O}(n)\right)\right)$, proof length $O\left(\tilde{O}(n) + \ell\left(\tilde{O}(n)\right)\right)$, and rejection probability $\Omega\left(\rho\left(\tilde{O}(n)\right)\right)$. Furthermore, if V has κ-row/column access with assignments sub-matrix, then V' has $O(\kappa)$-row/column access.*

Proof sketch. The basic idea of the proof is to decompose the linear assertion that should be verified to a collection of $\tilde{O}\left(\sqrt{n}\right)$ linear assertions of size $\tilde{O}(\sqrt{n})$ by using a decomposition method of Section **??** in Chapter 3, and then applying the SLV to verify the latter linear assertions. Details follow.

We begin by recalling the notion of circuit decomposition of Chapter 3 in our terms. A circuit decomposition may be thought as a linear PCPP verifier that satisfies only the following trivial soundness requirement: For every $x \notin W_\varphi$ and every π, it is required that the decomposition rejects with *non-zero* probability. The circuit decomposition D of Chapter 3 has proof length $\tilde{O}(n)$ and query complexity $\tilde{O}(\sqrt{n})$.

For our purposes, it is more convenient to define the decomposition a little differently: Instead of viewing D as a probabilistic oracle machine, we view D as a deterministic algorithm that takes a linear circuit φ as an input, and outputs linear circuits $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$ of size $\tilde{O}(\sqrt{n})$, where each ψ_i takes as input a part of the string $x \circ \pi$ (where the inputs of different ψ_i's may overlap). The linear circuits $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$ correspond to all the possible tests that D may perform. In this terminology, the soundness of D requires that for every $x \notin W_\varphi$ and every π, there exists at least one circuit ψ_i that does not accept its input (which is derived from x and π). We mention that the term "circuit decomposition" was chosen since D may be thought as decomposing φ into smaller circuits $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$.

It is important to note that, as argued in Chapter 3, the circuit decomposition D can be strengthened to have the following robustness property: for every x that is far from W_φ and for every π, there exists at least one circuit ψ_i whose input is far from any string accepted by ψ_i. This is done using the standard robustization technique.

Now, we construct V' as follows. When given as input a circuit φ, the verifier V' first applies the circuit decomposition D of Chapter 3 to φ, resulting in a collection of circuits $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$. Then, the verifier V' invokes the SLV V to verify that all the circuits in $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$ are satisfied simultaneously. The idea that underlies the soundness analysis is that if V' is given oracle access to a vector x that is far from W_φ, then for at least one circuit ψ_i, the input of ψ_i is far from a satisfying assignment to ψ_i. Therefore, the SLV V is expected to reject with high probability.

It remains to show the "furthermor" part of the theorem. To this end, we note that, as observed in Chapter 3, the decomposition D has $O(1)$-block access, so the coordinates of its oracle can be arranged in a matrix N, such that each circuit ψ_i queries at most $O(1)$ rows of N. Now, let M be the matrix with respect to which V has row/column access with assignments sub-matrix, so V always queries at most $O(1)$ rows and columns of M. By assumption, the assignments to the circuits $\psi_1, \ldots, \psi_{\tilde{O}(\sqrt{n})}$ form a submatrix of M. When combining this property with the block access property of D, we get that whenever V queries a row of M, the verifier V' can emulate this query by querying $O(1)$ rows of N, and an additional row from the proof string of V. Moreover, whenever V queries a column of M, the verifier can emulate this query by querying one column of N, and an additional column from the proof string of V. By a careful choice of the implementation details of V', the foregoing considerations can be used to guaranteed that V' has $O(1)$-row/column access. ■

4.5.3 Construction of simultaneous linear verifiers

In the rest of this section, we finish the construction of linear PCPPs by showing the following result.

Theorem 4.5.4. *There exists an SLV with proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\text{poly}\log n$.*

By combining the latter result with Theorem 4.5.3, we obtain as a corollary the required linear PCPPs.

Corollary 4.5.5. *There exists a linear PCPP with proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\text{poly}\log n$.*

We first describe how to construct an SLV for the simple special case in which all the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ are the same. Then, we show how to construct an SLV for a more interesting case, which we call colorable constraint systems, by reducing it to few instances of the foregoing simple case. Finally, we show how to reduce the general case to the case of colorable constraint systems.

4.5.3.1 A simple case

We begin with describing how to construct an SLV V for case in which all the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ are the same. Let $W \subseteq \{0,1\}^m$ denote the subspace that is accepted by the circuit $\varphi_1 = \varphi_2 = \ldots = \varphi_{\sqrt{n}}$.

In order to verify that $x_1, \ldots, x_{\sqrt{n}} \in W$, we consider a $\sqrt{n} \times m$ matrix A whose rows are exactly the vectors $x_1, \ldots, x_{\sqrt{n}}$. Let A' be the matrix obtained by encoding each *column* of M by some systematic linear code C with constant rate and relative distance. The proof string π of V is expected to contain all the rows of A' that do not belong to A.

Observe that if $x_1, \ldots, x_{\sqrt{n}} \in W$, then all the rows of A' belong to W, including the rows that are not in A. This is true since all the rows of A' are linear combinations of rows of A.

Now, when given oracle access to vectors $x_1, \ldots, x_{\sqrt{n}}$ and to a purported proof π, the verifier V acts as follows. The verifier V views its oracle as a purported matrix A', and performs the following checks:

1. V chooses a row of A' uniformly at random and checks that it belongs to W.
2. V chooses a column of A' uniformly at random and checks that it is a legal codeword of C.

V accepts if and only if both checks accept. It is easy to see that the query complexity and proof length of V are as required. Furthermore, it is easy to see that V has 1-row/column access with assignments sub-matrix - here, V has 1-row/column access with respect to the matrix $M \overset{\text{def}}{=} A'$, and the assignments $x_1, \ldots, x_{\sqrt{n}}$ form the matrix A, which is indeed a sub-matrix of A'. It remains to analyze the rejection ratio of V. Suppose that V is given oracle access to a purported matrix M', and that one of the rows of M' (say, the i-th row) is ε-far from W.

First, as a warm-up, assume that all the columns of A' are legal codewords of C. Observe that in this case, if at least one row of A' does not belong to W, then at least δ_C fraction of the rows of A' do not belong to W, where δ_C is the relative distance of C: To see it, let $w^{\perp}$ be any vector that is orthogonal to W but is not orthogonal to the i-th row of A' (such $w^{\perp}$ must exist, since by assumption the i-th row of A' does not belong to W). Now, let $v = A' \cdot w^{\perp}$, and note that

1. v is a codeword of C, since v is a linear combination of columns of A', which are codewords of C.

2. v is a non-zero vector, since the i-th row of A' is not orthogonal to $w^{\perp}$, and hence the i-th coordinate of v is non-zero.

We conclude that v is a non-zero codeword of C, and therefore at least δ_C fraction of its coordinates are non-zero. However, for each non-zero coordinate of v, the corresponding row of M' is not orthogonal to $w^{\perp}$ and thus does not belong to W. Hence, at least δ_C fraction of the rows of A' do not belong to W, and V therefore rejects in this case with probability at least δ_C.

Next, assume that some of the columns of A' are not legal codewords of C, and let T denote the set of those columns. If the density of T is at least ε, then V rejects with probability at least ε, and we are done. Suppose otherwise. Let A'_0 be the matrix obtained from A' by removing the columns in T, and let W_0 be the vector space obtained by projecting the vectors of W to the coordinates in $[m] \backslash T$.

Now, observe that the i-th row of A'_0 can not belong to W_0, or otherwise the i-th row of A' would have belonged to W. Thus, A'_0 is a matrix whose columns are all legal codewords of C, and such that one of the rows of A'_0 does not belong to W_0. We therefore conclude as before that at least δ_C fraction of the rows of A'_0 do not belong to W_0. The latter assertion implies that at least δ_C fraction of the rows of A' do not belong to W, and hence V rejects with probability at least δ_C, as required.

4.5.3.2 The case of colorable constraint systems

We proceed to show a construction of an SLV for circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ that are a colorable constraints system (CCS), to be defined below. We will later show that any sequence of circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ can be transformed to a CCS.

Informally, we say that a collection of circuits $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ forms a CCS if the subspaces $W_{\varphi_1}, \ldots, W_{\varphi_k}$ can be described by a collection $\mathcal{S}$ of linear constraints that can be "legally colored" using few colors. We say that a coloring of the constraints in $\mathcal{S}$ is legal if constraints of the same color do not share coordinates. The idea that underlies our construction of an SLV for a CCS is that in a CCS can be decomposed to few monochromatic systems of constraints, such that each system can be reduced to the simple case discussed above. Some details follow.

For the following definitions and constructions, it will be convenient for us to describe linear subspaces by sets of linear constraints, defined next, rather than by linear circuits.

Notation 4.5.6. Let $S \subseteq \{0,1\}^m$. We say that a subspace $W \subseteq \{0,1\}^m$ is the subspace described by S if W contains exactly the vectors that are orthogonal to all the vectors in S. We refer to the vectors of S as constraints. We say that a constraint $s \in S$ touches a coordinate $i \in [m]$ if $s_i = 1$.

Observe that every linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ can be transformed in polynomial time to a set S_φ of constraints which describes W_φ such that $|S_\varphi| = t$. This is done simply by taking, for each output of φ, the constraint which touches exactly the coordinates that affect this output. We turn to define the notion of CSS.

Definition 4.5.7 (Colorable constraints system)**.** Let $\chi \in \mathbb{N}$, let $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ be linear circuits, and let $S_1, \ldots, S_k \subseteq \{0,1\}^m$ be the corresponding sets of constraints. We say that $\varphi_1, \ldots, \varphi_k$ form a χ-colorable constraints system (abbreivated χ-CCS) if the union $\mathcal{S} \stackrel{\mathrm{def}}{=} S_1 \cup \ldots \cup S_k$ satisfies the following requirement: The constraints in $\mathcal{S}$ can be colored by χ colors, such that no two constraints in $\mathcal{S}$ of the same color touch the same coordinate.

A construction of SLV for a CCS. We turn to describe a construction of an SLV V for a CCS. Suppose that the verifier V is given as input circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ and is also given oracle access to vectors $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$. Let $S_1, \ldots, S_{\sqrt{n}}$, and χ be as in Definition 4.5.7.

Our strategy is to construct for each color $c \in [\chi]$ a collection of vectors $x_1^c, \ldots, x_{\sqrt{n}}^c \in \{0,1\}^m$ and a subspace $U^c \subseteq \{0,1\}^m$ such that the following holds: The circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ accept $x_1, \ldots, x_{\sqrt{n}}$ (respectively) if and only if the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ all belong to U^c *for every color* $c \in [\chi]$. The point is that verifying that $x_1^c, \ldots, x_{\sqrt{n}}^c$ belong to U^c can be done as in the simple case of Section 4.5.3.1. The prover will be expected to provide the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$, as well as additional information that allows verifying their consistency with $x_1, \ldots, x_{\sqrt{n}}$.

For each color $c \in [\chi]$, we define the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ to be the vectors obtained from $x_1, \ldots, x_{\sqrt{n}}$ by zeroing every coordinate that is not touched by a constraint of color c. More formally, for each x_j, we define $\left(x_j^c\right)_i \stackrel{\mathrm{def}}{=} (x_j)_i$ if S_j contains a constraint of color c that touches the coordinate i, and $\left(x_j^c\right)_i \stackrel{\mathrm{def}}{=} 0$ otherwise. We define the subspace $U^c \subseteq \{0,1\}^m$ to be the subspace that is described by all the constraints in $\mathcal{S}$ of color c. It is not hard to see that he circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ accept $x_1, \ldots, x_{\sqrt{n}}$ (respectively) if and only if the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ all belong to U^c *for every color* $c \in [\chi]$. Thus, if the prover indeed provides vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ that are constructed as defined above, we are done.

It remains to verify that the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ are obtained from $x_1, \ldots, x_{\sqrt{n}}$ as expected. To this end, observe that the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ can be obtained from $x_1, \ldots, x_{\sqrt{n}}$ using multiplication. More specifically, for every $c \in [\chi]$ and $j \in [\sqrt{n}]$, we define a vector z_j^c by setting $\left(z_j^c\right)_i \stackrel{\mathrm{def}}{=} 1$ if S_j contains a constraint of color c that touches the coordinate i, and $\left(x_j^c\right)_i \stackrel{\mathrm{def}}{=} 0$ otherwise. We then observe that $x_j^c = x_j \cdot z_j^c$ (where the multiplication is *coordinate-wise*).

Let (C_A, C_B, C_M) be the multiplication codes of Assumption 4.3.4. Now, the verifier checks the equality $x_j^c = x_j \cdot z_j^c$ by checking that the encodings $C_M(x_j^c)$ and $C_A(x_j) \cdot C_B(z_j^c)$ agree on a random coordinate. To this end, the verifier expects the prover to provide $C_A(x_j)$ and $C_M(x_j^c)$ (for every c and j), while $C_B(z_j^c)$ is computed by the verifier itself.

Finally, the verifier should verify that the purported vectors $C_A(x_j)$, $C_M(x_j^c)$ that are provided by the prover are indeed legal codewords of C_A and C_M, and this should be done for all j's simultaneously. However, this check can again be done as in the simple case of Section 4.5.3.1. This concludes our construction of the SLV.

An SLV that has row/column access. Recall that we want our SLV to have $O(1)$-row/column access with assignments sub-matrix. To this end, we modify the foregoing construction by a little.

Recall that for every j and $c \in [\chi]$, the verifier V checks that $x_j^c = x_j \cdot z_j^c$ by checking that $C_M(x_j^c)$ and $C_A(x_j) \cdot C_B(z_j^c)$ agree on a random coordinate. We now modify V such that it chooses the same random coordinate for all j and $c \in \chi$. It is not hard to see that this modification does not harm the soundness analysis. However, as we will now show, the modified SLV indeed has row/column access with assignments sub-matrix. We also mention that this modification is required in order to save in the randomness complexity of V.

In order to show that the modified SLV has row/column access with assignments sub-matrix, consider the matrix M whose rows are the following:

1. For each $j \in [\sqrt{n}]$, the matrix M contains the row that consists of x_j concatenated with $C_A(x_j)$.
2. For each $j \in [\sqrt{n}]$ and $c \in [\chi]$, the matrix M contains the row that consists of x_j^c concatenated with $C_M(x_j^c)$.
3. The matrix M contains all the additional rows that are needed for the invocations of the simple case of Section 4.5.3.1.

We proceed to show that the modified SLV has row/column access with assignments sub-matrix with respect to M. First, observe that the assignments $x_1, \ldots, x_{\sqrt{n}}$ indeed form a sub-matrix of M. Next, note that each invocation of the the simple case of Section 4.5.3.1 can be emulated by querying exactly one row or column of M. Finally, recall that except for the invocations of the simple case, the only check made by the SLV is the checks that the codewords $C_M(x_j^c)$ and $C_A(x_j) \cdot C_B(z_j^c)$ agree on a random coordinate, for all $j \in [\sqrt{n}]$ and $c \in [\chi]$. Now, observe that since the same random coordinate is used for all j and c, this check can be done by querying exactly one column of M (the column that corresponds to the chosen random coordinate). It follows that the SLV has $(\chi + 1)$-row/column access with assignments sub-matrix, as required.

Remark 4.5.8. We note that the foregoing description slightly oversimplifies things. In particular, the encoding $C_M(x_j^c)$ is not well-defined because the message length of C_M is larger than $\left|x_j^c\right|$. Thus, in the actual construction, the prover is not required to provide $C_M(x_j^c)$ but is rather required to provide $C_A(x_j)\cdot C_B(z_j^c)$. We then use the fact that x_j^c is a prefix of the message encoded by $C_A(x_j)\cdot C_B(z_j^c)$, due to the fact that C_A, C_B, and C_M are systematic.

4.5.3.3 The general case

We conclude our construction of SLVs by showing that the general case can be reduced to the case of 6-CCS (i.e., CCS with 6 colors). The basic idea of the reduction is that every constraint system can be embedded on any routing network, and in particular one may choose a routing network whose edges can be colored using few colors. It should be mentioned that while the idea of embedding a constraint system on a routing network has been used in several prior works (e.g. [BFLS91, PS94]), the use of this technique for reducing a general constraint system to a CCS is new.

In order to embed a system of linear constraints on a routing network, we add auxiliary variables to the system in a way that essentially does not change the solution space. This notion of adding auxiliary variables while "essentially" not changing the solution space is captured by the following notion of *extension.*

Definition 4.5.9. Let $W \subseteq \{0,1\}^m$ and let $l \in \mathbb{N}$. We say that a subspace $W' \subseteq \{0,1\}^{m+l}$ is an extension of W if it satisfies the following property: A vector $x \in \{0,1\}^m$ belongs to W if and only if there exists a vector $y \in \{0,1\}^l$ such that $x \circ y \in W'$.

In the foregoing Definition 4.5.9, the vector y represents the assignment to the auxiliary variables, and the fact that the solution space remains intact is captured by the fact that the m long prefix of each vector in W' is in W, and by the fact that every vector in W has a corresponding vector in W'. Our reduction of a general constraint system to a CCS can now be stated as follows.

Claim 4.5.10. *Let $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ be linear circuits of size n. Then, one can transform $\varphi_1, \ldots, \varphi_k$ in polynomial time to circuits $\varphi'_1, \ldots, \varphi'_k : \{0,1\}^{m'} \to \{0,1\}^{t'}$ that form a 6-CCS, such that for each $i \in [k]$ the subspace $W_{\varphi'_i}$ is an extension of W_{φ_i}. Furthermore, the circuits $\varphi'_1, \ldots, \varphi'_k$ are of size $\tilde{O}(n)$, and it holds that $m' = \tilde{O}(n)$, $t' = \tilde{O}(n)$.*

Proving Claim 4.5.10 essentially finishes our work, since it is not hard to use it to construct SLVs for arbitrary linear circuits given this claim and the construction of Section 4.5.3.2. Moreover, the resulting SLV will have 7-row/column access with assignment sub-matrix, as required. In the rest of this section we sketch the proof of Claim 4.5.10. Fix a collection of circuits $\varphi_1, \ldots, \varphi_k$, let $W_1, \ldots, W_k$ be the subspaces accepted by those circuits, and let $S_1, \ldots, S_k$ be sets of linear constraints corresponding to $\varphi_1, \ldots, \varphi_k$. For each W_i, we denote by $W'_i \stackrel{\text{def}}{=} W_{\varphi'_i}$ the extension of W_i that we seek to construct.

Warm-up. As a warm-up, consider the case in which every set S_i is a collection of disjoint equality constraints. That is, we consider the case in which every constraint in S_i touches exactly two coordinates, and every coordinate is touched by exactly one constraint. We show that that such sets S_i can be extended to a 4-CCS - note that we use two colors less than in Claim 4.5.10, a fact that will be used later.

We would like to construct for each W_i a subspace $W'_i \subseteq \{0,1\}^{m'}$ which is an extension of the subspace W_i, such that $W'_1, \ldots, W'_k$ form a 4-CCS. The idea that underlies the construction is the following. We identify every coordinate in $[m']$ with a vertex of a routing network G (see Section 4.2.3). We then embed each equality constraint of the set S_i on a path in G that connects the coordinates touched by the constraint. The 4-colorability of the corresponding constraints system follows from the fact that G is 4-edge colorable. Some details follow.

Let G be a routing network with $m' = m \cdot \operatorname{poly}\log m$ vertices. We now construct a set S'_i of constraints that describes the subspace $W'_i \subseteq \{0,1\}^{m'}$ as follows. We begin by identifying each coordinate in $[m']$ with a vertex of G, and in particular identify the coordinates in $[m]$ with vertices of G. Next, we find a collection $\mathcal{P}$ of vertex-disjoint paths on G, such that for each equality constraint $s \in S_i$, there is a path in $\mathcal{P}$ that connects the coordinates that s touches. Finding such paths is possible by the fact that G is a routing network. Finally, for each edge e of G, we put in S'_i an equality constraint between the endpoints of e if and only if e participates in one of the aforementioned vertex-disjoint paths.

It should be clear that W'_i is an extension of W_i. To see that $W'_1, \ldots, W'_k$ form a 4-CCS, let $\mathcal{S}' \stackrel{\text{def}}{=} S'_1 \cup \ldots \cup S'_k$ and observe that every constraint in $\mathcal{S}'$ is an equality constraint between the endpoints of some edge of G. Now, we can choose the network G to be 4-edge colorable, in which case the constraints in $\mathcal{S}'$ can be colored using 4 colors such that no two constraints of the same color touch the same coordinate. Thus, $W'_1, \ldots, W'_k$ and $\mathcal{S}'$ satisfy Definition 4.5.7 with $\chi = 4$. This concludes the construction.

Handling arbitrary linear circuits. It remains to show how to transform arbitrary linear circuits $\varphi_1, \ldots, \varphi_k$ to a 6-CCS as in Claim 4.5.10. Let $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ be linear circuits of size n, and assume for simplicity that every gate in $\varphi_1, \ldots, \varphi_k$ has exactly two incoming and two outgoing wires (it is not hard to remove this assumption). The proof strategy is to construct for each subspace W_i an extension $U_i \subseteq \{0,1\}^{2n}$ that is described by constraints of two types: The constraints of the first type form a 2-CCS. The constraints of the second type are equality constraints. We then complete the proof by handling the second type equality constraints in the same way as in the "warm-up" case of Section 4.5.3.3.

We turn to describe how to construct the subspace $U_i \subseteq \{0,1\}^{2n}$. For each wire v of φ_i, we associate v with two coordinates in $[2n]$, denoted v^{in} and v^{out}. The subspace U_i is described by two types of constraints:

1. **Computation constraints:** For each gate g in φ_i, whose incoming wires are v_1, v_2 and whose outgoing wires are v_3, v_4, we have two linear constraints of the form $v_3^{\text{out}} = v_1^{\text{in}} + v_2^{\text{in}}$ and $v_4^{\text{out}} = v_1^{\text{in}} + v_2^{\text{in}}$. In addition, for each output wire v of φ_i, we have a constraint of the form $v^{\text{out}} = 0$.
2. **Consistency constraints:** For every wire v in φ_i, we have the linear constraint $v^{\text{in}} = v^{\text{out}}$.

Observe that each U_i is indeed an extension of W_i.

Now, let us re-arrange the coordinates in $[2n]$ such that, if a gate g of φ_i has incoming wires v_1, v_2 and outgoing wires v_3, v_4, then the coordinates v_1^{in}, v_2^{in}, v_3^{out}, v_4^{out} are consecutive as numbers. The critical observation is that when using this ordering, the computation constraints are the same for every subspace U_i. In other words, the only difference between distinct subspaces U_i, U_j is the consistency constraints. In addition, observe that the computation constraints can be colored using two colors such that no two computation constraints of the same color touch the same coordinate. The latter two observations imply that the computation constraints already satisfy the requirements of a 2-CCS.

We have seen that the computation constraints already satisfy the requirements of a CCS, so it remains to handle the consistency constraints. This is done as in the "warm-up" case of Section 4.5.3.3, by embedding the consistency constraints on a routing network, and coloring them using 4 colors. Note that this embedding requires to construct for each U_i an extension $W'_i \subseteq \{0,1\}^{m'}$ for $m' = \tilde{O}(n)$. Finally, we take $W'_1, \ldots, W'_k$ to be the required 6-CCS. Note that each W'_i is indeed an extension of W_i, since W'_i is an extension of U_i which is in turn an extension of W_i.

4.6 Proof of the main theorem

In this section, we show how to prove our main theorem, restated below, by combining the tools that were developed in the previous sections.

Theorem (4.1.1, main theorem, restated). *For every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in \mathbf{NTIME}(t)$, there exists a PCP verifier for L with proof length $\ell(n) = t(n) \cdot (\log(t(n)))^{O(\log\log t(n))}$, query complexity $O(1)$, and rejection probability $\Omega(1)$.*

Fix a time-constructible function t. We construct PCPs with the required parameters for $\mathbf{NTIME}(t)$. Our starting point is the linear PCPP constructed in Corollary 4.5.5, which we denote here V_1. The verifier V_1 has proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\text{poly}\log n$. In addition, V_1 has $O(1)$-row/column access. We thus apply the robustization technique of Theorem 4.4.7 to V_1, resulting in a linear PCPP verifier V_2 that has proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and *robustness* $1/\text{poly}\log n$.

Our next step is to compose V_2 with itself for $\log\log n$ times, resulting in a linear PCPP verifier V_3 that has proof length $n \cdot (\log n)^{O(\log\log n)}$, query complexity $(\log n)^{O(\log\log n)}$, and rejection ratio $1/(\log n)^{O(\log\log n)}$. Then, we apply the transformation of linear PCPPs to (general) PCPs of Theorem 4.3.6, resulting in a PCP verifier V_4 for $\mathbf{NTIME}(t)$, which has proof length $t \cdot (\log t)^{O(\log\log t)}$, query complexity $(\log t)^{O(\log\log t)}$, and rejection probability $1/(\log t)^{O(\log\log t)}$.

We proceed by reducing the query complexity of V_4 to $O(1)$ by applying a standard query reduction technique that works roughly like the reduction of CircuitSat to 3Sat. This technique increases the proof length by a factor that is polynomial in the original query complexity, and decreases the rejection probability by a similar factor. We are left with a PCP verifier V_5 for $\mathbf{NTIME}(t)$, which has proof length $t \cdot (\log t)^{O(\log\log t)}$, constant query complexity, and rejection probability $1/(\log t)^{O(\log\log t)}$.

Finally, we apply the gap amplification technique of Dinur [Din07] to V_5. This technique can be applied to PCPs with constant query complexity, and increases the rejection probability of a PCP verifier to a constant, while increasing the proof length by a factor that is inversly polynomial in the original rejection probability, and maintaining the constant query complexity. We therefore obtain a PCP verifier V_6 for $\mathbf{NTIME}(t)$, which has proof length $t \cdot (\log t)^{O(\log \log t)}$, constant query complexity, and constant rejection probability. V_6 is the required PCP verifier.

Remark 4.6.1. The foregoing description oversimplifies things a little. In particular, a few of the steps taken above require bounds on the randomness complexity and the decision complexity of the verifiers. However, such bounds can be proved.

Chapter 5

Combinatorial PCPs with Low Soundness Error

5.1 Introduction

A PCP (Probabilistically Checkable Proof) is a proof system that allows checking the validity of a claim by reading only a constant number of bits of the proof. The PCP theorem asserts the existence of PCPs of polynomial length for any claim that can be stated as membership in an **NP** language. In this chapter, we consider the soundness error of PCPs, which is the probability that a false claim is accepted, and give a new construction of a PCP with sub-constant soundness error and two queries. This setting is particularly important for inapproximability, as will be discussed shortly below. Formally, we prove the following result.

Theorem 5.1.1 (Two-query PCP with small soundness). *There exists a constant $\kappa > 0$ such that for every function $\varepsilon : \mathbb{N} \to (0,1)$ satisfying $1/n^{\kappa} \leq \varepsilon(n) \leq 1/\mathrm{poly}\log n$ the following holds: Every language $L \in \mathbf{NP}$ has a two-query PCP system with perfect completeness, soundness error $1/\mathrm{poly}\log n$, alphabet size $2^{1/\mathrm{poly}(\varepsilon)}$, proof length $\mathrm{poly}\,(n)$, and randomness complexity $O(\log n)$. Furthermore, the verifier in this PCP system makes only 'projection' queries.*

This theorem matches the parameters of the folklore "manifold vs. point" construction which has been the only construction in the literature for this parameter range. The technical heart of that construction is a sub-constant error low degree test [RS97, AS03], see full details in [MR08].

Our proof of Theorem 5.1.1 is based on the elegant derandomized direct product test of [IKW09]. In a nutshell, our construction is based on applying this test to obtain a "derandomized parallel repetition theorem". While it is not clear how to do this for an arbitrary PCP, it turns out to be possible for PCPs with certain structure. We show how to convert any PCP to a PCP with the required structure, and then prove a "derandomized parallel repetition theorem" for such PCPs, thereby getting Theorem 5.1.1. The derandomized parallel repetition theorem relies on a reduction from the derandomized direct product test of [IKW09].

The Moshkovitz-Raz Construction. Recently, Moshkovitz and Raz [MR08] constructed even stronger PCPs. Specifically, they managed to remove the limitation $\varepsilon(n) \leq 1/\mathrm{poly}\log n$ from Theorem 5.1.1, thus allowing any function $\varepsilon(n) \geq 1/n^{\kappa}$. This allows constructing PCPs with sub-constant error and *any* alphabet size smaller than $2^{\mathrm{poly}\log n}$, at the expense of a suitable increase in the soundness error. Being able to reduce the alphabet size has strong consequences for inapproximability, see [MR08] for details. The technique of [MR08] (as explained in the later simplification of [DH09]) is essentially based on the composition of certain PCP constructions. In fact, their main building block is the "manifold vs. point" construction mentioned above.

Our construction can be extended to yield a so-called decodable PCP [DH09], which is an object slightly stronger than a PCP. This can be plugged into the scheme of [DH09] to give a nearly[1] combinatorial proof of the following result of [MR08]. Namely,

Theorem 5.1.2 ([MR08]). *There exists a constant $\kappa > 0$ such that for every function $\varepsilon(n) \geq 1/n^{\kappa}$ the following holds: Every language $L \in$ **NP** has a two-query PCP system with perfect completeness, soundness error ε, alphabet size at most $2^{1/\mathrm{poly}(\varepsilon)}$, proof length $\mathrm{poly}(n)$, and randomness complexity $O(\log n)$. Furthermore, the verifier in this PCP system makes only 'projection' queries.*

We note that the result of [MR08] is in fact even stronger than claimed above since their verifier has almost-linear proof length (specifically $n^{1+o(1)}$), and has randomness complexity of only $(1 + o(1)) \log n$ random bits, see also Remark 5.6.27.

Organization of the introduction. In the following four sections we outline the background and main ideas of this chapter. We start by describing the parallel repetition technique in general and its relation with direct product tests. We proceed to describe our technique of derandomized parallel repetition. We then describe our notion of "PCPs with linear structure", to which the derandomized parallel repetition is applied.

After the foregoing outline, we discuss relevant works and possible future directions, and describe the organization of this chapter.

Parallel repetition and Direct Products

A natural approach to reducing the soundness error of a PCP verifier is by running it several times independently, and accepting only if all runs accept. This is called *sequential repetition.* Obviously, if the verifier is invoked k times the soundness error drops exponentially in k. However, the total number of queries made into the proof grows k-fold, and in particular, it is greater than 2. Since our focus is on constructing PCPs that make only two queries, we can not afford sequential repetition.

In order to decrease the soundness error while maintaining the query complexity, one may use *parallel repetition.* For the rest of this discussion, we consider only PCPs that use only two queries. Let us briefly recall what parallel repetition means in this context. As in the case of sequential repetition, one starts out with a PCP with constant soundness error, and then amplifies the rejection probability by repetition of the verifier. However, in order to save on queries, the prover is expected to give the k-wise direct product encoding of the original proof. Formally, if $\pi : [n] \to \Sigma$ describes the original proof then its direct product encoding, denoted by $\pi^{\otimes k}$, is the function $\pi^{\otimes k} : [n]^k \to \Sigma^k$ defined by

$$\pi^{\otimes k}(x_1, \ldots, x_k) = (\pi(x_1), \ldots, \pi(x_k)).$$

The new verifier will simulate the original verifier on k independent runs, but will read only *two* symbols from the new proof, which together contain answers to k independent runs of the original verifier.

Of course, there is no a priori guarantee that the given proof is a direct product encoding $\pi^{\otimes k}$ of any underlying proof π, as intended in the construction. This is the main difficulty in proving the celebrated parallel repetition due to Raz [Raz98] that shows that the the soundness error does go down exponentially with k.

One may try to circumvent the difficulty in analyzing the parallel repetition theorem by augmenting it with a direct product test. That is, making the verifier *test* that the given proof Π is a direct product

[1]It is debatable whether our use of "linear structure" disqualifies the result from being considered purely combinatorial.

encoding of some string π, and only then running the original parallel repetition verifier. This can sometimes be done without even incurring extra queries. Motivated by this approach Goldreich and Safra [GS00] suggested and studied the following question:

DP testing: Given a function $F : [n]^k \to \Sigma^k$ test that it is close to $f^{\otimes k}$ for some $f : [n] \to \Sigma$.

Let us now describe a two query direct product test. From now on let us make the simplifying assumption that the function $F : [n]^k \to \Sigma^k$ to be tested is given as a function of k-sized subsets rather than tuples, meaning that $F(x_1, \dots, x_k)$ is the same for any permutation of $x_1, \dots, x_k$. The test chooses two random k-subsets $B_1, B_2 \in \binom{[n]}{k}$ that intersect on a subset $A = B_1 \cap B_2$ of a certain prescribed size and accept if and only if $F(B_1)_{|A} = F(B_2)_{|A}$. This test was analyzed further in several works, see [GS00, DR06, DG08, IKW09].

Remark 5.1.3. An expert reader may note that the above direct product test is not a projection test, while we need a projection test for Theorem 5.1.1. Indeed, in our actual proof we use a variant of the above direct product test which is a projection test (see Section 5.2.1 for details).

Derandomized Direct Product Testing

Recall that our goal is to construct PCPs with sub-constant soundness error. Note, however, that since the parallel repetition increases the proof length exponentially in k (and the randomness of the verifier grows k-fold), one can only afford to make a constant number of repetitions if one wishes to maintain polynomial proof length and logarithmic randomness complexity. On the other hand, obtaining sub-constant soundness error requires a super-constant number of repetitions.

This leads to the derandomization question, addressed already 15 years ago [FK95]. Can one recycle randomness of the verifier in the parallel repetition scheme without losing too much in soundness error?

Motivated by this question, Impagliazzo, Kabanets, and Wigderson [IKW09] introduced a method for analyzing the direct product test which allowed them to derandomize it. Namely, they exhibited a relatively small collection of subsets $\mathcal{K} \subset \binom{[n]}{k}$, and considered the restriction of the direct product encoding $f^{\otimes k}$ to this collection. They then showed that this form of derandomized direct product can be tested using the above test. The collection $\mathcal{K}$ is as follows: identify $[n]$ with a vector space $\mathbb{F}^m$, let $k = |\mathbb{F}|^d$ for constant d, and let $\mathcal{K}$ be the set of all d-dimensional linear subspaces.

A natural next step is to use the derandomized direct product of [IKW09] to obtain a derandomized parallel repetition theorem. Recall that the parallel repetition verifier works by simulating k independent invocations of the original verifier on π, and querying the (supposed) direct product Π on the resulting k-tuples of queries. However, in the derandomized setting, the k-tuples of queries generated by the verifier may fall outside $\mathcal{K}$. This is the main difficulty that we address in this chapter.

This is where the structure of the PCP comes to our aid. We show that for PCPs with a certain linear structure, the k-tuples of queries can be made in a way that is compatible with the derandomized direct product test of [IKW09]. More specifically, the k-tuples of queries always belong to the collection $\mathcal{K}$, and are distributed like queries of the derandomized direct product test. This allows us to prove a derandomized parallel repetition theorem for the particular case of PCPs with linear structure. Our main theorem is proved by constructing PCPs with linear structure (discussed next), and applying the derandomized parallel repetition theorem.

PCPs with Linear Structure

We turn to discuss PCPs with linear structure. The *underlying graph structure* of a two-query PCP is a graph defined as follows. The vertices are the proof coordinates, and the edges correspond to all possible query pairs of the verifier. (See also Section 5.2.3). We say that a graph has *linear structure* if the vertices can be identified with a vector space $\mathbb{F}^m$ and the edges, which clearly can be viewed as a subset of $\mathbb{F}^{2m}$, form a linear subspace of $\mathbb{F}^{2m}$ (see also Definition 5.3.1). A two-query PCP has linear structure if its underlying graph has linear structure.

As mentioned above, an additional contribution of this chapter is the construction of PCPs with linear structure. That is, we prove the following result.

Theorem 5.1.4 (PCPs with linear structure). *Every language $L \in$* **NP** *has a two-query PCP system with a linear structure which has perfect completeness, soundness error* $1-1/\text{poly}\log n$*, constant alphabet size, proof length* poly (n)*, and randomness complexity* $O(\log n)$*.*

We believe that Theorem 5.1.4 is interesting in its own right: For known PCPs, the underlying graph structure is quite difficult to describe, mostly due to the fact that PCP constructions are invariably based on composition. In principle, however, the fact that a PCP is a "complex" object need not prevent the underlying graph from being simple. In analogy, certain Ramanujan expanders [LPS88] are Cayley graphs that are very easy to describe, even if the proof of their expansion is not quite so easy. It is therefore interesting to study whether there exist PCPs with simple underlying graphs.

Philosophically, the more structured the PCP, the stronger is the implied statement about the class NP, and the easier it is to exploit for applications. Indeed, the structure of a PCP system has been used in several previous works. For example, Khot constructs [Kho06] a PCP with quasi-random structure in order to establish the hardness of minimum bisection. Dinur [Din07] imposes an expansion structure on a PCP to obtain amplification.

We prove Theorem 5.1.4 by embedding a given PCP into the de Bruijn graph and relying on the algebraic structure of this graph. We remark that the de Bruijn graph has been used in constructions of PCPs before, e.g. [PS94, BFLS91], in similar contexts. We believe that structured PCPs are an object worthy of further study. One may view their applicability towards proving Theorem 5.1.1 as supporting evidence. An interesting question which we leave open is whether Theorem 5.1.4 can be strengthened so as to get *constant* soundness error. By simply plugging such a PCP into our derandomized parallel repetition theorem one would get a direct proof of the aforementioned result of [MR08], *without* using two-query composition.

Remark 5.1.5. Our notion of PCPs with linear structure should not be confused with the notion of "linear PCPPs" that appeared in the literature before (see [BSHLM09], and the related "linear inner verifier" of [GS00]). A linear PCPP is, roughly, a PCP system for checking the membership of a vector in a given linear subspace, in which the proof is required to be a linear function of the aforementioned vector. This requirement is unrelated to our definition, which does not restrict the claim to be verified or the proof, and on the other hand restricts the query structure of the PCP verifier.

Decodable PCPs

We extend our results to also yield a new construction of *decodable PCPs* (dPCPs). A dPCP gives a way to encode NP witnesses so that a verifier (called a decoder in this context) is able to both locally test their validity as well as to locally decode bits from the encoded NP witness. Decodable PCPs[2] were introduced in [DH09] towards simplifying and modularizing the work of [MR08] on two-query PCPs with small soundness. In [DH09] the result of [MR08] was reproved assuming the existence of two building blocks, a PCP and a dPCP, which were used as a black box. Until this work there has been only one known construction of a dPCP, based on the manifold vs. point construction. In this chapter we give

[2]Decodable PCPs generalize the notion of "locally decode/reject codes" of [MR08] and the even earlier notion of "LDF readers" of [DFK$^+$99].

a new construction of a dPCP which is obtained by applying derandomized parallel repetition in an analogous way to Theorem 5.1.1. We prove

Theorem 5.1.6 (dPCP, informal version)**.** *There exists a two-query PCP decoder with perfect completeness, soundness error* $1/\text{poly}\log n$*, list size* $\text{poly}\log n$*, proof alphabet* $2^{\text{poly}\log n}$*, proof length* $\text{poly}\,(n)$*, and randomness complexity* $O(\log n)$*.*

The notion of dPCPs is described in detail in Section 5.6, and in particular in Section 5.6.2. Theorem 5.1.6 is stated and proved in Section 5.6.4 based on two main lemmas, which are proved in Sections 5.7 and 5.8.

In order to prove this theorem we generalize each of the steps of the proof of Theorem 5.1.1. First, we construct a dPCP with linear structure but with relatively high soundness error in an analogous way to our proof of Theorem 5.1.4 (PCPs with linear structure). Next, we apply derandomized parallel repetition to get the desired dPCP. The two steps are described in Sections 5.7 and 5.8 respectively.

An additional contribution of this chapter is an extension of the definitions of [DH09], of dPCPs that work with low soundness error, to one that works with high soundness error. This is necessary because plugging in a higher value for the soundness error parameter into the existing definition of [DH09] turns out to be useless. Instead, we give a variant which we call uniquely decodable PCPs (udPCPs). We show that udPCPs are in fact equivalent to PCPs of Proximity (PCPPs). This allows us to rely on known constructions of PCPPs [BSGH$^+$06, DR06] as our starting point. For more details see Section 5.6.2.

Together, Theorem 5.1.1 and Theorem 5.1.6 imply Theorem 5.1.2 (the [MR08] result). This is sketched in Section 5.6.5.

Remark 5.1.7. In fact, Theorem 5.1.6 can be proved for any soundness error $\varepsilon(n)$ satisfying $1/n^\kappa \leq \varepsilon(n) \leq 1/\text{poly}\log n$ (for some constant $\kappa > 0$. As in Theorem 5.1.1, the alphabet size in such case is $2^{1/\text{poly}(\varepsilon)}$, and furthermore the list size becomes $1/\text{poly}\,(\varepsilon)$. However, in this chapter we only prove Theorem 5.1.6 for $\varepsilon(n) = 1/\text{poly}\log n$, since this is all we need to in order to prove Theorem 5.1.2 (the [MR08] result).

Related Work and Future directions

Our final construction of a two-query PCP has exponential relation between the alphabet size and the error probability (that is, $|\Sigma| = 2^{1/\text{poly}(\varepsilon)}$). In general, one can hope for a polynomial relation, and this is the so-called "sliding scale" conjecture of [BGLR93]. Our approach is inherently limited to an exponential relation both because of a lower bound on direct product testing from [DG08], and, more generally, because of the following lower bound of Feige and Kilian [FK95] on parallel repetition of games. Feige and Kilian prove that for every PCP system and $k = O(\log n)$ invocations of the original verifier, if one insists on the parallel repetition using only $O(\log n)$ random bits, then the soundness error must be at least $1/\text{poly}\log n$ (and not $1/\text{poly}(n)$ as one might hope). For the choice of $k = O(\log n)$, the results of this chapter match the [FK95] lower bound by exhibiting a derandomized parallel repetition theorem, albeit only for PCPs with linear structure, that achieves a matching upper bound of $1/\text{poly}\log n$ on the soundness error.

Nevertheless, for three queries we are in a completely different ball-game, and no lower bound is known. It would be interesting to find a derandomized direct product test with three queries with lower soundness error, and to try and adapt it to a PCP. We note that there are "algebraic" constructions [RS97, DFK$^+$99] that make only three queries and have much better relationship between the error and the alphabet size.

It has already been mentioned that while our result matches the soundness error and alphabet size of the [MR08] result, it does not attain nearly linear proof length. Improving our result in this respect is another interesting direction.

Structure of the chapter

The chapter has two main parts, the first part is concerned with proving the main result for PCPs, and the second part generalizes this result to dPCPs.

- **Part 1.** The structure of the proof is "top to bottom". Our main theorem for PCPs is based on two main steps: (i) embedding a PCP into a PCP with linear structure, and (ii) a derandomized parallel repetition theorem for such PCPs. We begin, in Section 5.3, by stating the two main lemmas corresponding to the two steps above, and then proving the main theorem, assuming correctness of the lemmas. We then proceed to prove each main lemma. In Section 5.4 we show how to embed a PCP into one with linear structure (by routing it on a de Bruijn like graph). In Section 5.5 we prove the "derandomized parallel repetition" theorem for PCPs with linear structure. This is done by reduction to the derandomized direct product test of [IKW09]. More accurately, our analysis relies on a specialized variant of this test which we call an S-test, which is analyzed in Section 5.9.
- **Part 2.** The second part of the chapter adapts our PCP construction to a dPCP. In Section 5.6 we discuss and define dPCPs, and prove Theorem 5.1.6. We also show how to use this theorem to derive the [MR08] result (Theorem 5.1.2) as a corollary. The two main steps in the proof of Theorem 5.1.6 are described in Sections 5.7 and 5.8 and are analogous to the two main steps of proving Theorem 5.1.1.
- Finally, we analyze the specialized direct product test (called the S-test) in Section 5.9, based on the work of [IKW09].

5.2 Preliminaries

Let $g : U \to \Sigma$ be an arbitrary function, and let $A \subset U$ be a subset. We denote by $g_{|A}$ the restriction of g (as a function) to A. We also use the following convention.

Notation 5.2.1. Given two functions $f, g : U \to \Sigma$, we denote $f \overset{\alpha}{\approx} g$ ($f \overset{\alpha}{\not\approx} g$) to mean that they differ on at most (more than) α fraction of the elements of U.

We refer to a d-dimensional linear subspace of an underlying vector space simply as a *d-subspace*. For two linear subspaces A_1 and A_2, the standard notation $A_1 + A_2$ denotes the smallest linear subspace containing both of them. We say that A_1, A_2 are *independent* if and only if $A_1 \cap A_2 = \{0\}$. If A_1 and A_2 are disjoint, the standard notation $A_1 \oplus A_2$ is used to denotes $A_1 + A_2$.

Let $G = (V, E)$ be a directed graph. For each edge $e \in E$ we denote by left (e) and right (e) the left and right endpoints of e respectively. That is, if we view the edge $e \in E$ as a pair in $V \times V$, then left (e) and right (e) are the first and second elements of the pair e respectively. Given a set of edges $E_0 \subseteq E$, we denote by left (E_0) and right(E_0) the set of left endpoints and right endpoints of the edges in E_0 respectively.

5.2.1 Direct product testing [IKW09]

Let us briefly describe the setting in which we use the derandomized direct product test of [IKW09]. In [IKW09] the main derandomized direct product test is a so-called "V-test". We consider a variation of this test that appears in [IKW09, Section 6.3] to which we refer as the "P-test" (P for projection).

Given a string $\pi \in \Sigma^{\ell}$, we define its (derandomized) P-direct product Π as follows: We identify $[\ell]$ with $\mathbb{F}^m$, where $\mathbb{F}$ is a finite field and $m \in \mathbb{N}$, and think of π as an assignment that maps the points in

1. Choose a uniformly distributed d_1-subspace $B \subseteq \mathbb{F}^m$.
2. Choose a uniformly distributed d_0-subspace $A \subseteq B$.
3. Accept if and only if $\Pi(B)_{|A} = \Pi(A)$.

Figure 5.1: The P-test

$\mathbb{F}^m$ to Σ. We also fix $d_0 < d_1 \in \mathbb{N}$. Now, we define Π to be the assignment that assigns each d_0- and d_1-subspace W of $\mathbb{F}^m$ to the function $\pi_{|W} : W \to \Sigma$ (recall that $\pi_{|W}$ is the restriction of π to W).

We now consider the task of testing whether a given assignment Π is the P-direct product of some string $\pi : \mathbb{F}^m \to \Sigma$. In those settings, we are given an assignment to subspaces, i.e. a function Π that on input a d_0-subspace $A \subset \mathbb{F}^m$ (respectively d_1-subspace $B \subset \mathbb{F}^m$), answers with a function $a : A \to \Sigma$ (respectively, $b : \mathbb{F}^m \to \Sigma$). We wish to test whether Π is a P-direct product of some $\pi : \mathbb{F}^m \to \Sigma$, and to this end we invoke the P-test, described in Figure 5.1.

It is easy to see that if Π is a P-direct product then the P-test always accepts. Furthermore, it can be shown that if Π is "far" from being a P-direct product, then the P-test rejects with high probability. Formally, we have the following result.

Theorem 5.2.2 (Soundness of the P-test[IKW09])**.** *There exists a universal constant $h \in \mathbb{N}$ such that the following holds: Let $\varepsilon \geq h \cdot d_0 \cdot |\mathbb{F}|^{-d_0/h}$, $\alpha \overset{\text{def}}{=} h \cdot d_0 \cdot |\mathbb{F}|^{-d_0/h}$. Assume that $d_1 \geq h \cdot d_0$, $m \geq h \cdot d_1$. Suppose that an assignment Π passes the P-test with probability at least ε. Then, there exists an assignment π such that*

$$\Pr\left[\Pi(B)_{|A} = \Pi(A) \quad \text{and } \Pi(B) \overset{\alpha}{\approx} \pi_{|B} \quad \text{and } \Pi(A) \overset{\alpha}{\approx} \pi_{|A}\right] = \Omega(\varepsilon^4), \tag{5.1}$$

where the probability is over A, B chosen as in the P-test.

Theorem 5.2.2 can be proved by adapting the analysis of [IKW09] (in particular, Sections 3.4 and 4) to the setting of the P-test, while relying on a lemma of [IKW09]. The proof can be found in [DM10, App. A].

Working with randomized assignments. As observed by [IKW09], Theorem 5.2.2 works in even stronger settings. Suppose that Π is a randomized function, i.e., a function of both its input and some additional randomness. Then, Theorem 5.2.2 still holds for Π, where the probability in (5.1) is over both the choice of A and B, *and over the internal randomness of* Π. We will rely on this fact in a crucial way in this chapter.

5.2.2 Sampling tools

The following is the standard definition of a sampler, stated in the terminology of graphs, see e.g. [IJKW08].

Definition 5.2.3 (Sampler Graph)**.** A bipartite graph $G = (L, R, E)$ is said to be an (ε, δ)-sampler if, for every function $f : L \to [0, 1]$, there are at most $\delta |R|$ vertices $u \in R$ for which

$$\left|\mathbb{E}_{v \in N(u)}[f(v)] - \mathbb{E}_{v \in L}[f(v)]\right| > \varepsilon.$$

Observe that if G is an (ε, δ)-sampler, and if $F \subset L$, then by considering the function $f \equiv 1_F$ we get that there are at most $\delta |R|$ vertices $u \in R$ for which

$$\left|\Pr_{v \in N(u)}[v \in F] - \Pr_{v \in L}[v \in F]\right| > \varepsilon.$$

The following lemma is stated in [IKW09, Lemma 2.2] and is proved implicitly in [IJKW08, Lemma 2.9]. For completeness, we include its proof.

Lemma 5.2.4 (Subspace-point sampler [IJKW08])**.** *Let $d' < d$ be natural numbers, let V be a linear space over a finite field $\mathbb{F}$, and let W be a fixed d'-subspace of V. Let G be the bipartite graph whose left vertices are all points of V and whose right vertices are all d-subspaces of V that contain W. We place an edge between a d-subspace X and $x \in V$ if and only if $x \in X$. Then G is an $(\tau + \frac{1}{|\mathbb{F}|^{d-d'}}, \frac{1}{|\mathbb{F}|^{d-d'-2} \cdot \tau^2})$-sampler for every $\tau > 0$.*

Proof. Fix a function $f : V \to [0,1]$. We show that for a uniformly distributed d-subspace $X \subseteq V$ that contains W it holds with probability at least $1 - \frac{1}{|\mathbb{F}|^{d-d'-2} \cdot \tau^2}$ that

$$\left|\mathbb{E}_{x\in X}\left[f(x)\right] - \mathbb{E}_{v \in V}\left[f(v)\right]\right| \le \tau + \frac{1}{|\mathbb{F}|^{d-d'}}.$$

Let $\overline{W}$ be a fixed subspace of V for which $V = W \oplus \overline{W}$. Let $f_W : \overline{W} \to [0,1]$ be the function that maps each vector $\overline{w}$ of $\overline{W}$ to $\mathbb{E}_{v \in \overline{w}+W}\left[f(v)\right]$, and observe that $\mathbb{E}_{v\in V}\left[f(v)\right] = \mathbb{E}_{\overline{w}\in\overline{W}}\left[f_W(\overline{w})\right]$. Furthermore, observe that every d-subspace X that contains W can be written as $X = W \oplus U$ where U is a $(d-d')$-subspace of $\overline{W}$, and moreover that $\mathbb{E}_{x\in X}\left[f(x)\right] = \mathbb{E}_{u\in U}\left[f_W(u)\right]$. Thus, it suffices to prove that for a uniformly distributed $(d-d')$-subspace U of $\overline{W}$ it holds with probability at least $1 - \frac{1}{|\mathbb{F}|^{d-d'-2} \cdot \tau^2}$ that

$$\left|\mathbb{E}_{u\in U}\left[f_W(u)\right] - \mathbb{E}_{\overline{w}\in\overline{W}}\left[f_W(\overline{w})\right]\right| \le \tau + \frac{1}{|\mathbb{F}|^{d-d'}}. \tag{5.2}$$

To that end, let U be a uniformly distributed $(d-d')$-subspace of $\overline{W}$. Let S_1 be a uniformly distributed set of $Q \overset{\text{def}}{=} \frac{|\mathbb{F}|^{d-d'}-1}{|\mathbb{F}|-1}$ vectors of U such that every two vectors in S_1 are linearly independent[3]. For every $\alpha \in \mathbb{F}^*$ let S_α be the set obtained by multiplying every vector in S_1 by α. Observe that all the sets S_α have the property that every two vectors in S_α are linearly independent, and that the sets S_α form a partition of $U \setminus \{0\}$. We will show that for every $\alpha \in \mathbb{F}^*$ it holds with probability at least $1 - \frac{1}{|\mathbb{F}|^{d-d'-1} \cdot \tau^2}$ that

$$\left|\mathbb{E}_{u\in S_\alpha}\left[f_W(u)\right] - \mathbb{E}_{\overline{w}\in\overline{W}}\left[f_W(\overline{w})\right]\right| \le \tau,$$

and the required result will follow by taking the union bound over all $\alpha \in \mathbb{F}^*$, and by noting that the vector 0 contributes at most $\frac{1}{|\mathbb{F}|^{d-d'}}$ to the difference in Inequality 5.2.

Fix $\alpha \in \mathbb{F}^*$, and let $s_1, \ldots, s_Q$ be the vectors in S_α. It is a known fact that $s_1, \ldots, s_Q$ are pair-wise independent and uniformly distributed vectors of $\overline{W}$ (over the random choice of U). This implies that $f_W(s_1), \ldots, f_W(s_Q)$ are pair-wise independent random variables with expectation $\mathbb{E}_{\overline{w}\in\overline{W}}\left[f_W(\overline{w})\right]$, and therefore by the Chebyshev inequality it follows that

$$\Pr\left[\left|\frac{1}{Q}\sum_{i=1}^{Q} f_W(s_i) - \mathbb{E}_{\overline{w}\in\overline{W}}\left[f_W(\overline{w})\right]\right| > \tau\right] \le \frac{1}{Q\cdot\tau^2} \le \frac{1}{|\mathbb{F}|^{d-d'-1}\cdot\tau^2},$$

as required. ∎

[3]Such a set can be sampled, for example, by iteratively choosing a uniformly distributed vector of U that is linearly independent from each of the previously chosen vectors individually. It is not hard to see that such a process will halt after choosing $Q \overset{\text{def}}{=} \frac{|\mathbb{F}|^{d-d'}-1}{|\mathbb{F}|-1}$ vectors.

5.2.3 Constraint graphs and PCPs

As discussed in the introduction, the focus of this chapter is on claims that can be verified by reading a small number of symbols of the proof. A PCP system for a language L is an oracle machine M, called a verifier, that has oracle access to a proof π over an alphabet Σ. The verifier M reads the input x, tosses r coins, makes at most q "oracle" queries into π, and then accepts or rejects. If x is in the language then it is required that M accepts with probability 1 for some π, and otherwise it is required that M accepts with probability at most ε for every π. More formally:

Definition 5.2.5. Let $r, q : \mathbb{N} \to \mathbb{N}$, and let Σ be a function that maps the natural numbers to finite alphabets. A $(r,q)_\Sigma$*-PCP verifier* M is a probabilistic polynomial time oracle machine that when given input $x \in \{0,1\}^*$, tosses at most $r(|x|)$ coins, makes at most $q(|x|)$ *non-adaptive* queries to an oracle that is a string over $\Sigma(|x|)$, and outputs either "accept" or "reject". We refer to r, q, and Σ as the *randomness complexity*, *query complexity*, and *proof alphabet* of the verifier respectively.

Remark 5.2.6. Note that for an $(r,q)_\Sigma$-PCP verifier M and an input x, we can assume without loss of generality that the oracle is a string of length at most $2^{r(|x|)} \cdot q(|x|)$, since this is the maximal number of different queries that M can make. Hence, it is unnecessary to keep track of the proof length of the verifier.

Definition 5.2.7. Let r, q and Σ be as in Definition 5.2.5, let $L \subseteq \{0,1\}^*$ and let $\varepsilon : \mathbb{N} \to [0,1)$. We say that $L \in \mathbf{PCP}_{\varepsilon,\Sigma}[r,q]$ if there exists an $(r,q)_\Sigma$-PCP verifier M that satisfies the following requirements:

- **Completeness:** For every $x \in L$, there exists $\pi \in \Sigma(|x|)^*$ such that $\Pr[M^\pi(x) \text{ accepts}] = 1$.
- **Soundness:** For every $x \notin L$ and for every $\pi \in \Sigma(|x|)^*$ it holds that $\Pr[M^\pi(x) \text{ accepts}] \le \varepsilon(|x|)$.

One possible formulation of the PCP theorem is as follows.

Theorem 5.2.8 (PCP Theorem [AS98, ALM+98])**.** *There exist universal constant $\varepsilon \in (0,1)$ and a finite alphabet Σ such that* $\mathbf{NP} \subseteq \mathbf{PCP}_{\varepsilon,\Sigma}[O(\log n), 2]$.

PCPs that have query complexity 2 correspond to graphs in a natural way: Consider the action of an $(r,2)_\Sigma$-verifier M on some fixed string x, and let $r \overset{\text{def}}{=} r(|x|), \Sigma \overset{\text{def}}{=} \Sigma(|x|)$. The verifier M is given access to some proof string π of length ℓ, and may make 2^r possible tests on this string, where each such test consists of making two queries to π and deciding according to the answers. We now view the action of M as a graph in the following way. We consider the graph G whose vertices are the coordinates in $[\ell]$, and that has an edge for each possible test of the verifier M. The endpoints of an edge e of G are the coordinates that are queried by M in the test that corresponds to e. We also associate an edge e with a constraint $c_e \in \Sigma \times \Sigma$, which contains all the pairs of answers that make M accept when performing the test that corresponds to e. We think of π as an assignment that assigns the vertices of G values in Σ, and say that π *satisfies* an edge (u,v) if $(\pi(u), \pi(v)) \in c_{(u,v)}$. If $x \in L$, then it is required that there exists some assignment π that satisfies all the edges of G, and otherwise it is required that every assignment satisfies at most ε fraction of the edges. This correspondence is called the FGLSS correspondence [FGL+96]. We turn to state it formally:

Definition 5.2.9 (Constraint graph)**.** A *(directed) constraint graph* is a directed graph $G = (V,E)$ together with an alphabet Σ, and, for each edge $(u,v) \in E$, a binary constraint $c_{u,v} \subseteq \Sigma \times \Sigma$. The *size* of G is the number of edges of G. The graph is said to have *projection constraints* if it is bipartite with all the edges directed from the left to the right, and every constraint $c_{u,v}$ has an associated function

$f_{u,v} : \Sigma \to \Sigma$ such that $c_{u,v}$ is satisfied by (a, b) if and only if $f_{u,v}(a) = b$.
Given an assignment $\pi : V \to \Sigma$, we define

$$\mathrm{SAT}(G, \pi) = \Pr_{(u,v)\in E}[(\pi(u), \pi(v)) \in c_{u,v}] \quad \text{and} \quad \mathrm{SAT}(G) = \max_{\pi}(\mathrm{SAT}(G, \pi)).$$

We also denote $\mathrm{UNSAT}(G, \pi) = 1 - \mathrm{SAT}(G, \pi)$ and similarly $\mathrm{UNSAT}(G) = 1 - \mathrm{SAT}(G)$.

Remark 5.2.10. Note that Definition 5.2.9 uses *directed graphs*, while the common definition of constraint graphs refers to undirected graphs.

Remark 5.2.11. Note that if the graph G has projection constraints, then this is simply a label cover instance with projection constraints [AL96].

Proposition 5.2.12 (FGLSS correspondence [FGL+96])**.** *The following two statements are equivalent:*

- $L \in \mathbf{PCP}_{\varepsilon,\Sigma}[r, 2]$.
- *There exists a polynomial-time algorithm that transforms strings $x \in \{0,1\}^*$ to constraint graphs G_x of size $2^{r(|x|)}$ with alphabet $\Sigma(|x|)$ such that: (1) if $x \in L$ then $\mathrm{SAT}(G_x) = 1$, and (2) if $x \notin L$ then $\mathrm{SAT}(G_x) \leq \varepsilon$.*

Given a PCP system for L, we refer to the corresponding family of graphs $\{G_x\}$ where x ranges over all possible instances as its underlying graph family*. If the graphs $\{G_x\}$ have projection constraints then we say that the PCP system has the* projection property*.*

Using the [FGL+96] correspondence, we can rephrase the PCP theorem in the terminology of constraint graphs:

Theorem 5.2.13 (PCP Theorem for constraint graphs)**.** *There exist universal constant $\varepsilon \in (0, 1)$ and a finite alphabet Σ such that for every language $L \in \mathbf{NP}$ the following holds: There exists a polynomial time reduction that on input $x \in \{0,1\}^*$, outputs a constraint graph G_x such that if $x \in L$ then $\mathrm{SAT}(G_x) = 1$ and otherwise $\mathrm{SAT}(G_x) \leq \varepsilon$.*

Remark 5.2.14. The connection between PCPs and approximation problems (such as Proposition 5.2.12) was discovered by [FGL+96]. However, the precise correspondence between PCPs and constraint graphs that is given in Proposition 5.2.12 was only stated for the first time by [ALM+98]. Still, in the rest of this chapter we refer to Proposition 5.2.12 as the [FGL+96] correspondence.

Remark 5.2.15. Note the tight relationship between the randomness complexity of the PCP and the size of the corresponding constraint graphs. In particular, observe that PCP verifiers with randomness complexity $O(\log n)$ correspond to constraint graphs of polynomial size. This relationship is one of the main reasons for the study of the randomness complexity of PCP verifiers.

Moreover, recall that the work of [MR08] constructs PCPs that are very randomness efficient, i.e., have randomness complexity $(1 + o(1)) \log n$ (see also Remark 5.6.27). This randomness efficiency is translated into constraints graphs of almost-linear size, namely $n^{1+o(1)}$.

5.2.4 Basic facts about random subspaces

In this section we present two useful propositions about random subspaces. The following proposition says that a uniformly distributed subspace is independent from every fixed subspace with high probability.

Proposition 5.2.16. *Let $d, d' \in \mathbb{N}$ such that $d > 2d'$, and let V be a d-dimensional space. Let W_1 be a uniformly distributed d'-subspace of V, and let W_2 be a fixed d'-subspace of V. Then,*

$$\Pr[W_1 \cap W_2 = \{0\}] \geq 1 - 2 \cdot d' / |\mathbb{F}|^{d-2\cdot d'} .$$

Proof. Suppose that W_1 is chosen by choosing random basis vectors $v_1, \ldots, v_{d'}$ one after the other. It is easy to see that $W_1 \cap W_2 \neq \{0\}$ only if $v_i \in \text{span}\,(W_2 \cup \{v_1, \ldots, v_{i-1}\})$ for some $i \in [d']$. For each fixed i, the vector v_i is uniformly distributed in $V \backslash \text{span}\,\{v_1, \ldots, v_{i-1}\}$, and therefore the probability that $v_i \in \text{span}\,(W_2 \cup \{v_1, \ldots, v_{i-1}\})$ for a fixed i is at most

$$\begin{aligned} \frac{|\text{span}\,(W_2 \cup \{v_1, \ldots, v_{i-1}\})|}{|V \backslash \text{span}\,\{v_1, \ldots, v_{i-1}\}|} &= \frac{|\mathbb{F}|^{d'+i-1}}{|\mathbb{F}|^{d} - |\mathbb{F}|^{i-1}} \\ &\leq \frac{2 \cdot |\mathbb{F}|^{d'+i-1}}{|\mathbb{F}|^{d}} \qquad (5.3) \\ &\leq \frac{2 \cdot |\mathbb{F}|^{2\cdot d'-1}}{|\mathbb{F}|^{d}} \\ &\leq \frac{2}{|\mathbb{F}|^{d-2\cdot d'}}, \end{aligned}$$

where Inequality 5.3 can be observed by noting that $|\mathbb{F}|^{i-1} \leq |\mathbb{F}|^{d-1} \leq \frac{1}{2} \cdot |\mathbb{F}|^{d}$. By the union bound, the probability that this event occurs for some $i \in [d']$ is at most $\frac{2\cdot d'}{|\mathbb{F}|^{d-2\cdot d'}}$. It follows that the probability that $W_1 \cap W_2 \neq \{0\}$ is at most $\frac{2\cdot d'}{|\mathbb{F}|^{d-2\cdot d'}}$ as required. ■

The following proposition says that the span of d' uniformly distributed vectors is with high probability a uniformly distributed d'-subspace.

Proposition 5.2.17. *Let V be a d-dimensional space over a finite field $\mathbb{F}$, let $w_1, \ldots, w_{d'}$ be independent and uniformly distributed vectors of V, and let $W = \text{span}\{w_1, \ldots, w_{d'}\}$. Then, with probability at least $1 - d' / |\mathbb{F}|^{d-d'}$ it holds that $\dim W = d'$. Furthermore, conditioned on the latter event, W is a uniformly distributed d'-subspace of V.*

Proof. The fact that $\dim W = d'$ with probability at least $1 - d' / |\mathbb{F}|^{d-d'}$ can be proved in essentially the same way as Proposition 5.2.16. To see that conditioned on the latter event it holds that the subspace W is uniformly distributed, observe that since $w_1, \ldots, w_{d'}$ were originally chosen to be uniformly distributed, all the possible d'-sets of linearly independent vectors have the same probability to occur. ■

Finally, the following proposition shows the equivalence of two different ways of choosing subspaces $A_1, A_2 \subseteq B$ where A_1 and A_2 are independent.

Proposition. *Let V be a linear space over a finite field $\mathbb{F}$, and let $d_0, d_1 \in \mathbb{N}$ be such that $d_0 < d_1 < \dim V$. The following two distributions over d_0-subspaces A_1, A_2 and a d_1-subspace B are the same:*

1. *Choose B to be a uniformly distributed d_1-subspace of V, and then choose A_1 and A_2 to be two uniformly distributed and independent d_0-subspaces of B.*

2. *Choose A_1 and A_2 to be two uniformly distributed and independent d_0-subspaces of V, and then choose B to be a uniformly distributed d_1-subspace of V that contains A_1 and A_2.*

Proof. Observe that choosing A_1, A_2, B under the first distribution amounts to choosing d_1 uniformly distributed and linearly independent vectors in V (those vectors will serve as the basis of B), and then choosing two disjoint subsets of those vectors to serve as the basis of A_1 and as the basis of A_2. On the other hand, choosing A_1, A_2 and B under the second distribution amounts to choosing d_0 uniformly distributed and linearly independent vectors in V to serve as the basis of A_1, then choosing another d_0 uniformly distributed and linearly independent vectors in V to serve as the basis of A_2 while making sure that this basis is also linearly independent from the basis of A_1, and then completing the basis of A_1 and the basis of A_2 to a basis of B. It is easy to see that those two distributions over a set of d_1 vectors and its two disjoint subsets are identical. ■

5.2.5 Similarity of distributions

In this section we introduce a notion of "similarity of distributions", which we will use in the second part of the chapter. Let X_1 and X_2 be two random variables that take values from a set $\mathcal{X}$, and let $\gamma \in (0,1]$. We say that X_1 and X_2 are γ-similar if for every $x \in \mathcal{X}$ it holds that

$$\gamma \cdot \Pr[X_1 = x] \leq \Pr[X_2 = x] \leq \frac{1}{\gamma} \cdot \Pr[X_1 = x].$$

Note that if X_1 and X_2 are γ-similar then actually it holds for every $S \subseteq \mathcal{X}$ that

$$\gamma \cdot \Pr[X_1 \in S] \leq \Pr[X_2 \in S] \leq \frac{1}{\gamma} \cdot \Pr[X_1 \in S],$$

The following claim says roughly that if f is a randomized function, then the random variable $f(X_1)$ is γ-similar to $f(X_2)$.

Claim 5.2.18. *Let X_1 and X_2 be two random variables that take values from a set $\mathcal{X}$ that are γ-similar. Let Y_1 and Y_2 be two random variables that take values from a set $\mathcal{Y}$ such that for every $x \in \mathcal{X}$, $y \in \mathcal{Y}$ it holds that*

$$\Pr[Y_1 = y | X_1 = x] = \Pr[Y_2 = y | X_2 = x].$$

Then, the variables Y_1, Y_2 are γ-similar.

Proof. It holds that

$$\begin{aligned}
\Pr[Y_1 = y] &= \sum_{x \in \mathcal{X}} \Pr[Y_1 = y | X_1 = x] \cdot \Pr[X_1 = x] \\
&= \sum_{x \in \mathcal{X}} \Pr[Y_2 = y | X_2 = x] \cdot \Pr[X_1 = x] \\
&\geq \sum_{x \in \mathcal{X}} \Pr[Y_2 = y | X_2 = x] \cdot \gamma \cdot \Pr[X_2 = x] \\
&= \gamma \cdot \Pr[Y_2 = y].
\end{aligned}$$

Similarly it can be proved that $\Pr[Y_1 = y] \leq \frac{1}{\gamma} \cdot \Pr[Y_2 = y]$. ■

5.2.6 Expanders

Expanders are graphs with certain properties that make them extremely useful for many applications in theoretical computer science. Below we give a definition of expanders that suits our needs.

Definition 5.2.19. Let $G = (V, E)$ be a d-regular graph. Let $E\left(S, \overline{S}\right)$ be the set of edges from a subset $S \subseteq V$ to its complement. We say that G has edge expansion h if for every $S \subseteq V$ such that $|S| \leq |V|/2$ it holds that

$$\left|E(S, \overline{S})\right| \geq h \cdot d_0 \cdot |S| .$$

A useful fact is that there exist constant degree expanders over any number of vertices:

Fact 5.2.20. *There exist $d_0 \in \mathbb{N}$ and $h_0 > 0$ such that there exists a polynomial-time constructable family $\{G_n\}_{n \in \mathbb{N}}$ of d_0-regular graphs G_n on n vertices that have edge expansion h_0 (such graphs are called expanders).*

5.3 Main theorem

In this section we prove our main PCP theorem (Theorem 5.1.1), which asserts the existence of two-query PCPs with soundness error $\varepsilon(n)$ for any function $1/n^{\kappa} \leq \varepsilon(n) \leq 1/\text{poly}\log n$. To that end, we use the PCP theorem for graphs (Theorem 5.2.13) to reduce the problem of deciding membership of a string x in the language L to the problem of checking the satisfiability of a constraint graph with constant soundness error. We then show that every constraint graph can be transformed into one that has "linear structure", defined shortly below. This is done in Lemma 5.3.3, which directly proves Theorem 5.1.4 (the existence of PCPs with linear structure). Finally, in Lemma 5.3.4 we prove a derandomized parallel repetition theorem for constraint graphs with linear structure. Theorem 5.1.1 follows by combining the two lemmas. We begin by defining the notion of a graph with linear structure.

Definition 5.3.1 (Linear Structure)**.** We say that a directed graph G has a *linear structure if it satisfies the following conditions:*

1. *The vertices of G can be identified with the linear space $\mathbb{F}^m$, where $\mathbb{F}$ is a finite field and $m \in \mathbb{N}$.*
2. *We identify the set of pairs of vertices $(\mathbb{F}^m)^2$ with the linear space $\mathbb{F}^{2m}$. Using this identification, the edges E of G are required to form a linear subspace of $\mathbb{F}^{2m}$.*
3. *We require that* $\text{left}(E) = \text{right}(E) = \mathbb{F}^m$. *In other words, this means that every vertex of G is both the left endpoint of some edge and the right point of some edge.*

Remark 5.3.2. We mention that although it is not required by Definition 5.3.1, a graph with linear structure must be regular, i.e., all the vertices in the graph have the same in-degree and out-degree. This is a straightforward corollary of Items 2 and 3 of the definition.

The following lemmas are proved in Sections 5.4 and 5.5 respectively.

Lemma 5.3.3 (Linear Structure Embedding)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***
  - *A constraint graph G of size n over alphabet Σ.*
  - *A finite field $\mathbb{F}$ of size q.*
- ***Output:*** *A constraint graph $G' = (\mathbb{F}^m, E')$ such that the following holds:*
  - *G' has a linear structure.*
  - *The size of G' is at most $O(q^2 \cdot n)$.*

  - *G' has alphabet $\Sigma^{O(\log_q(n))}$.*
  - *If G is satisfiable then G' is satisfiable.*
  - *If $\mathrm{UNSAT}(G) \geq \rho$ then $\mathrm{UNSAT}(G') \geq \Omega\left(\frac{1}{q \cdot \log_q(n)} \cdot \rho\right)$.*

Lemma 5.3.4 (Derandomized Parallel Repetition)**.** *There exist a universal constant h and a polynomial time procedure that satisfy the following requirements:*

- ***Input:***
  - *A finite field $\mathbb{F}$ of size q*
  - *A constraint graph $G = (\mathbb{F}^m, E)$ over alphabet Σ that has a linear structure.*
  - *A parameter $d_0 \in \mathbb{N}$ such that $d_0 < m/h^2$. This parameter will determine the dimension of linear subspaces used in the derandomized parallel repetition, and thus together with q will determine the number of repetitions used in the derandomized parallel repetition.*
  - *A parameter $\rho \in (0,1)$ such that $\rho \geq h \cdot d_0 \cdot q^{-d_0/h}$. Intuitively, the parameter ρ should be chosen such that $1 - \rho$ is an upper bound on the soundness error of G.*
- ***Output:*** *A constraint graph G' such that the following holds:*
  - *G' has size $n^{O(d_0)}$.*
  - *G' has alphabet $\Sigma^{q^{O(d_0)}}$.*
  - *If G is satisfiable then G' is satisfiable.*
  - *If $\mathrm{SAT}(G) < 1 - \rho$ then $\mathrm{SAT}(G') < h \cdot d_0 \cdot q^{-d_0/h}$.*
  - *G' has the projection property.*

We turn to prove the main theorem from the above lemmas.

Theorem (5.1.1, restated)**.** *There exists a constant $\kappa > 0$ such that for every function $\varepsilon : \mathbb{N} \to (0,1)$ satisfying $1/n^{\kappa} \leq \varepsilon(n) \leq 1/\mathrm{poly}\log n$ the following holds: Every language $L \in$* **NP** *has a two-query PCP system with perfect completeness, soundness error $1/\mathrm{poly}\log n$, alphabet size $2^{1/\mathrm{poly}(\varepsilon)}$, proof length* poly (n)*, and randomness complexity $O(\log n)$. Furthermore, the verifier in this PCP system makes only 'projection' queries.*

Proof. Let $\kappa > 0$ be a constant to be chosen later, and let $\varepsilon : \mathbb{N} \to (0,1)$ be a function satisfying $1/n^{\kappa} \leq \varepsilon(n) \leq 1/\mathrm{poly}\log n$. Fix a language $L \in$ **NP**. We show that L has a two-query PCP system with perfect completeness, soundness error $\varepsilon(n)$ and alphabet size $2^{1/\mathrm{poly}(\varepsilon)}$, which has the projection property. By the [FGL+96] correspondence (Proposition 5.2.12), it suffices to show a polynomial time procedure that on input $x \in \{0,1\}^*$, outputs a constraint graph G' of size poly (n) such that the following holds: If $x \in L$ then G' is satisfiable (i.e. $\mathrm{SAT}(G') = 1$), and if $x \notin L$ then $\mathrm{SAT}(G') \leq \varepsilon(n)$. The procedure begins by transforming x, using the PCP theorem for constraint graphs (Theorem 5.2.13), to a constraint graph G of size $n = \mathrm{poly}|x|$ such that if $x \in L$ then $\mathrm{SAT}(G) = 1$ and if $x \notin L$ then $\mathrm{SAT}(G) \leq \varepsilon_0$, where $\varepsilon_0 \in (0,1)$ is a universal constant that does not depend on x. Let $n = \mathrm{poly}(|x|)$ be the size of G, and let $\rho_0 = 1 - \varepsilon_0$.

Next, the procedure sets $\mathbb{F}$ to be the smallest field of size at least $1/(\varepsilon(n))^c$ for some constant $c > 1$ to be determined later, and sets $q = |\mathbb{F}|$. Note that $q \geq \mathrm{poly}\log n$. The procedure now invokes Lemma 5.3.3 (linear structure embedding) on input G and $\mathbb{F}$, thus obtaining a new constraint graph G_1. Note that by Lemma 5.3.3 if $\mathrm{UNSAT}(G) \geq \rho_0$, then $\rho_1 \overset{\mathrm{def}}{=} \mathrm{UNSAT}(G_1) \geq \Omega\left(\frac{1}{q \cdot \log_q(n)} \cdot \rho_0\right)$.

Finally, the procedure sets d_0 to be an arbitrary constant such that $\rho_1 \geq h \cdot d_0 \cdot q^{-d_0/h}$. Note that this is indeed possible, since $\log_q(1/\rho_1)$ is a constant that depends only on ρ (here we use the fact that $q \geq \operatorname{poly}\log n$). Finally, the procedure invokes Lemma 5.3.4 (derandomized parallel repetition) on input G_1, $\mathbb{F}$, ρ_1, and d_0, and outputs the resulting constraint graph G'. We note that we use here the assumption that $\varepsilon(n) \geq n^{\kappa}$, and choose κ to be sufficiently small, in order to guarantee that G_1 satisfies the requirements of Lemma 5.3.4.

It remains to analyze the parameters of G'. It is not hard to see that G' has size $n^{O(d_0)}$ and alphabet $\Sigma^{q^{O(d_0)}} = \Sigma^{1/\operatorname{poly}(\varepsilon)}$. Furthermore, if UNSAT $(G) \geq \rho$, then UNSAT $(G_1) \geq \rho_1$. Therefore, by Lemma 5.3.4 and by the choice of d_0, it holds that $\mathrm{SAT}(G') \leq O(1/q^{\Omega(1)})$. Since $q = 1/(\varepsilon(n))^c$, it holds for sufficiently large c that $\mathrm{SAT}(G') \leq \varepsilon(n)$, as required. ■

Remark 5.3.5. Recall that [MR08] prove a stronger version of the main theorem, saying that for every soundness error $\varepsilon(n) > n^{\kappa}$, not necessarily upper bounded by $1/\operatorname{poly}\log n$, it holds that **NP** has a PCP system with soundness ε and alphabet size $\exp(1/\operatorname{poly}(\varepsilon))$ (Theorem 5.1.2). If one could prove a stronger version of Lemma 5.3.3 (Linear Structure Embedding) in which the soundness of G' is $\rho/\operatorname{poly}(q)$ and the alphabet size is $|\Sigma|^{\operatorname{poly}(q)}$ then the stronger Theorem 5.1.2 would follow using the same proof as above, without using a composition technique as in [MR08, DH09], by choosing q to be sufficiently small.

Remark 5.3.6. The reduction described in Theorem 5.1.1 yields graphs of polynomial size, but not of nearly-linear size as in [MR08] (see Remark 5.2.6). In fact, the construction of graphs with linear structure (Lemma 5.3.3) is nearly linear size (taking an instance of size n to an instance of size $q^2 \cdot n$). The part that incurs a polynomial and not nearly-linear blow-up is the derandomized parallel repetition (Lemma 5.3.4) that relies on the derandomized direct product. It is possible that a more efficient derandomized direct product may lead to a nearly-linear size construction in total.

5.4 PCPs with Linear Structure

In this section we prove Lemma 5.3.3 (linear structure embedding), which implies Theorem 5.1.4 (the existence of PCPs with linear structure) by combining it with the PCP theorem (Theorem 5.2.13). The lemma which says that every constraint graph can be transformed into one that has linear structure. To this end, we use a family of structured graphs called de-Bruijn graphs. We show that de-Bruijn graphs have linear structure, and that every constraint graph can be embedded in some sense on a de-Bruijn graph. This embedding technique is a variant of a technique introduced by Babai et. al. [BFLS91] and Polishchuk and Spielman [PS94] for embedding circuits on de-Bruijn graphs. We begin by defining de-Bruijn graphs.

Definition 5.4.1. Let Λ be a finite alphabet and let $m \in \mathbb{N}$. The *de Bruijn graph* $\mathcal{DB}_{\Lambda,m}$ is the directed graph whose vertices set is Λ^m such that each vertex $(\alpha_1, \ldots, \alpha_m) \in \Lambda^m$ has outgoing edges to all the vertices of the form $(\alpha_2, \ldots, \alpha_m, \beta)$ for $\beta \in \Lambda$.

Remark 5.4.2. We note that previous works used a slightly different notion, the "wrapped de Bruijn graph", which is a layered graph in which the edges between layers are connected as in the de Bruijn graph. Also, we note that previous works fixed Λ to be the binary alphabet, while we we use a general alphabet.

Lemma 5.3.3 follows easily from the following two propositions. Proposition 5.4.3 says that de Bruijn graphs have linear structure. Proposition 5.4.4 says that any constraint graph can be embedded on a de Bruijn graph.

Proposition 5.4.3. *Let $\mathbb{F}$ be a finite field and let $m \in \mathbb{N}$. Then, the de Bruijn graph $\mathcal{DB}_{\mathbb{F},m}$ has linear structure.*

Proof. Items 1 and 3 of the definition of linear structure (Definition 5.3.1) follow immediately from the definition of de Bruijn graphs. To see that Item 2 holds, observe that in order for a tuple in $\mathbb{F}^{2m}$ to be an edge of $\mathcal{DB}_{\mathbb{F},m}$, it only needs to satisfy equality constraints, which are in turn linear constraints. Thus, the set of edges of $\mathcal{DB}_{\mathbb{F},m}$ form a linear subspace of $\mathbb{F}^{2m}$. ■

Proposition 5.4.4 (Embedding on de-Bruijn graphs)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***
  - *A constraint graph G of size n over alphabet Σ.*
  - *A finite alphabet Λ.*
  - *A natural number m such that $|\Lambda|^m \geq 2 \cdot n$*
- ***Output:*** *A constraint graph G' such that the following holds:*
  - *The underlying graph of G' is the de Bruijn graph $\mathcal{DB}_{\Lambda,m}$.*
  - *The size of G' is $|\Lambda|^{m+1}$.*
  - *G' has alphabet $\Sigma^{O(m)}$.*
  - *If G is satisfiable then G' is satisfiable.*
  - *If $\mathrm{UNSAT}(G) \geq \rho$ then $\mathrm{UNSAT}(G') \geq \Omega\left(\frac{n}{|\Lambda|^{m+1} \cdot m} \cdot \rho\right)$.*

Lemma 5.3.3 (linear structure embedding) is obtained by invoking Proposition 5.4.4 with $\Lambda = \mathbb{F}$, $m = \lceil \log_q (2 \cdot n) \rceil$ and combining it with Proposition 5.4.3. The rest of this section is devoted to proving Proposition 5.4.4, and is organized as follows: In Section 5.4.1 we give the required background on the routing properties of de Bruijn graphs. Then, in Section 5.4.2, we give an outline of the proof of Proposition 5.4.4. Finally, we give the full proof of the proposition in Section 5.4.3.

5.4.1 de Bruijn graphs as routing networks

The crucial property of the de Bruijn graphs that we use is that the de Bruijn graph is a permutation routing network. To explain the intuition that underlies this notion, let us think of the vertices of the de Bruijn graph as computers in a network, such that two computers can communicate if and only if they are connected by an edge. Furthermore, sending a message from a computer to its neighbor takes one unit of time. Suppose that each computer in the network wishes to send a message to some other computer in the network, and furthermore each computer needs to receive a message from exactly one computer (that is, the mapping from source computers to target computers is a permutation). Then, the routing property of the de Bruijn network says that we can find paths in the network that have the following properties:

1. Each path corresponds to a message that needs to be sent, and goes from the message's source computer to its target computer.
2. If all the messages are sent simultaneously along their corresponding paths, then at each unit of time, each computer processes exactly one message. By "processing" we mean that the computer receives the message from one of its neighbors and sends it to one of its neighbors.

3. The paths are of length exactly $2 \cdot m$. This means that if all the messages are sent simultaneously along their corresponding paths, then after $2 \cdot m$ units of time all the messages will reach their destination.

Formally, this property can be stated as follows.

Fact 5.4.5. *Let $\mathcal{DB}_{\Lambda,m}$ be a de-Brujin graph. Then, given a permutation μ on the vertices of $\mathcal{DB}_{\Lambda,m}$ one can find a set of undirected paths of length $l = 2m$ which connect each vertex v to $\mu(v)$ and which have the following property: For every $j \in [l]$, each vertex v is the j-th vertex of exactly one path. Furthermore, finding the paths can be done in time that is polynomial in the size of $\mathcal{DB}_{\Lambda,m}$.*

Fact 5.4.5 is proved in [Lei92] for the special case of $\Lambda = \{0, 1\}$. The proof of the general case essentially follows the original proof, except that the looping algorithm of Beneš is replaced with the decomposition of d-regular graphs to d perfect matchings. The proof of the general case can be found in [DM10, App. B].

Remark 5.4.6. Note that the paths mentioned in Fact 5.4.5 are undirected. That is, if a vertex u appears immediately after a vertex v in path, then either (u, v) or (v, u) are edges of $\mathcal{DB}_{\Lambda,m}$.

5.4.2 Proof overview

Suppose we are given as input a constraint graph G which we want to embed on $\mathcal{DB} = \mathcal{DB}_{\Lambda,m}$. Recall that the size of G is at most $|\Lambda|^m$, so we may identify the vertices of G with some of the vertices of $\mathcal{DB}$.

Handling degree 1 As a warm up, assume that G has degree 1, i.e., G is a perfect matching. In this case, we construct G' as follows. We choose the alphabet of G' to be Σ^l for $l \overset{\text{def}}{=} 2m$. Fix any assignment π to G. We describe how to construct a corresponding assignment π' to G'. We think of the vertices of G as computers, such that each vertex v wants to send the value $\pi(v)$ as a message to its unique neighbor in G. Using the routing property of the de Bruijn graph, we find paths for routing those messages along the edges of G'. Recall that if all the messages are sent simultaneously along those paths, then every computer has to deal with one packet at each unit of time, for l units of time. We now define the assignment π' to assign each vertex v of G' a tuple in Σ^l whose j-th element is the message with which v deals at the j-th unit of time.

We define the constraints of G' such that they verify that the routing is done correctly. That is, if the computer u is supposed to send a message to a vertex v between the j-th unit of time and the $(j + 1)$-th unit of time, then the constraint of the edge between u and v checks that $\pi'(u)_j = \pi'(v)_{j+1}$. Furthermore, for each edge (u, v) of G, the constraints of G' check that the values $\pi'(v)_l$ and $\pi'(v)_1$ satisfy the edge (u, v). This condition should hold because if π' was constructed correctly according to π then $\pi'(v)_l = \pi(u)$ and $\pi'(v)_1 = \pi(v)$. It should be clear that the constraints of G' "simulate" the constraints of G. We discuss the exact behavior of the soundness error in the detailed proof.

Handling arbitrary degree graphs Using the expander replacement technique of Papadimitriou and Yannakakis [PY91], we may assume that G is d-regular for some universal constant d. The d-regularity of G implies that the edges of G can be partitioned to d disjoint perfect matchings $\mu_1, \ldots, \mu_d$ in polynomial time (see, e.g., [Cam98, Proposition 18.1.2]). Now, we set the alphabet of G' to be $(\Sigma^l)^d$, and handle each of the matchings μ_i as before, each time using a "different part" of the alphabet symbols. In other words, the alphabet of G' consists of d-tuples of Σ^l, and so the constraints used to handle each matching μ_i will refer to the i-th coordinates in those tuples. Finally, for vertex v, its constraints will also check that the message it sends in each of the d paths is the same. In other words, if $\pi'(v) = (\sigma_1, \ldots, \sigma_d) \in \left(\Sigma^l\right)^d$ then the constraints will check that $(\sigma_1)_1 = \ldots = (\sigma_d)_1$. As before, the constraints of resulting graph G' "simulate" the constraints of the original graph G.

Remark 5.4.7. Observe that the foregoing proof used only the routing property of de Bruijn graphs, and will work for any graph that satisfies this property. In other words, Proposition 5.4.4 (embedding on de-Bruijn graphs) holds for any graph for which Fact 5.4.5 holds.

5.4.3 Detailed proof

We use the following version of the expander-replacement technique of [PY91].

Lemma 5.4.8 ([Din07, Lemma 3.2]). *There exist universal constants $c, d \in \mathbb{N}$ and a polynomial time procedure that when given as input a constraint graph G of size n outputs a constraint graph G' of size $2 \cdot d \cdot n$ over alphabet Σ such that the following holds:*

- *G' has $2 \cdot n$ vertices and is d-regular.*
- *If G is satisfiable then so is G'.*
- *If $\mathrm{UNSAT}(G) \geq \rho$ then $\mathrm{UNSAT}(G') \geq \rho/c$.*

We turn to proving Proposition 5.4.4 (embedding on de-Bruijn graphs). When given as input a constraint graph G, a finite alphabet Λ and a natural number m such that $|\Lambda^m| \geq 2 \cdot n$, the procedure of Proposition 5.4.4 acts as follows. The procedure begins by invoking Lemma 5.4.8 on G, resulting in a d-regular constraint graph G_1 over $2 \cdot n$ vertices. Then, the vertices of G_1 are identified with a subset of the vertices of $\mathcal{DB} = \mathcal{DB}_{\Lambda,m}$ (note that this is possible since $|\Lambda^m| \geq 2 \cdot n$).

Next, the procedure partitions the edges of G_1 to d disjoint perfect matchings, and views those matchings as permutations $\mu_1, \ldots, \mu_d$ on the vertices of $\mathcal{DB}$ in the following way: Given a vertex v of $\mathcal{DB}$, if v is identified with a vertex of G_1 then μ_i maps v to its unique neighbor in G via the i-th matching, and otherwise μ_i maps v to itself. The procedure then applies Fact 5.4.5 to each permutation μ_i resulting in a set of paths $\mathcal{P}_i$ of length $l \stackrel{\text{def}}{=} 2m$. Let $\mathcal{P} = \bigcup \mathcal{P}_i$.

Finally, the procedure constructs G' in the following way. We set the alphabet of G' to be $\Sigma^{l \cdot d}$, viewed as $\left(\Sigma^l\right)^d$. If $\sigma \in \left(\Sigma^l\right)^d$, and we denote $\sigma = (\sigma_1, \ldots, \sigma_d)$, then we denote by $\sigma_{i,j}$ the element $(\sigma_i)_j \in \Sigma$. To define the constraints of G', let us consider their action on an assignment π' of G'. An edge (u, v) of $\mathcal{DB}'$ is associated with the constraint that accepts if and only if all the following conditions hold:

1. For every $i \in [d]$, the values $\left(\pi'(u)_{i,l}, \pi'(u)_{i,1}\right)$ satisfy the edge $\left(\mu_i^{-1}(u), u\right)$ of G.
2. It holds that $\pi'(u)_{1,1} = \ldots = \pi'(u)_{d,1}$ and that $\pi'(v)_{1,1} = \ldots = \pi'(v)_{d,1}$.
3. For every $i \in [d]$ and $j \in [l-1]$ such that u and v are the j-th and $(j+1)$-th vertices of a path in $p \in \mathcal{P}_i$ respectively, it holds that $\pi'(u)_{i,j} \neq \pi'(v)_{i,j+1}$.
4. Same as Condition 3, but when v is the j-th vertex of p and u is its $(j+1)$-th vertex.

The size of G' is indeed $|\Lambda|^{m+1}$, since the graph is $|\Lambda|$-regular and contains $|\Lambda|^m$ vertices. Furthermore, if G is satisfiable, then so is G': The satisfiability of G implies the satisfiability of G_1, so there exists a satisfying assignment π_1 for G_1. We construct a satisfying assignment π' from π_1 by assigning each vertex v of G' a value $\pi'(v)$, such that for each $i \in [d]$, if v is the j-th vertex of a path $p \in \mathcal{P}_i$ that connects the vertices u and $\mu_i(u)$, then we set $\pi'(v)_{i,j} = \pi_1(u)$. Note that this is well defined, since every vertex is the j-th vertex of exactly one path in $\mathcal{P}_i$.

It remains to analyze the soundness of G'. Suppose that $\mathrm{UNSAT}(G) \geq \rho$. Then, by Lemma 5.4.8 it holds that $\mathrm{UNSAT}(G_1) \geq \rho/c$. Let π' be an assignment to G' that minimizes the fraction of violated edges of G'. Without loss of generality, we may assume that for every vertex v of the $\mathcal{DB}$ it holds that $\pi'(v)_{1,1} = \ldots = \pi'(v)_{d,1}$: If there is a vertex v that does not match this condition, all of the edges attached to v are violated and therefore we can modify the $\pi'(v)$ to match this condition without increasing the fraction of violated edges of π'. Define an assignment π_1 to G_1 by setting $\pi_1(v) = \pi'(v)_{1,1}$ (when v is viewed as a vertex of $\mathcal{DB}$).

Since $\mathrm{UNSAT}(G_1) \geq \rho/c$, it holds that π_1 violates at least ρ/c fraction of the edges of G_1, or in other words π_1 violates at least $\rho \cdot 2 \cdot n \cdot d/c$ edges of G_1. Thus, there must exist a permutation μ_i such that π_1 violates at least $\rho \cdot 2 \cdot n/c$ edges of G_1 of the form $(u, \mu_i(u))$. Fix such an edge $(u, \mu_i(u))$ and consider the corresponding path $p \in \mathcal{P}_i$. Observe that π' must violate at least one of the edges of p: To see it, note that if π' would satisfy all the edges on p, then it would imply that $\pi'(\mu_i(u))_{i,l} = \pi_1(u)$ and that $\pi'(\mu_i(u))_{i,1} = \pi_1(\mu_i(u))$, but the last two values violate the edge $(u, \mu_i(u))$ of G_1, and therefore π' must violate the last edge of p - contradiction. It follows that for each of the $\rho \cdot 2 \cdot n/c$ edges of the matching μ_i that are violated by π_1 it holds that π' violates at least one edge of their corresponding path. By averaging there must exist $j \in [l]$ such that for at least $\rho \cdot 2 \cdot n/c \cdot l$ edges of the matching μ_i it holds that π' violates the j-th edge of their corresponding path.

Now, by the definition of the paths in $\mathcal{P}_i$, no edge of G' can be the j-th edge of two distinct paths in $\mathcal{P}_i$, and therefore it follows that there at least $\rho \cdot 2 \cdot n/c \cdot l$ edges of G' are violated by π'. Finally, there are $|\Lambda|^{m+1}$ edges in G', and this implies that π' violates a fraction of the edges of G' that is at least

$$\frac{\rho \cdot 2 \cdot n/c \cdot l}{|\Lambda|^{m+1}} = \Omega\left(\frac{n}{|\Lambda|^{m+1} \cdot l} \cdot \rho\right),$$

as required. ■

5.5 Derandomized Parallel Repetition of Constraint Graphs with Linear Structure

In this section we prove Lemma 5.3.4, restated below, by implementing a form of derandomized parallel repetition on graphs that have linear structure.

Lemma 5.5.1 (5.3.4, restated). *There exist a universal constant h and a polynomial time procedure that satisfy the following requirements:*

- ***Input:***
  - *A finite field $\mathbb{F}$ of size q*
  - *A constraint graph $G = (\mathbb{F}^m, E)$ over alphabet Σ that has a linear structure.*
  - *A parameter $d_0 \in \mathbb{N}$ such that $d_0 < m/h^2$. This parameter will determine the dimension of linear subspaces used in the derandomized parallel repetition, and thus together with q will determine the number of repetitions used in the derandomized parallel repetition.*
  - *A parameter $\rho \in (0,1)$ such that $\rho \geq h \cdot d_0 \cdot q^{-d_0/h}$. Intuitively, the parameter ρ should be chosen such that $1-\rho$ is an upper bound on the soundness error of G.*
- ***Output:*** *A constraint graph G' such that the following holds:*
  - *G' has size $n^{O(d_0)}$.*

- *G' has alphabet $\Sigma^{q^{O(d_0)}}$.*
- *If G is satisfiable then G' is satisfiable.*
- *If $\mathrm{SAT}(G) < 1-\rho$ then $\mathrm{SAT}(G') < h \cdot d_0 \cdot q^{-d_0/h}$.*
- *G' has the projection property*

The basic idea of the proof is as follows. G' contains two kinds of vertices: the first kind corresponds to small subspaces of the vertices space $\mathbb{F}^m$, and of the other kind corresponds to small subspaces of the edges space E, where in both cases "small subspaces" means $O(d_0)$-dimensional subspaces. A satisfying assignment Π to G' is expected to be constructed in the following way: Take a satisfying assignment π to G. For each vertex of G' which is a subspace A of vertices, the assignment Π should assign A to $\pi_{|A}$. For each vertex of G' which is a subspace F of edges, the assignment Π should assign F to $\pi_{|\mathrm{left}(F)\cup\mathrm{right}(F)}$.

The edges of G' are constructed so as to simulate a test on Π to which we refer as the "E-test", and acts roughly as follows (see Figure 5.2 for the actual test): Choose a random subspace F of edges and a random subspace A of endpoints of F, and accept if and only if the labeling of the endpoints of the edges in F by $\Pi(F)$ satisfies the edges and is consistent with the labeling of the vertices of A by $\Pi(A)$.

The intuition that underlies the soundness analysis of G' is the following: The E-test performs some form of a "derandomized direct product test" on Π - if we compare it to the P-test (Figure 5.1), then the pair (A, F) here is analogous to the pair (A, B) there. Therefore, if $\Pi(F)$ is consistent with $\Pi(A)$, the labeling $\Pi(F)$ should be roughly consistent with some assignment π to G. Therefore, by checking that the labeling $\Pi(F)$ satisfies the edges in F, the E-test checks that π satisfies many edges of π in parallel. In this sense, the E-test can be thought as a form of "derandomized parallel repetition".

The rest of this section is organized as follows. In Section 5.5.1 we provide a formal description of the construction of G' and analyze all its parameters except for the soundness. In order to analyze the soundness of G', we introduce in Section 5.5.2 a specialized direct product test. Finally, in Section 5.5.3, we analyze the soundness of G' by reducing it to the analysis of the specialized direct product test.

Notation 5.5.2. Given a function $f : U \to \Sigma$ and two subsets $S, T \subseteq U$ we denote by $f_{|(S,T)}$ the pair of functions $\left(f_{|S}, f_{|T}\right)$.

Notation 5.5.3. Recall that in Notation 5.2.1 we denoted the notation $f \overset{\alpha}{\approx} g$ ($f \overset{\alpha}{\not\approx} g$) to mean that f and g differ on at most (more than) α fraction of the elements of U. We now extend this notation to pairs of functions. Given two pairs of functions $f_1, f_2 : U \to \Sigma$ and $g_1, g_2 : V \to \Sigma$, we denote by $(f_1, g_1) \overset{\alpha}{\approx} (f_2, g_2)$ the fact that both $f_1 \overset{\alpha}{\approx} f_2$ and $g_1 \overset{\alpha}{\approx} g_2$, and otherwise we denote $(f_1, g_1) \overset{\alpha}{\not\approx} (f_2, g_2)$.

5.5.1 The construction of G'

We begin by describing the construction of G'. Let $G = (\mathbb{F}^m, E)$ be the given constraint graph, let d_0 be the parameter from Lemma 5.3.4, and let $d_1 = h \cdot d_0$ where h is the universal constant from Lemma 5.3.4 to be chosen later. The graph G' is bipartite. The right vertices of G' are identified with all the $2d_0$-subspaces of $\mathbb{F}^m$ (the vertex space of G). The left vertices of G' are identified with all the $2d_1$-subspaces of the edge space E of G. An assignment Π to G' should label each $2d_0$-subspace A of $\mathbb{F}^m$ with a function from A to Σ, and each $2d_1$-subspace F of E with a function that maps the endpoints of the edges in F to Σ. The edges of G' are constructed such that they simulate the action of the "E-test" described in Figure 5.2.

The completeness of G' is clear. It is also clear that G' has projection constraints. Let us verify the size and alphabet-size of G'. The size of G' is at most the number of $2d_1$-subspaces of E multiplied by the number of $2d_0$-subspaces of $\mathbb{F}^m$, which is $|E|^{2d_1} \cdot |\mathbb{F}^m|^{2d_0}$. It holds that $d_0 < d_1$, and furthermore the linear structure of G' implies that $\dim E \geq m$ (by Item 3 of Definition 5.3.1), so it follows that

1. Let F_L and F_R be random d_1-subspaces of E, and let

$$B_L \overset{\text{def}}{=} \text{left}(F_L), \quad B_R \overset{\text{def}}{=} \text{right}(F_R), \quad F \overset{\text{def}}{=} F_L + F_R.$$

   F_L and F_R are chosen to be uniformly and independently distributed d_1-subspaces of E conditioned on $\dim(F) = 2d_1$, $\dim(B_L) = d_1$, $\dim(B_R) = d_1$, and $B_L \cap B_R = \{0\}$.

2. Let A_L and A_R be uniformly distributed d_0-subspaces of B_L and B_R respectively, and let
$$A \overset{\text{def}}{=} A_L + A_R.$$

3. Accept if and only if $\Pi(F)_{|(A_L,A_R)} = \Pi(A)_{|(A_L,A_R)}$ and the assignment $\Pi(F)$ satisfies the edges in F.

Figure 5.2: The E-test

1. Choose uniformly distributed pair of independent d_1-subspaces B_1, B_2 of $\mathbb{F}^m$.

2. Choose uniformly distributed pair of d_0-subspaces $A_1 \subseteq B_1$, $A_2 \subseteq B_2$.

3. Accept if and only if $\Pi(B_1, B_2)_{|(A_1,A_2)} = \Pi(A_1 + A_2)_{|(A_1,A_2)}$.

Figure 5.3: The S-test

$|\mathbb{F}^m|^{2d_0} \leq |E|^{2d_1}$ and thus $|E|^{2d_1} \cdot |\mathbb{F}^m|^{2d_0} \leq |E|^{4d_1}$. Finally, observe that the size of G is $n = |E|$, so it follows that the size of G' is at most $n^{4d_1} = n^{O(d_0)}$, as required.

For the alphabet size, recall that an edges subspace F is labeled by a function that maps the endpoints of the edges to Σ. Such a function can be represented by a string in $\Sigma^{2 \cdot q^{2 \cdot d_1}}$, since each $2d_1$-subspace F contains q^{2d_1} edges and each has two endpoints. It can be observed similarly that the labels assigned by Π to $2d_0$-subspaces A of $\mathbb{F}^m$ can be represented by strings in $\Sigma^{2 \cdot q^{2 \cdot d_1}}$. The alphabet of G' is therefore $\Sigma^{2 \cdot q^{2 \cdot d_1}} = \Sigma^{q^{O(d_0)}}$, as required.

5.5.2 The specialized direct product test

In order to analyze the soundness of the E-test, we introduce a variant of the direct product test of [IKW09] that is specialized to our needs. We refer to this variant as the *specialized direct product test*, abbreviated the "S-test".

Given an string $\pi : \mathbb{F}^m \to \Sigma$, we define its *S-direct product* Π (with respect to $d_0, d_1 \in \mathbb{N}$) as follows: Π assigns each $2d_0$-subspace $A \subseteq \mathbb{F}^m$ the function $\pi_{|A}$, and assigns each pair of independent d_1-subspaces (B_1, B_2) the pair of functions $\pi_{|(B_1,B_2)}$.

We turn to consider the task of testing whether a given assignment Π is the S-direct product of some string $\pi : \mathbb{F}^m \to \Sigma$. In our settings, we are given an assignment Π that assigns each $2d_0$-subspace A to a function $a : A \to \Sigma$ and each pair of independent d_1-subspaces (B_1, B_2) to a pair of functions $b_1 : B_1 \to \Sigma$, $b_2 : B_2 \to \Sigma$. We wish to check whether Π is a S-direct product of some $\pi : \mathbb{F}^m \to \Sigma$. To this end we invoke the S-test, described in Figure 5.3.

It is easy to see that if Π is a S-direct product then the S-test always accepts. Furthermore, it can be shown that if Π is "far" from being a S-direct product, then the S-test rejects with high probability. As in the P-test, this holds even if Π is a randomized assignment. Formally, we have the following result.

Theorem 5.5.4 (the soundness of the S-test)**.** *There exist universal constants $h', c \in \mathbb{N}$ such that the following holds: Let $d_0 \in \mathbb{N}$, $d_1 \geq h' \cdot d_0$, and $m \geq h' \cdot d_1$, and let $\varepsilon \geq h' \cdot d_0 \cdot q^{-d_0/h'}$, $\alpha \overset{\text{def}}{=} h' \cdot d_0 \cdot q^{-d_0/h'}$. Suppose that a (possibly randomized) assignment Π passes the S-test with probability at least ε. Then there exists an assignment $\pi : \mathbb{F}^m \to \Sigma$ for which the following holds. Let B_1, B_2 be uniformly distributed and independent d_1-subspaces of $\mathbb{F}^m$, let A_1 and A_2 be uniformly distributed d_0-subspaces of B_1 and B_2 respectively, and denote $A = A_1 + A_2$. Then:*

$$\Pr\left[\Pi\left(B_1, B_2\right)_{|(A_1,A_2)} = \Pi\left(A\right)_{|(A_1,A_2)} \text{ and } \Pi\left(B_1, B_2\right) \overset{\alpha}{\approx} \pi_{|(B_1,B_2)}\right] = \Omega\left(\varepsilon^c\right). \tag{5.4}$$

We defer the proof of Theorem 5.5.4 to Section 5.9.

Remark 5.5.5. Note that Equation 5.4 only says that Π is close to the S-direct product of π on pairs (B_1, B_2), and not necessarily on $2d_0$-subspaces A. In fact, it could be also proved that Π is close to the S-direct product of π on the $2d_0$-subspaces, but this is unnecessary for our purposes.

5.5.3 The soundness of the derandomized parallel repetition

In this section we prove the soundness of G': namely, that if $\text{SAT}\left(G\right) < 1 - \rho$, then

$$\text{SAT}(G') \leq \varepsilon \overset{\text{def}}{=} h \cdot d_0 \cdot q^{-d_0/h},$$

where h is the universal constant from Lemma 5.3.4 (derandomized parallel repetition). We will choose h to be sufficiently large such that the various inequalities in the following proof will hold. To this end, we note that throughout all the following proof, increasing the choice of h does not break any of our assumptions on h, so we can always choose a larger h to satisfy the required inequalities.

Let h' and c be the universal constants whose existence is guaranteed by Theorem 5.5.4 (the soundness of the S-test), and let α denote the corresponding value from Theorem 5.5.4. We will choose the constant h to be at least h'.

Let Π be an assignment to G'. Let us denote by $\mathcal{T}$ the event in which the E-test accepts Π. With a slight abuse of notation, for a subspace $F \subseteq E$ and an assignment $\pi : \mathbb{F}^m \to \Sigma$, we denote by $\Pi\left(F\right) \overset{\alpha}{\approx} \pi$ the claim that for at least $1 - \alpha$ fraction of the edges e of F it holds that $\Pi\left(F\right)$ is consistent with π on both the endpoints of e, and otherwise we denote $\Pi\left(F\right) \overset{\alpha}{\not\approx} \pi$. Our proof is based on two steps:

- We will show (in Proposition 5.5.6 below) that if the test accepts with probability ε, then it is "because" Π is consistent with some underlying assignment $\pi : \mathbb{F}^m \to \Sigma$. This is done essentially by observing that the E-test "contains" an S-test, and reducing to the analysis of the S-test.
- On the other hand, we will show (in Proposition 5.5.7 below) that for every assignment $\pi : \mathbb{F}^m \to \Sigma$ the probability that the test accepts while being consistent with π is negligible. This is done roughly as follows: Any fixed assignment π is rejected by at least ρ fraction of G's edges. Furthermore, the subspace F queried by the test is approximately a uniformly distributed subspace of E, and hence a good sampler of E. It follows F must contain $\approx \rho$ fraction of edges of G that reject π, and therefore $\Pi\left(F\right)$ must be inconsistent with π.

The conclusions of each of the foregoing two steps clearly contradict each other, we therefore conclude that the E-test accepts with probability less than ε. We now state the two said propositions, which formalize the foregoing two steps, and which are proved in Sections 5.5.3.1 and 5.5.3.2 respectively.

Proposition 5.5.6. *There exists $\varepsilon_0 = \Omega\left(\varepsilon^c\right)$ such that the following holds: If $\Pr\left[\mathcal{T}\right] \geq \varepsilon$, then there exists an assignment $\pi : \mathbb{F}^m \to \Sigma$ such that $\Pr\left[\mathcal{T} \text{ and } \Pi\left(F\right) \overset{4\cdot\alpha}{\approx} \pi\right] \geq \varepsilon_0$.*

Proposition 5.5.7. *Let ε be as in Proposition 5.5.6. Then, for every assignment $\pi : \mathbb{F}^m \to \Sigma$ it holds that* $\Pr\left[\mathcal{T} \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi\right] < \varepsilon_0$.

Clearly, the two propositions together imply that $\Pr[\mathcal{T}] \le \varepsilon$, as required.

Before turning to the proofs of Propositions 5.5.6 and 5.5.7 let us state a useful claim that says that if we take a random d-subspace of edges and project it to its left endpoints (respectively, right endpoints), we get a random d-subspace of vertices with high probability.

Claim 5.5.8. *Let $d \in \mathbb{N}$ and let E_a be a uniformly distributed d-subspace of E. Then,* $\Pr[\dim(\text{left}(E_a)) = d] \ge 1 - d/q^{m-d}$, *and conditioned on* $\dim(\text{left}(E_a)) = d$, *it holds that* $\text{left}(E_a)$ *is a uniformly distributed d-subspace of $\mathbb{F}^m$. The same holds for* $\text{right}(E_a)$.

More generally, let E_b be a fixed subspace of E such that $\dim(E_b) > d$ *and* $\dim(\text{left}(E_b)) = D > d$. *Let E_a be a uniformly distributed d-subspace of E_b. Then,* $\Pr[\dim(\text{left}(E_a)) = d] \ge 1 - d/q^{D-d}$, *and conditioned on* $\dim(\text{left}(E_a)) = d$, *it holds that* $\text{left}(E_a)$ *is a uniformly distributed d-subspace of* $\text{left}(E_b)$. *Again, the same holds for* $\text{right}(E_a)$.

Proof. We prove the proposition only for special case in which $E_b = E$ and only for $\text{left}(E_a)$. The proof of the general case and of the case of for $\text{right}(E_a)$ is analogous. Let $e_1, \ldots, e_d$ be independent and uniformly distributed vectors of E, and let $E'_a = \text{span}\{e_1, \ldots, e_d\}$. We prove Proposition 5.5.8 by showing that E_a is distributed similarly to E'_a, and analyzing the distribution of E'_a.

Observe that by Proposition 5.2.17, it holds that conditioned on $\dim(E'_a) = d$, the subspace E'_a is a uniformly distributed d-subspace of E. It therefore holds that

$$\begin{aligned} \Pr[\dim(\text{left}(E_a)) = d] &= \Pr[\dim(\text{left}(E'_a)) = d | \dim(E'_a) = d] \\ &\ge \Pr[\dim(\text{left}(E'_a)) = d \text{ and } \dim(E'_a) = d] \\ &= \Pr[\dim(\text{left}(E'_a)) = d], \end{aligned}$$

where the last equality holds since clearly $\dim(\text{left}(E'_a)) = d$ implies $\dim(E'_a) = d$. Now, since $\text{left}(\cdot)$ is a linear function, it holds that $\text{left}(e_1), \ldots \text{left}(e_d)$ are independent and uniformly distributed vectors of $\text{left}(E) = \mathbb{F}^m$, and therefore by Proposition 5.2.17 it holds that $\Pr[\dim(\text{left}(E'_a)) = d] \ge 1 - d/q^{m-d}$. It thus follows that $\Pr[\dim(\text{left}(E_a)) = d] \ge 1 - d/q^{m-d}$, as required.

It remains to show that conditioned on $\Pr[\dim(\text{left}(E_a)) = d]$ it holds that $\text{left}(E_a)$ is a uniformly distributed d-subspace of $\mathbb{F}^m$. To see it, observe that for every fixed d-subspace D of $\mathbb{F}^m$, it holds that

$$\begin{aligned} \Pr[\text{left}(E_a) = D | \dim(\text{left}(E_a)) = d] &= \Pr[\text{left}(E'_a) = D | \dim(E'_a) = d \text{ and } \dim(\text{left}(E'_a)) = d] \\ &= \Pr[\text{left}(E'_a) = D | \dim(\text{left}(E'_a)) = d], \end{aligned}$$

where the first equality again holds since conditioned on $\dim(E'_a) = d$ it holds that E'_a is a uniformly distributed d-subspace, and the second equality again holds since $\dim(\text{left}(E'_a)) = d$ implies $\dim(E'_a) = d$. Now, it holds that $\text{left}(E'_a)$ is the span of d uniformly distributed vectors of $\mathbb{F}^m$, and therefore by Proposition 5.2.17 it holds that conditioned on $\dim(\text{left}(E'_a)) = d$ the subspace $\text{left}(E'_a)$ is a uniformly distributed d-subspace of $\text{left}(E_b)$. This implies that the probability

$$\Pr[\text{left}(E'_a) = D | \dim(\text{left}(E'_a)) = d]$$

is the same for all possible choices of D, and therefore the probability

$$\Pr[\text{left}(E_a) = D | \dim(\text{left}(E_a)) = d]$$

is the same for all possible choices of D, as required. ■

5.5.3.1 Proof of Proposition 5.5.6

Suppose that $\Pr[\mathcal{T}] \geq \varepsilon$. We prove Proposition 5.5.6 by arguing that the E-test contains an "implicit S-test" and applying Theorem 5.5.4 (the soundness of the S-test).

Observe that, without loss of generality, we may assume that for every edge-subspace F such that $\Pi(F)$ violates one of the edges in F, it holds that $\Pi(F)_{(A_L,A_R)} \neq \Pi(A)_{(A_L,A_R)}$ for any choice of A_L and A_R. The reason is that for every such F, we can modify $\Pi(F)$ such that it assigns symbols outside of the alphabet Σ of G, so $\Pi(F)$ will always disagree with $\Pi(A)$. Note that this modification indeed does not change the acceptance probability of Π. This assumption that we make on Π implies in particular that the event $\mathcal{T}$ is equivalent to the event $\Pi(F)_{(A_L,A_R)} \neq \Pi(A)_{(A_L,A_R)}$, and this equivalence is used in the following analysis.

We turn back to the proof of Proposition 5.5.6. We begin the proof by extending Π to pairs of independent d_1-subspaces of $\mathbb{F}^m$ in a randomized manner as follows: Given a pair of independent d_1-subspaces B_1 and B_2, we choose F_1 and F_2 to be uniformly distributed and independent d_1-subspaces of E such that $\text{left}(F_1) = B_1$ and $\text{right}(F_2) = B_2$, and set $\Pi(B_1, B_2) = \Pi(F_1 + F_2)_{|(B_1,B_2)}$.

Now, observe that the probability that the E-test accepts equals to the probability that the S-test accepts the extended Π. The reason is that the subspaces B_L, B_R, A_L, A_R of the E-test are distributed like the subspaces B_1, B_2, A_1, A_2 of the S-test. It thus follows the E-test performs in a way an S-test on the extended assignment Π.

Next, we note that by choosing h to be sufficiently large, the foregoing "implicit S-test" matches the requirements of Theorem 5.5.4 (the soundness of the S-test), and we can thus apply this theorem. It follows that there exists an assignment $\pi : \mathbb{F}^m \to \Sigma$ such that

$$\Pr\left[\Pi(B_L, B_R)_{(A_L,A_R)} = \Pi(A)_{|(A_L,A_R)} \text{ and } \Pi(B_L, B_R) \overset{\alpha}{\approx} \pi_{(B_L,B_R)}\right] \geq \Omega(\varepsilon^c). \tag{5.5}$$

By using the equivalence between the event $\mathcal{T}$ and the event $\Pi(F)_{(A_L,A_R)} \neq \Pi(A)_{(A_L,A_R)}$, it follows that Inequality 5.5 is equivalent to the inequality

$$\Pr\left[\mathcal{T} \text{ and } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{|(B_L,B_R)}\right] \geq \Omega(\varepsilon^c). \tag{5.6}$$

We turn to show that

$$\Pr\left[\mathcal{T} \text{ and } \Pi(F) \overset{4\alpha}{\approx} \pi\right] \geq \Omega(\varepsilon^c).$$

We will prove that if F is such that $\Pi(F) \overset{4\alpha}{\not\approx} \pi$, then for a random choice of B_L, B_R conditioned on F, it is highly unlikely that Inequality 5.6 still holds. Formally, we will prove the following.

Claim 5.5.9. *For every fixed $2d_0$-subspace F_0 of E such that $\Pi(F_0) \overset{4\alpha}{\not\approx} \pi$, it holds that*

$$\Pr\left[\Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{|(B_L,B_R)} \middle| F = F_0\right] \leq 1/\left(q^{d_1-2} \cdot \alpha^2\right).$$

We defer the proof of Claim 5.5.9 to the end of this section. Claim 5.5.9 immediately implies the following.

Corollary 5.5.10. *It holds that*

$$\Pr\left[\Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{|(B_L,B_R)} \middle| \Pi(F) \overset{4\alpha}{\not\approx} \pi\right] \leq 1/\left(q^{d_1-2} \cdot (\alpha/2)^2\right).$$

By combining Corollary 5.5.10 with Inequality 5.6, and by choosing h to be sufficiently large, it follows that

$$\Pr\left[\mathcal{T} \text{ and } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{|(B_L,B_R)} \text{ and } \Pi(F) \overset{4\alpha}{\approx} \pi\right] \geq \Omega(\varepsilon^c).$$

This implies that

$$\Pr\left[\mathcal{T} \text{ and } \Pi(F) \overset{4\alpha}{\approx} \pi\right] \geq \Omega(\varepsilon^c).$$

Setting ε_0 to be the latter lower bound finishes the proof. ■

Proof of Claim 5.5.9. Observe that the assumption $\Pi(F_0) \overset{4\alpha}{\not\approx} \pi$ implies that one of the following holds

$$\Pi(F_0)_{|\text{left}(F_0)} \overset{2\alpha}{\not\approx} \pi_{|\text{left}(F_0)},$$

$$\Pi(F_0)_{|\text{right}(F_0)} \overset{2\alpha}{\not\approx} \pi_{|\text{right}(F_0)}.$$

Without loss of generality, assume that the first holds. Now, when conditioning on $F = F_0$, it holds that F_L is a uniformly distributed d_1-subspace of F_0 satisfying $\dim(\text{left}(F_L)) = d_1$. By Claim 5.5.8 (with $E_b = F_0$ and $E_a = F_L$), under the conditioning on $\dim(\text{left}(F_L)) = d_1$, it holds that $B_L \overset{\text{def}}{=} \text{left}(F_L)$ is a uniformly distributed d_1-subspace of $\text{left}(F_0)$. Therefore, by Lemma 5.2.4 (subspace-point sampler), the event $\Pi(F)_{|B_L} \overset{\alpha}{\not\approx} \pi_{|B_L}$ occurs with probability at least

$$1 - 1/\left(q^{d_1-2} \cdot \left(\alpha - q^{-d_1}\right)^2\right) \geq 1 - 1/\left(q^{d_1-2} \cdot (\alpha/2)^2\right),$$

as required. ■

5.5.3.2 Proof of Proposition 5.5.7

Fix an assignment $\pi : \mathbb{F}^m \to \Sigma$. By assumption it holds that $\text{SAT}(G) < 1 - \rho$, and therefore π must violate a set E^* of edges of G of density at least ρ. Below we will show that at least $\rho/2$ fraction of the edges in F are in E^* with probability greater than $1 - \varepsilon_0$. Now, observe that $\Pi(F)$ cannot satisfy the edges of F and at the same time be consistent with π on the edges in E^*, and hence whenever the latter event occurs it either holds that the E-test fails or that $\Pi(F) \overset{\rho/2}{\not\approx} \pi$. However, for sufficiently large choice of h, it holds that $\rho/2 > 4 \cdot \alpha$, and therefore the probability that the E-test passes and at the same time it holds that $\Pi(F) \overset{4\cdot\alpha}{\approx} \pi$ is less than ε_0, as required.

It remains to show that

$$\Pr\left[\frac{|F \cap E^*|}{|F|} \geq \rho/2\right] > 1 - \varepsilon_0.$$

We prove the above inequality by showing that F is close to being a uniformly distributed $2d_1$-subspace of E, and then applying Lemma 5.2.4 (subspace-point sampler). To this end, let F'_L and F'_R be uniformly distributed d_1-subspaces of F, and let $F' = F'_L + F'_R$. Let us denote by $\mathcal{E}_1$ the event in which $\dim(F') = 2d_1$, and by $\mathcal{E}_2$ the event in which $\text{left}(F'_L)$ and $\text{right}(F'_R)$ are independent and are of dimension d_1. Observe that conditioned on $\mathcal{E}_1$ and $\mathcal{E}_2$ the subspace F' is distributed exactly like the subspace F. It therefore holds that

$$\begin{aligned}
\Pr\left[\frac{|F \cap E^*|}{|F|} \geq \rho/2\right] &= \Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \,\middle|\, \mathcal{E}_1 \text{ and } \mathcal{E}_2\right] \\
&\geq \Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \text{ and } \mathcal{E}_2 \,\middle|\, \mathcal{E}_1\right] \\
&\geq \Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \,\middle|\, \mathcal{E}_1\right] - \Pr\left[\neg\mathcal{E}_2 | \mathcal{E}_1\right] \\
&\geq \Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \,\middle|\, \mathcal{E}_1\right] - \frac{\Pr\left[\neg\mathcal{E}_2\right]}{\Pr\left[\mathcal{E}_1\right]}.
\end{aligned}$$

Now, observe that conditioned on $\mathcal{E}_1$, the subspace F' is a uniformly distributed $2d_1$-subspace of E. Thus, by Lemma 5.2.4 (subspace-point sampler) it holds that

$$\Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \middle| \mathcal{E}_1\right] \geq 1 - 1/q^{2d_1-2} \cdot \left(\rho/2 - q^{-2d_1}\right)^2 \geq 1 - 1/q^{2d_1-2} \cdot (\rho/3)^2 .$$

Moreover, by Proposition 5.2.16 it holds that

$$\begin{aligned} \Pr[\mathcal{E}_1] &\geq 1 - 2d_1/q^{\dim E - 2d_1} \\ &\geq 1 - 2d_1/q^{m-2d_1} \\ &\geq \frac{1}{2}, \end{aligned}$$

Finally, we upper bound $\Pr[\neg\mathcal{E}_2]$ by showing that $\Pr[\mathcal{E}_2] \geq 1 - 4d_1/q^{m-2\cdot d_1}$. By Claim 5.5.8 (with $E_b = E$ and $E_a = F'_L, F'_R$) it holds that $\dim(\text{left}(F'_L)) = \dim(\text{right}(F'_R)) = d_1$ with probability at least $1 - 2 \cdot d_1/q^{m-d_1}$. Furthermore, conditioned on the latter event, it holds that $\text{left}(F'_L)$ and $\text{right}(F'_R)$ are uniformly distributed d_1-subspaces of $\mathbb{F}^m$, and it is also easy to see that those subspaces are independent. By Proposition 5.2.16, this implies that conditioned on $\dim(\text{left}(F'_L)) = \dim(\text{right}(F'_R)) = d_1$ the subspaces $\text{left}(F'_L)$ and $\text{right}(F'_R)$ are independent with probability at least $1 - 2d_1/q^{m-2\cdot d_1}$, and hence $\Pr[\mathcal{E}_2] \geq 1 - 4d_1/q^{m-2\cdot d_1}$ as required.

We conclude that that

$$\begin{aligned} \Pr\left[\frac{|F \cap E^*|}{|F|} \geq \rho/2\right] &\geq \Pr\left[\frac{|F' \cap E^*|}{|F'|} \geq \rho/2 \middle| \mathcal{E}_1\right] - \frac{\Pr[\neg\mathcal{E}_2]}{\Pr[\mathcal{E}_1]} \\ &\geq 1 - 1/q^{2\cdot d_1-2} \cdot (\rho/3)^2 - \frac{4 \cdot d_1/q^{m-2\cdot d_1}}{1/2} \\ &= 1 - 1/q^{2\cdot d_1-2} \cdot (\rho/3)^2 - 8 \cdot d_1/q^{m-2\cdot d_1} \\ &> 1 - \varepsilon_0, \end{aligned}$$

where the last inequality holds for sufficiently large choice of h. This concludes the proof. ■

5.6 Decodable PCPs

The PCP theorem says that CircuitSat has a proof system in which the (randomized) verifier reads only $O(1)$ bits from the proof. In known constructions this proof is invariably an *encoding* of a satisfying assignment to the input circuit. Although this is not stipulated by the classical definition of a PCP, the fact that a PCP is really an encoding of a 'standard' NP witness is sometimes useful. Various attempts to capture this behavior gave rise to such objects as PCPs of Proximity (PCPPs) [BSGH$^+$06] or assignment testers [DR06], and more recently to decodable PCPs (dPCPs) [DH09].

Application: alphabet reduction through composition. The notion of dPCPs is useful for reducing the alphabet size of PCPs with small soundness error via composition. They were introduced in [DH09] in an attempt to simplify and modularize the construction of [MR08]. Indeed this notion is a refinement of [MR08]'s so-called "locally decode or reject codes (LDRCs)" which allowed [DH09] prove a generic two-query composition theorem. This theorem allows one to improve parameters of a PCP using any dPCP. The only known construction of a dPCP (until this work) is the so-called "manifold vs. point" construction. In the next sections we give a new construction of a dPCP by adapting the work of the previous sections to a dPCP. Our dPCP can then be plugged into the composition scheme of [DH09] to reprove the result of [MR08]. We sketch this in Section 5.6.5.

Decodable PCPs and PCPs of Proximity (PCPPs). We can define dPCPs for any NP language but we focus on the language CircuitSat since it suffices for our purposes. A dPCP system for CircuitSat is a proof system in which the satisfying assignments of the input circuit are *encoded* into a special "dPCP" format. These encodings can then be both locally verified and locally decoded in a probabilistic manner. In other words, the verifier is given an input circuit as well as oracle access to a proof string, and is able to simultaneously check that the given string is a valid encoding of a *satisfying* assignment, as well as to decode a random symbol in that assignment. The formal definition is given below in Section 5.6.2.

dPCPs are closely related to PCPs of proximity [BSGH$^+$06] or assignment testers [DR06] (to be defined shortly below). In fact dPCPs were first defined in the context of low soundness error to overcome inherent limitations of PCPPs in this parameter range. In this chapter we extend the definition of a dPCP also to the high soundness error range (i.e. matching the parameter range of PCPPs). We call these uniquely decodable PCPs (udPCPs) as opposed to list decodable dPCPs. It is natural to consider such an object in our context since our approach is to reduce the error by parallel repetition. Thus we start with a dPCP with relatively high error and then reduce the error. Uniquely decodable PCPs turn out to be roughly equivalent to PCPPs in the sense that any PCPP can be used to construct a udPCP and vice versa. In retrospect, we find the notion of udPCPs (and dPCPs) just as natural as that of PCPPs. In fact, many known constructions of PCPPs work by implicitly constructing a udPCP and then adding comparison checks.

As mentioned above, our main goal in Sections 5.6, 5.7, and 5.8 is to give a new construction of dPCPs with low soundness error (Theorem 5.1.6). Our construction of dPCPs with low soundness error follows the same steps as our construction of PCPs with low soundness error: In the first step, we construct a dPCP with high soundness error (that is, a udPCP). In the second step, we apply derandomized parallel repetition to the foregoing udPCP to reduce its soundness error to a sub-constant function.

In the following subsections we recall the definitions of PCPPs (Section 5.6.1) and define udPCPs (Section 5.6.2). We then prove the equivalence of PCPPs and udPCPs. Next we state two lemmas that capture the two main steps in constructing dPCPs. This is followed by a proof of Theorem 5.1.6 (construction of dPCPs). Finally, we sketch a proof of Theorem 5.1.2 (the [MR08] result) based on Theorem 5.1.6.

5.6.1 Recalling the definition of PCPPs

PCPs of Proximity (PCPPs) were defined simultaneously in [BSGH$^+$06] and in [DR06] under the name assignment testers. PCPPs allow the verifier to check not only that a given circuit is satisfiable, but also that a given assignment is (close to being) satisfying. They were introduced for various motivations, and in particular, they facilitate composition of PCPs which is important for constructing PCPs with reasonable parameters.

Intuitively, a PCP verifier for CircuitSat is an oracle machine V that is given as input a circuit $\varphi : \{0,1\}^t \to \{0,1\}$, and is also given oracle access to an assignment x to φ and a proof π. The verifier V is required to verify that x is close to a satisfying assignment of φ, and to do so by making only few queries to x and π. For technical reasons, it is often preferable to define V in a different way. In this definition, instead of requiring that V makes few queries to its a oracle and decides according to the answers it gets, we require that V outputs explicitly the queries it intends to make and the predicate ψ it intends to apply to the answers it gets. The advantage of this definition is that it allows us to measure the complexity of the predicate ψ. The formal definitions of PCPP are given below.

Definition 5.6.1 (PCPP verifier)**.** A *PCPP verifier* for CircuitSat is a probabilistic polynomial-time algorithm V that on input circuit $\varphi : \{0,1\}^t \to \{0,1\}$ of size n tosses $r(n)$ coins and generates

1. $q = q(n)$ queries $I = (i_1, \dots, i_q)$ in $[t + \ell]$ (where $\ell = \ell(n)$ and the queries are viewed as coordinates of a string in $\{0,1\}^{t+\ell}$).

2. A circuit $\psi : \{0,1\}^q \to \{0,1\}$ of size at most $s(n)$.

We shall refer to $r(n)$, $q(n)$, $\ell(n)$, and $s(n)$ as the *randomness complexity*, *query complexity*, *proof length*, and *decision complexity* respectively.

Definition 5.6.2 (PCPPs)**.** Let V, $r(n)$, $q(n)$, $\ell(n)$, and $s(n)$, be as in Definition 5.6.1, and let $\rho : \mathbb{N} \to (0,1]$. We say that V is a *PCPP system* for $\textsc{CircuitSat}_{\{0,1\}}$ with *rejection ratio* ρ if the following holds for every circuit $\varphi : \{0,1\}^t \to \{0,1\}$ of size n:

- **Completeness:** For every satisfying assignment x for φ there exists a proof string $\pi_x \in \{0,1\}^\ell$ such that
$$\Pr_{I,\psi}\left[\psi\left((x \circ \pi_x)_{|I}\right) = 1\right] = 1,$$
where I and ψ are the (random) output of $V(\varphi)$.

- **Soundness:** For every $x \in \{0,1\}^t$ that is ε-far from a satisfying assignment to φ and every proof string $\pi \in \{0,1\}^\ell$ the following holds:
$$\Pr_{I,\psi}\left[\psi\left((x \circ \pi)_{|I}\right) = 0\right] \geq \rho \cdot \varepsilon.$$

The starting point for our construction of a dPCP is the fact that NP has PCPPs with reasonable parameters:

Theorem 5.6.3 ([BSGH$^+$06, DR06])**.** $\textsc{CircuitSat}_{\{0,1\}}$ *has a PCPP system with randomness complexity* $O(\log n)$*, query complexity* $O(1)$*, proof length* $\mathrm{poly}(n)$*, decision complexity* $O(1)$*, and rejection ratio* $\Omega(1)$.

Remark 5.6.4. The PCPPs described in Definition 5.6.2 are known in the literature as "strong PCPPs". Here, the term "strong" means that the rejection probability is linearly related to to the distance ε of x from a satisfying assignment. In particular, this implies that even if ε is small (but non-zero), then the PCPP rejects with non-zero probability.

An alternative definition of PCPPs, known as "weak PCPPs", requires only that every assignment $x \in \{0,1\}^t$ that is very far from a satisfying assignment will be rejected with high probability, while x's that are close to a satisfying assignment may be accepted with probability 1.

5.6.2 The definition of decodable PCPs

Decodable PCPs (dPCPs) were defined in the work of [DH09] in order to overcome certain limitations of PCPPs[4]. As mentioned above, the definition of [DH09] is only useful if the soundness error is indeed very low. Below, we recall the definition of [DH09] and suggest an alternative definition for the case where the soundness error is high. This alternative definition will be useful later in the construction of decodable PCPs with low soundness error.

[4]In particular, using arguments in the spirit of [BSHLM09], it is easy to prove that a PCPP that has low soundness error must make at least three queries. Hence, PCPPs can not be used to construct two-query PCPs with low soundness error.

5.6.2.1 Recalling the definition of [DH09]

Intuitively, a PCP decoder for CircuitSat is an oracle machine D that is given as input a circuit φ, and is also given oracle access to a "proof" π that is supposed to be the encoding of some *satisfying* assignment x to φ. The PCP decoder D is required to decode a uniformly distributed coordinate k of the assignment x by making only few queries to π. It could also be the case that the proof π is too corrupted for the decoding to be possible, in which case D is allowed to output a special failure symbol $\perp$. Thus, we say that D has made an error only if it outputs a symbol other than x_k and $\perp$. We refer to the probability of the latter event as the "decoding error of D", and would like it to be minimal. We do note, however, that if π is not corrupted, then D is not allowed to output $\perp$.

It turns out that if we wish the decoding error of D to be very small, we need to relax the foregoing definition, and allow the PCP decoder D to perform "list decoding". That is, instead of requiring that there would be a single assignment x that is decoded by D, we only require that there exists a *short* list of assignments $x^1, \ldots, x^L$ such that the decoder outputs either $\perp$ or one of the symbols $x^1_k, \ldots, x^L_k$ with very high probability. Of course, this is meaningless if the assignments are binary strings, and therefore we extend the definition of CircuitSat to circuits whose inputs are symbols from some large alphabet Γ.

We turn to give the formal definitions of (list-)decodable PCPs. As in the case of PCPPs, instead of letting the decoder make the queries and process the answers directly, we require the decoder to output the queries and a circuit ψ that given the answers to the queries outputs the decoded value.

Notation 5.6.5. Let Σ and Γ be finite alphabets, and let $f : \Gamma^k \to \Sigma^n$ be a function. We say that a circuit C computes f if it takes as input a binary string of length $k \cdot \lceil \log |\Gamma| \rceil$ and outputs a binary string of length $n \cdot \lceil \log |\Sigma| \rceil$ that represent the input in Γ^k and the output in Γ^n in the natural way. We will usually omit the function f and simply refer to the circuit $C : \Gamma^k \to \Sigma^n$. We will also view the circuit C as taking as input k symbols in Γ and outputs n symbols in Σ. Given a circuit $\varphi : \Gamma^t \to \{0, 1\}$, an assignment $x \in \Gamma^t$ for φ is said to *satisfy* φ if $\varphi(x)$, and otherwise it is said to be *unsatisfying.*

Definition 5.6.6 (PCP decoders, similar to [DH09, Definition 3.1])**.** Let $r, q, s, \ell : \mathbb{N} \to \mathbb{N}$, and let Γ, Σ be functions that map each $n \in \mathbb{N}$ to some finite alphabet. A *PCP decoder* for CircuitSat$_\Gamma$ over *proof alphabet* Σ is a probabilistic polynomial-time algorithm D that for every $n \in \mathbb{N}$ acts as follows. Let $\Gamma = \Gamma(n)$, $\Sigma = \Sigma(n)$, $\ell = \ell(n)$. When given as input an input circuit $\varphi : \Gamma^t \to \{0, 1\}$ of size n and an index $k \in [t]$, the PCP decoder D tosses $r(n)$ coins and generates

1. A sequence of queries $I = \left(i_1, \ldots, i_{q(n)}\right)$ in $[\ell]$ (where the queries are viewed as coordinates of a proof string in Γ^ℓ).
2. A circuit $\psi : \Sigma^{q(n)} \to \Gamma \cup \{\perp\}$ of size at most $s(n)$.

We shall refer to the functions $r(n)$, $q(n)$, $\ell(n)$, and $s(n)$ as the *randomness complexity*, *query complexity*, *proof length*, and *decoding complexity* respectively. Without loss of generality we have $\ell\left(n\right) = 2^{r(n)} \cdot q(n) \cdot t$.

Definition 5.6.7 (List Decodable PCPs, similar to [DH09, Definition 3.2])**.** Let D, Γ, Σ, and ℓ be as in Definition 5.6.6, and $L : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0, 1]$. We say that a PCP decoder D with the foregoing parameters is a *(list) decodable PCP system* for CircuitSat$_\Gamma$ (abbreviated ldPCP) with *list size* $L = L(n)$, *soundness error* $\varepsilon = \varepsilon(n)$ if the following holds for every circuit $\varphi : \Gamma^t \to \{0, 1\}$ of size n:

- **Completeness:** For every $x \in \Gamma^t$ such that $\varphi(x) = 1$ there exists a proof string $\pi_x \in \Sigma^\ell$ such that

$$\Pr_{k;I,\psi}\left[\psi\left(\pi_{x|I}\right) = x_k\right] = 1,$$

  where k is uniformly distributed in $[t]$ and I and ψ are the (random) output of $D\left(\varphi, k\right)$.

- **Soundness:** For every proof string $\pi \in \Sigma^{\ell}$, there exist a (possibly empty) list of satisfying assignments $x^1, \ldots, x^L \in \Gamma^t$ for φ such that

$$\Pr_{k;I,\psi}\left[\psi\left(\pi_{|I}\right) \notin \left\{x_k^1, \ldots, x_k^L, \perp\right\}\right] \leq \varepsilon,$$

where k, I, ψ are as before.

5.6.2.2 Uniquely-decodable PCPs

We turn to discuss our suggested definition for dPCPs for the case of high soundness error. If the soundness error is high, then we can actually require the PCP decoder to decode a unique assignment, instead of decoding a list of assignments. Thus, we refer to dPCPs with high soundness error as "uniquely decodable PCPs" (udPCPs).

The straightforward definition for udPCPs would be to take the foregoing definition of ldPCPs, and set ε to be large and L to be 1. However, this definition turns out to be useless for our purposes. To see why, recall that our ultimate goal is to construct dPCPs with *low error* by first constructing dPCPs with *high error* and then decreasing their error using derandomized parallel repetition. However, if we define udPCPs using the above straightforward definition, then it is not even clear that *sequential repetition* decreases their error[5].

We therefore use the following alternative definition for udPCP. We now require that if the proof π is such that the PCP decoder D errs with high probability, then D detects that there is an error with at least proportional probability. In other words, we require that the probability that D outputs $\perp$ is related to the probability that D errs. Observe that such PCP decoders can indeed be improved by sequential repetition: If the proof π is erroneous and we invoke the PCP decoder D many times, then the probability that D detects the error and outputs $\perp$ improves. Below we give the formal definition.

Definition 5.6.8. Let D, Γ, Σ, and ℓ be as in Definition 5.6.6. Let $\varphi : \Gamma^t \to \{0,1\}$ be a circuit of size n, let x be an assignment to φ, and let $\pi \in \Sigma^{\ell(n)}$ be a proof for D. We define the *decoding error of D on π with respect to x* as the probability

$$\Pr_{k;I,\psi}\left[\psi\left(\pi_{|I}\right) \notin \left\{x_k, \perp\right\}\right],$$

where k, I, ψ are as in Definition 5.6.7. We define the *decoding error of D on π* as the minimal decoding error of D on π with respect to an assignment x' for φ, over all possible assignments x' to φ.

Definition 5.6.9 (Uniquely Decodable PCPs)**.** Let D, Γ, Σ, and ℓ be as in Definition 5.6.6, and let $\rho : \mathbb{N} \to [0,1]$. We say that the PCP decoder D is a *(uniquely) decodable PCP system* for $\textsc{CircuitSat}_\Gamma$ (abbreviated udPCP) with *rejection ratio* ρ if for every circuit $\varphi : \Gamma^t \to \{0,1\}$ of size n the PCP decoder D satisfies the completeness requirement of Definition 5.6.7, and furthermore satisfies the following requirement:

- **Soundness:** For every proof string $\pi \in \Sigma^{\ell}$, if D has decoding error ε on π then

$$\Pr_{k;I,\psi}\left[\psi\left(\pi_{|I}\right) = \perp\right] \geq \rho(n) \cdot \varepsilon,$$

where k, I, ψ are as in Definition 5.6.7.

[5]The problem in performing sequential repetition for such definition of udPCPs is that we must invoke the PCP decoder on a *uniformly distributed and independent index k* in each invocation, and it is not clear how to use invocations for different indices k in order to decrease the error.

Remark 5.6.10. We could have also defined the decoding error of D on π with respect to x as the probability $\Pr_{k;I,\psi}\left[\psi\left(\pi_{|I}\right) \neq x_k\right]$. This definition may be more natural, but it is more convenient to work with the current definition.

Remark 5.6.11. Note that the soundness requirement in our definition of udPCPs is similar to the soundness requirement of PCPPs, and in particular to definition of soundness of *strong* PCPPs (see Remark 5.6.4). We could also use a definition that is analogous to the definition of a *weak* PCPP. Specifically, we could have required only that when the decoding error is very large, the decoder rejects with high probability. However, our definition is stronger, and since we can satisfy it, we prefer to work with it. It is also more convenient to work with this definition throughout this chapter.

We next argue that every PCPP implies a udPCP.

Proposition 5.6.12. *Let V be a PCPP system for* $\textsc{CircuitSat}_{\{0,1\}}$ *with randomness complexity $r(n)$, query complexity $q(n)$, proof length $\ell(n)$, decision complexity $s(n)$, and rejection ratio $\rho(n)$. Then, for every $u : \mathbb{N} \to \mathbb{N}$ there exists a udPCP for* $\textsc{CircuitSat}_{\{0,1\}^{u(n)}}$ *with proof alphabet $\{0,1\}$, randomness complexity $r(n)$, query complexity $q(n)+u(n)$, proof length $n+\ell(n)$, decoding complexity $s(n)+O(u(n))$, and rejection ratio $\rho(n)/u(n)$.*

Proof. Let $u : \mathbb{N} \to \mathbb{N}$ and denote $u = u(n)$. For every circuit $\varphi : \left(\{0,1\}^u\right)^t \to \{0,1\}$ of size n and satisfying assignment x for φ, we define the corresponding proof string for D to be $x \circ \pi_x$, where π_x is the proof string of V for x when x is treated as a binary string.

Fix a circuit $\varphi : \left(\{0,1\}^u\right)^t \to \{0,1\}$ and $k \in [t]$, and let $x' \in \{0,1\}^{u\cdot t}$, $\pi \in \{0,1\}^{\ell}$. On input (φ, k) and oracle access to a proof $x' \circ \pi$, the decoder D first emulates the verifier V on φ with oracle access to $x' \circ \pi_x$. If V rejects, then D outputs $\perp$. Otherwise, D queries the coordinates

$$u \cdot (k-1)+1, \ldots, u \cdot k$$

of x and outputs the tuple of answers as the symbol in $\{0,1\}^u$ that it is ought to decode.

It should be clear that D satisfies the completeness requirement, and has the correct randomness complexity, query complexity, proof length, and decoding complexity.

It remains to analyze the rejection ratio of D. Let π' be a proof string for D and assume that $\pi' = x \circ \pi$ where $x \in \{0,1\}^{u\cdot t}$ and $\pi \in \{0,1\}^{\ell}$. Let x_0 be the satisfying assignment of φ that is nearest to x when viewed as a binary string. Let ε be the relative distance between x and x_0 when viewed as strings over the alphabet $\{0,1\}^u$. Clearly, the decoding error of D on $x \circ \pi$ with respect to x_0 is ε, and is an upper bound on the decoding error of D. Furthermore, the relative distance between x and x_0 as binary strings is at least ε/u. Thus, the emulation of V rejects $x \circ \pi$ with probability at least $\rho(n) \cdot \varepsilon/u$, and this is also the rejection probability of D, as required. ■

Remark 5.6.13. One could also prove Proposition 5.6.12 without a loss of a factor of u in the rejection ratio ρ using error correcting codes.

Remark 5.6.14. It is not hard to see that the converse of Proposition 5.6.12 also holds. Namely, given a udPCP it is easy to construct from it a PCPP. Roughly, given a udPCP D, construct a PCPP verifier that when given oracle access to $x \circ \pi$, invokes D with oracle access to π on a uniformly distributed k, and verifies that the output of D equals x_k.

Remark 5.6.15. Our definition of udPCPs (Definition 5.6.9) bears some similarities to the notion of relaxed locally decodable codes [BSGH+06], which are also constructed using PCPPs. However, the notions are fundamentally different. The most important difference between the notions is that while the decoder of a relaxed LDC should decode any possible message, the decoder of a udPCP is required to decode only satisfying assignments of a given circuit. This makes udPCPs significantly more powerful, and in fact makes them equivalent to PCPPs. A secondary difference is that when a udPCP is given oracle access to a corrupted oracle then it can output $\perp$ with any probability, while a relaxed LDC is required to output x_k (instead of $\perp$) with some given probability.

5.6.3 Decoding graphs

5.6.3.1 The definition of decoding graphs

Recall that in the first part of the chapter, we often found it more convenient to work with constraint graphs instead of working with PCPs. We now define the notion of "decoding graphs", which will serve as the graph analogue of decoding PCPs just as constraint graphs serve as the graph analogue of PCPs.

Definition 5.6.16 (Decoding graphs)**.** A *(directed) decoding graph* is a directed graph $G = (V, E)$ that is augmented with the following objects:

1. A circuit $\varphi : \Gamma^t \to \{0,1\}$, to which we refer as the *input circuit.* Here Γ denotes some finite alphabet.
2. A finite alphabet Σ, to which we refer as the *alphabet of* G.
3. For each edge $e \in E$, an index $k_e \in [t]$, and a circuit $\psi_e : \Sigma \times \Sigma \to \Gamma \cup \{\perp\}$. We say that e is *associated with* k_e and ψ_e. For $k \in [t]$, we denote by E_k the set of edges associated with k.

The *size* of G is the number of edges of G. We say that G has *decoding complexity* s if all the circuits are of size at most s. It is required that G satisfies the following property:

- **Completeness:** For every satisfying assignment $x \in \Gamma^t$ to φ, there exists an assignment $\pi_x : V \to \Sigma$ to G such that the following holds. For every edge (u, v) that is associated with an index $k = k_{(u,v)}$ and a circuit $\psi = \psi_{(u,v)}$, it holds that $\psi\left(\pi(u), \pi(v)\right) = x_k$.

Notation 5.6.17. We will use the following terminology regarding constraint graphs: Let $G = (V, E)$ be a decoding graph with input circuit $\varphi : \Gamma^t \to \{0,1\}$ alphabet Σ.

1. Let $(u, v) \in E$ and $\psi = \psi_{(u,v)}$ be and edge its associated circuit, and let $\pi : V \to \Sigma$ be an assignment to G. If ψ outputs $\perp$ on input $(\pi(u), \pi(v))$ then we say that (u, v) *rejects* π (or that π violates (u, v)), and otherwise we say that (u, v) *accepts* π (or that π satisfies (u, v)).
2. Let (u, v), ψ, and π be as before, let $k = k_{(u,v)}$ be the index associated with (u, v), and let x be an assignment to φ. We say that (u, v) *fails to decode* x if $\psi\left(\pi(u), \pi(v)\right) \notin \{x_k, \perp\}$. When x is clear from the context we will omit it, and we will also say that (u, v) *errs*, or that (u, v) *decodes correctly* (if (u, v) does not err). Note that outputting $\perp$ is *not considered to be failure.*
3. We say that G has the *projection property* if for every circuit $\psi_{(u,v)}$ has an associated function $f_{(u,v)} : \Sigma \to \Sigma$ such that $\psi_{(u,v)}\left(a, b\right) \neq \perp$ if and only if $f_{(u,v)}(a) = b$.
4. We refer to the quantity $\log\left(\max_{k\in[t]} |E_k|\right)$ as the *randomness complexity* of G, since it upper bounds the number of bits required to choose a uniformly distributed edge that is associated with a particular index.

We turn to define soundness properties of decoding graphs. As in the case of decodable PCPs, we have two definitions, one for the case of high soundness error (unique decoding) and one for the case of low soundness error (list decoding).

Definition 5.6.18. Let $G = (V, E)$, Σ, Γ, φ be as before, and let $\pi : V \to \Sigma$ be an assignment to G.

- **Unique decoding soundness:** For every satisfying assignment $x \in \Gamma^t$ to φ, we define the *decoding error of G on π with respect to x* as the probability

$$\Pr_{k \in [t], (u,v) \in E_k} \left[\psi_{(u,v)} \left(\pi\left(u\right), \pi\left(v\right) \right) \notin \{x_k, \perp\} \right],$$

where k is uniformly distributed in $[t]$ and (u, v) is uniformly distributed in E_k. Note that the edge (u, v) is chosen according to the decoding distribution of G.
We define the *decoding error of G on π* as the minimal decoding error of G on π with respect to any satisfying assignment of φ. Now, we say that G has rejection ratio ρ if for every assignment π to G, if G has decoding error ε on π then it holds that

$$\Pr_{k \in [t], (u,v) \in E_k} \left[\psi_{(u,v)} \left(\pi\left(u\right), \pi\left(v\right) \right) = \perp \right] \geq \rho \cdot \varepsilon,$$

where k and (u, v) are chosen as before.

- **List decoding soundness:** We say that G is *list-decoding* with *list size L* and *soundness error ε* if for every assignment π to G there exists a (possibly empty) list of satisfying assignments $x^1, \ldots, x^L \in \Gamma^k$ for φ such that

$$\Pr_{k \in [t], (u,v) \in E_k} \left[\psi_{(u,v)} \left(\pi\left(u\right), \pi\left(v\right) \right) \notin \left\{ x^1_k, \ldots, x^L_k, \perp \right\} \right] \leq \varepsilon,$$

where k and (u, v) are chosen as before

The following proposition gives the correspondence between decoding PCPs and decoding graphs, in analogy to the correspondence between PCPs and constraint graphs.

Proposition 5.6.19. *Let $r, s, \ell, \rho, \Gamma, \Sigma$ be as in Definition 5.6.9. The following two statements are equivalent:*

- $\textsc{CircuitSat}_\Gamma$ *has a udPCP with query complexity* 2*, randomness complexity* r*, decoding complexity* s*, proof length* ℓ*, proof alphabet* Σ*, and rejection ratio* ρ*.*
- *There exists a polynomial-time transformation that transforms a circuit $\varphi : \Gamma^t \to \{0, 1\}$ of size n to a decoding graph $G = (V, E)$ with $\ell(n)$ vertices, randomness complexity $r(n)$, decoding complexity $s(n)$, proof alphabet $\Sigma(n)$, and rejection ratio $\rho(n)$.*

A similar equivalence holds for ldPCPs and list-decoding graphs.

5.6.3.2 Additional properties of decoding graphs

Recall that when discussing constraint graphs, we were interested in the probability that a uniformly distributed edge of the graph is satisfied by a given assignment. As can be seen in Definition 5.6.18, when discussing decoding graphs we are interested in a different distribution over the edges, defined below.

Definition 5.6.20. The *decoding distribution $\mathcal{D}_G$* of a decoding graph $G = (V, E)$ is the distribution over the edges of G that is corresponds to the following way for picking a random edge of G: Choose $k \in [t]$ uniformly at random, and then choose an edge uniformly at random from E_k.

It is usually inconvenient to analyze the decoding distribution of the graphs we work with. However, we will work only with graphs whose decoding distribution is similar to the uniform distribution over the edges (where similarity is defined as in Section 5.2.5). The following definition aims to capture this property, which allows us to analyze the uniform distribution instead of the decoding distribution.

Definition 5.6.21. We say that a decoding graph $G = (V, E)$ has *smoothness* γ if its decoding distribution is γ-similar to the uniform distribution over E.

The following proposition gives a comfortable way for calculating the smoothness of a decoding graph. Intuitively, observe that if all the sets E_k are of the same size then the decoding distribution is identical to the uniform distribution. We now observe that if the sizes of the sets E_k are close to each other then the decoding distribution is similar to the uniform distribution.

Proposition 5.6.22 (Smoothness criterion)**.** *A decoding graph G with edge-set E has smoothness γ if and only if for every $k \in [t]$, the number of edges that are associated with k is between $\gamma \cdot \frac{|E|}{t}$ and $\frac{1}{\gamma} \cdot \frac{|E|}{t}$.*

Proof. Observe that if there are m_k edges associated with $k \in [t]$ then the probability for such an edge to be chosen under the decoding distribution is $\frac{1}{t} \cdot \frac{1}{m_k}$ while the corresponding probability under the uniform distribution is $\frac{1}{|E|}$. Now apply the definition of similarity of distributions. ■

We will often want our decoding graphs to be regular, or at least have bounded degree. The precise definition follows.

Definition 5.6.23. We say that a decoding graph G has *degree bound* $d \in \mathbb{N}$ if all the in-degrees and all out-degrees of the vertices in G are bounded by d. We say that it is *d-regular* if every vertex has *exactly* d incoming edges and *exactly* d outgoing edges.

5.6.3.3 General udPCPs and decoding graphs

Proposition 5.6.19 gave us only a correspondence between decoding graphs and udPCPs that makes exactly two queries. The next proposition shows that in fact any udPCP, even if it uses more than two queries, gives rise to a procedure that transforms circuits to decoding graphs with related parameters and unique decoding soundness. A nice property of this procedure is that it generates decoding graphs that are regular and have smoothness 1, which will be useful later in this chapter.

Proposition 5.6.24. *Let Γ, Σ, $r(n)$, $q(n)$, $\ell(n)$, $s(n)$, and $\rho(n)$ be as in Definition 5.6.9, and let h_0 and d_0 be the constants from Fact 5.2.20. If there exists a udPCP D for* CircuitSat$_\Gamma$ *with the foregoing parameters, then there exists a polynomial time procedure that acts as follows. When given a circuit $\varphi : \Gamma^t \to \{0,1\}$ of size n, the procedure outputs a corresponding vertex-decoding graph $G = (V, E)$ with randomness complexity $r(n) + \log(d_0 \cdot q(n))$, alphabet $\Sigma^{q(n)}$, decoding complexity $s(n) + \operatorname{poly}\log|\Sigma(n)|$, and rejection ratio $\Omega\left(\rho(n)/(q(n))^2\right)$. Furthermore, G is $(q(n) \cdot d_0)$-regular, and has $t \cdot 2^{r(n)}$ vertices and smoothness* 1.

Proof sketch The proof is a variant of a well known technique for reducing the query complexity of a PCP verifier to 2, and its full details can be found in [DM10, App. D]. The graph G is constructed roughly as follows: The graph G has a vertex for every possible invocation of the decoder D. Each such vertex v is expected to be labeled with the answers that D receives to its queries on the corresponding invocation, and the edges that are connected to v check that those answers are not rejected by D. The edges of G also verify that the labels of the different vertices are consistent with each other, and in order to save in the number of edges we choose the consistency checks according to an expander.

Observe that since a vertex should be labeled with all the answers that D gets to its queries on this particular invocation, we can use those labels to perform decoding. In particular, given that an edge (u, v) accepts, the value that it decodes can be decided based only on the label of u. This property will be useful in Section 5.7 (see Definition 5.7.1 for details). ■

5.6.4 Our construction of dPCPs, Theorem 5.1.6

In this section we state and prove Theorem 5.1.6.

Theorem (5.1.6, dPCP, restated formally)**.** *For every function Γ that maps natural numbers to finite alphabets such that $|\Gamma(n)| \leq 2^{\text{poly}\log n}$ the following holds. There exists an ldPCP D for* $\textsc{CircuitSat}_\Gamma$ *with query complexity* 2*, proof alphabet* $2^{\text{poly}\log n}$*, randomness complexity* $O(\log n)$*, soundness error* $1/\log^{\Omega(1)} n$*, and list size* $\text{poly}\log n$*. Furthermore, D has the projection property (see Notation 5.6.17, Item 3).*

We prove this theorem analogously to the proof of Theorem 5.1.1, which asserts the existence of two-query PCPs with soundness error $1/\text{poly}\log n$. Our starting point is a known construction of a PCPP, stated here as Theorem 5.6.3 which is then reduced to a transformation mapping circuits to decoding graphs. We then have two main steps. The first is to equip the decoding graphs with linear structure, as formulated in Lemma 5.6.25. The second step is to reduce the error by derandomized parallel repetition, as stated in Lemma 5.6.26. Theorem 5.1.6 follows by combining the two lemmas which we state next,

Lemma 5.6.25 (Linear Structure Embedding for udPCPs)**.** *There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***
  - *A decoding graph G of size n for input circuit $\varphi : \Gamma^t \to \{0,1\}$ with alphabet Σ, rejection ratio ρ, decoding complexity s, and smoothness γ.*
  - *A finite field $\mathbb{F}$ of size q such that $q \geq 4 \cdot d_0^2$, where d_0 is the constant from Fact 5.2.20.*
- ***Output:*** *A decoding graph $G' = (\mathbb{F}^m, E')$ for φ such that the following holds:*
  - *G' has a linear structure.*
  - *The size of G' is at most $O(q \cdot n/\gamma)$.*
  - *G' has alphabet $\Sigma^{O(\log_q(n/\gamma))}$.*
  - *G' has rejection ratio $\Omega\left(\rho/q^2 \cdot \log_q(n/\gamma)\right)$*
  - *G' has decision complexity $s + \text{poly}\left(\log_q(n/\gamma), \log|\Gamma|\right)$*
  - *G' has smoothness $\Omega(1/q)$.*

Lemma 5.6.26 (Derandomized Parallel Repetition for dPCPs)**.** *There exist a universal constant h and a polynomial time procedure that satisfy the following requirements:*

- ***Input:***
  - *A finite field $\mathbb{F}$ of size q.*
  - *A decoding graph $G = (\mathbb{F}^m, E)$ of size n for input circuit $\varphi : \Gamma^t \to \{0,1\}$ with linear structure, alphabet Σ, rejection ratio ρ, decision complexity s, and smoothness γ.*
  - *The rejection ratio ρ of G.*
  - *A parameter $d_0 \in \mathbb{N}$ such that $d_0 < m/h^2$ and $\rho \geq h \cdot d_0 \cdot q^{-d_0/h}/\gamma$.*
- ***Output:*** *A decoding graph G' for φ such that the following holds:*
  - *G' has size $n^{O(d_0)}$.*

– *G' has alphabet $\Sigma^{q^{O(d_0)}}$.*

– *G' is list-decoding with soundness error $\varepsilon \overset{\text{def}}{=} h \cdot d_0 \cdot q^{-d_0/h}/\gamma$ and list size $L \overset{\text{def}}{=} q^{O(d_0)}$.*

– *G' has the projection property.*

– *G' has decoding complexity $q^{O(d_0)} \cdot (s + \operatorname{poly}\log|\Sigma|)$.*

We now turn to prove Theorem 5.1.6.

Proof. Let V be a PCPP verifier for CircuitSat as in Theorem 5.6.3. By Proposition 5.6.12 this implies a udPCP for CircuitSat with similar parameters. Next, by Proposition 5.6.24 we get a polynomial time transformation taking a circuit $\varphi : \{0,1\}^n \to \{0,1\}$ into a vertex-decoding graph. The graph G has the following parameters. The randomness complexity is $r(n) = O(\log n)$, the decoding complexity, rejection ratio, and constant proof alphabet are constant, and the smoothness is 1.

We choose $\mathbb{F}$ to be the smallest finite field of size at least $\log n$, and set $\mathbb{F}$ to be the finite field of size q. We now invoke Lemma 5.6.25 (linear structure embedding for udPCPs) on input G and $\mathbb{F}$, and obtain a new vertex-decoding graph G_1 with linear structure and parameters:

- The size of G_1 is at most $O(q \cdot n)$.
- G_1 has alphabet size $2^{O(\log_q(n))}$.
- G_1 has rejection ratio $\rho_1 \overset{\text{def}}{=} \Omega\left(\rho/q^2 \cdot \log_q(n)\right)$
- G_1 has decision complexity $\operatorname{poly}(\log_q n)$
- G_1 has smoothness $\gamma_1 = \Omega\left(\frac{1}{q}\right)$.

Finally, we set d_0 to be an arbitrary constant such that $\rho_1 \geq h \cdot d_0 \cdot q^{-d_0/h}/\gamma_1$. Note that this is indeed possible, since $\log_q(1/\rho_1)$ is a constant that depends only on ρ. Finally, we invoke Lemma 5.6.26 (derandomized parallel repetition for dPCPs) on input G_1, $\mathbb{F}$, ρ_1, and d_0, and denote by G' the output decoding graph. The transformation taking the initial input φ into G' (via intermediate steps G and G_1) is equivalent, by Proposition 5.6.19, to a dPCP with the claimed parameters. ■

5.6.5 Proof of the result of [MR08], Theorem 5.1.2

Our Theorem 5.1.1 asserts the existence of a two query PCP with soundness error $1/\operatorname{poly}\log n$ and alphabet size $|\Sigma| = 2^{\operatorname{poly}\log n}$. In this section we will sketch a proof of Theorem 5.1.2 in which the alphabet size $|\Sigma|$ can be any value smaller than $2^{\operatorname{poly}\log n}$ while maintaining the relation of $\varepsilon \leq 1/\operatorname{poly}(\log|\Sigma|)$.

Theorem (5.1.2, restated, [MR08]**).** *For any function $\varepsilon(n) \geq 1/\operatorname{poly}\log n$ the class* **NP** *has a two-query PCP verifier with perfect completeness, soundness error at most ε over alphabet Σ of size at most $|\Sigma| \leq 2^{1/\operatorname{poly}(\varepsilon)}$.*

Our proof of Theorem 5.1.2 relies on the scheme of [DH09] who showed a generic way to compose a PCP with a dPCP, and then proved Theorem 5.1.2 by repeating the composition step, assuming the existence of two building blocks: a PCP and a dPCP. We plug in our constructions of a PCP (Theorem 5.1.1) and of a dPCP (Theorem 5.1.6) into the composition scheme of [DH09] and obtain a new construction of the verifier of Theorem 5.1.2 that does not rely on low degree polynomials.

Remark 5.6.27. An important feature of the theorem of [MR08] asserts that the verifier is randomness-efficient, i.e. it uses only $(1+o(1))\log n$ random bits rather than $O(\log n)$ random bits. This is equivalent to constructing constraint graphs of almost-linear size rather than polynomial size (see Remark 5.2.15). Using the composition scheme of [DH09], the outcome will be randomness efficient as long as the PCP verifier at the outermost level of composition is randomness-efficient. It does not, for example, depend on whether the dPCP is randomness-efficient.

However, since our PCP verifier from Theorem 5.1.1 is not randomness-efficient, we can only get this additional feature by relying at the outermost level on a PCP verifier as in [MR08]. The dPCP can still be based on our Theorem 5.1.6. Alternatively, if we also base the outermost PCP on theorem 5.1.1 we get a polynomial-size construction, but not a "randomness-efficient" one. It is also conceivable that the construction of Theorem 5.1.1 can be improved to yield a randomness-efficient PCP, and we leave this for future work.

In order to state the generic composition theorem of [DH09] let us first define the *decision complexity* of a PCP verifier. Roughly speaking, a PCP verifier has decision complexity $s(n)$ if every constraint in the underlying constraint graph can be computed by a circuit of size at most $s(n)$[6]. This definition is analogous to the definition of the decoding complexity of a PCP decoder. It is easy to see that the PCP verifier (from Theorem 5.1.1) has decision complexity $\operatorname{poly}\log n$ in the same way that the dPCP decoder (from Theorem 5.1.6) was shown to have decoding complexity $\operatorname{poly}\log n$.

We turn to state the composition theorem of [DH09]. As in all composition theorems in the literature, the goal of this theorem is to take an "outer verifier" (in this case, a PCP verifier), which has a large alphabet, and reduce its alphabet size by composing it with an "inner verifier" (in this case, a PCP decoder). The gain is obtained from the fact that the inner verifier is invoked on a claim of size $s(n) \ll n$, and thus can have a much smaller alphabet than the outer verifier. The result of the composition is a verifier that has the alphabet size roughly as of the inner verifier, and can still be invoked on a claim of size n. However, the composed verifier accumulates soundness error from the invocations of both the outer verifier and the inner verifier, and thus the composition does not come "for free".

Theorem 5.6.28 (Paraphrasing [DH09])**.** *Let V and D be a PCP verifier and a PCP decoder as follows:*

1. *Let V be a two-query PCP verifier for* **NP** *with perfect completeness, soundness error $\Delta(n)$, alphabet size $|\Sigma(n)|$, and decision complexity $s(n)$. Assume further that the PCP verifier makes projection queries.*

2. *Let D be a two-query PCP decoder for* CircuitSat$_\Gamma$ *for some $\Gamma(n)$. Assume D has perfect completeness, soundness error $\delta(n)$, list size $L(n)$, and alphabet size $|\sigma(n)|$.*

If both V and D have the projection property then there is a PCP verifier $V \circledast D$ with the following properties. $V \circledast D$ invokes D on inputs of length at most $s(n)$. $V \circledast D$ has perfect completeness, soundness error $O(\delta(s(n)) + L(s(n))\Delta(n))$, alphabet size $|\sigma(s(n))|^{\operatorname{poly}(L(s(n))/\delta(s(n)))}$, and $V \circledast D$ has the projection property.

As discussed above, the main gain from this theorem is that the alphabet size of $V \circledast D$ is much smaller than that of V. Let us see how this is useful. Suppose we take V, D from Theorems 5.1.1 and 5.1.6. We have $\Sigma(n) \le 2^{\operatorname{poly}\log n}$, $s(n) = \operatorname{poly}\log n$, and $\sigma(n) \le 2^{\operatorname{poly}\log n}$. Thus, $\sigma(s(n)) = 2^{\operatorname{poly}\log\log(n)}$. Similarly $L(s(n)) \le \operatorname{poly}\log\log n$ and $\delta(s(n)) = 1/\operatorname{poly}\log\log n$. This results in alphabet size of $2^{\operatorname{poly}\log\log(n)}$ and soundness error of $1/\operatorname{poly}\log\log n$. By composing this verifier again with D (yielding $(V \circledast D) \circledast D$) one can inductively obtain a PCP verifier with soundness error $1/\operatorname{poly}\log^{(i)} n$ for any i and corresponding alphabet size $|\Sigma| = 2^{1/\operatorname{poly}(\epsilon)}$. To get *any* alphabet size $|\Sigma|$ one must do careful padding and we do not go into these details.

[6]More precisely, the verifier should be able to compute this circuit based on its input and its randomness.

The composition theorem (Theorem 5.6.28) is stated here in the two-query terminology (rather than in the terminology of "robust" PCPs). Let us now give a brief outline of how to obtain this version from the version of [DH09]:

1. *From two-query to robust:* Use Lemma 2.5 of [DH09] to deduce existence of a robust PCP rV and a robust dPCP rD with parameters related to V and D. In particular, the number of accepting views for rD is bounded by $|\sigma|$.

2. *Composition:* Apply Theorem 4.2 of [DH09] with parameter $\varepsilon = \delta/L \geq |\sigma|^{\Omega(1)}$. Deduce a new robust PCP $rV \circledast rD$ with parameters as follows. The soundness error is $\delta + L\Delta + 4L\varepsilon = O(\delta + L\Delta)$. The number of accepting views is at most $|\sigma|^{4/\varepsilon^4}$ (this follows from inspecting the proof, but not directly from the theorem statement).

3. *Back to two queries:* Again use Lemma 2.5 to move back to a two query PCP. The new alphabet size is at most the number of accepting views of $rV \circledast rD$ which is at most $|\sigma(s(n))|^{4/\varepsilon^4} = |\sigma|^{(L/\delta)^{O(1)}}$ as claimed. ■

5.7 Decoding PCPs with Linear Structure

In this section we prove Lemma 5.6.25, i.e., that every decoding graph G can be embedded on a graph that has linear structure. The heart of the proof is very similar to the proof of the corresponding lemma for constraint graphs (Lemma 5.3.3) with few adaptations to the setting of decoding graphs. Two important differences are the following:

1. Recall that we prove Lemma 5.3.3 by embedding the constraint graph G on a de Bruijn graph $\mathcal{DB}$, and that this is done by identifying the vertices of G with the vertices of $\mathcal{DB}$. Furthermore, recall that if $\mathcal{DB}$ has more vertices than G, then some of the vertices of $\mathcal{DB}$ are not identified with vertices of G, and thus we place only trivial constraints on those vertices.
   This construction does not work for decoding graphs. The reason is that in the setting of decoding graphs every edge needs to be able to decode some index $k \in [t]$. Furthermore, every edge that fails to decode must contribute to the fraction of rejecting edges. Thus, we can not have many trivial edges.
   In order to resolve this issue, we prove a proposition that allows us to ensure that G has exactly the same number of vertices as in $\mathcal{DB}$, see Proposition 5.7.4 below.
   We note that Item 1 is *not* caused by the fact we chose a strong definition of udPCP and not a weak one (see Remark 5.6.11). Even if we used a weak definition of udPCP, requiring edges to reject only if the decoding error is above some threshold, we still could not use dummy vertices and edges in the embedding, as this would cause the aforementioned threshold to be too large for our purposes.

2. Recall that in the embedding of constraint graphs on de Bruijn graphs we used the expander-replacement technique (Lemma 5.4.8) to make sure that the graph G has small degree. Since such a lemma was not proved for decoding graphs in previous works, we have to prove it on our own. This is done in Proposition 5.7.3 below.

The rest of this section is organized as follows. In Section 5.7.1 we prove the aforementioned Propositions 5.7.3 and 5.7.4. Then, in Section 5.7.2, we prove Lemma 5.6.25.

5.7.1 Auxiliary propositions

In this section we prove Propositions 5.7.3 and 5.7.4 mentioned above. In order to state those two propositions, we need to define a special kind of decoding graphs, called "vertex-decoding graphs". The reason is that we only know how to prove Proposition 5.7.4 for vertex-decoding graphs. Fortunately, we can convert any decoding graph to a vertex-decoding one using Proposition 5.7.3.

We move to define the notion of vertex-decoding graphs. Intuitively, a decoding graph is vertex-decoding if the value that an edge (u, v) decodes depends only on the labeling of u, while the labeling of v only affects on whether the edge accepts or rejects. The formal definition follows.

Definition 5.7.1 (Vertex-decoding graphs)**.** We say that a decoding graph G is a *vertex-decoding graph* if it has the following properties:

1. For every edge (u, v) of G and its associated circuit $\psi = \psi_{(u,v)}$, there exists a function $f : \Sigma \to \Gamma$ that satisfies the following: For every assignment π to the vertices of G for which $\psi(\pi(u), \pi(v)) \neq \perp$ it holds that $\psi(\pi(u), \pi(v)) = f(\pi(u))$.

2. Every vertex has at least one outgoing edge. In other words, every vertex is capable of decoding at least one index $k \in [t]$.

Remark 5.7.2. While the property of a graph being vertex-decoding is reminiscent of the projection property, there are two important differences. First, note that Item 1 in Definition 5.7.1 is weaker than the projection property, since it only requires that $\pi(u)$ determines the decoded value, and not necessarily $\pi(v)$. Second, note that Item 2 is not required by the projection property, and is actually violated by the known constructions of graphs that have the projection property.

We turn to prove Propositions 5.7.3 and 5.7.4. We begin with Proposition 5.7.3, which says that we can always reduce the degree of decoding graphs while paying only a moderate cost in the parameters. As mentioned above, the proposition also transforms the decoding graph into a vertex-decoding graph.

Proposition 5.7.3. *Let d_0 be the constant from Fact 5.2.20, and let $d = 2d_0$. There exists a polynomial time procedure that acts as follows:*

- ***Input**: A decoding graph G of size n for input circuit $\varphi : \Gamma^t \to \{0, 1\}$ with alphabet Σ, rejection ratio ρ, decoding complexity s, and smoothness γ.*

- ***Output:** A d-regular vertex-decoding graph G' of size at most $d \cdot n/\gamma$ for input circuit φ, alphabet Σ^2, rejection ratio $\Omega(\rho)$, decoding complexity $s + \text{poly} \log |\Sigma|$, and smoothness 1. Furthermore, G' has at most n/γ vertices.*

Proof sketch We apply the same construction as in the proof of Proposition 5.6.24. Let $\varphi : \Gamma^t \to \{0, 1\}$ be the input circuit of G. The key observation is that G corresponds to a decoder D that acts on φ such that D has query complexity 2, randomness complexity $\log(n/t \cdot \gamma)$, proof alphabet Σ, rejection ratio ρ, and decoding complexity s. The reason for the foregoing randomness complexity is that by the smoothness of G and by the smoothness criterion of Proposition 5.6.22, it holds that for every $k \in [t]$ there are at most $n/t \cdot \gamma$ edges that are associated with k, and therefore choosing a uniformly distributed edge that is associated with G requires $\log(n/(t \cdot \gamma))$ uniformly distributed bits. Now, by applying the construction of the proof of Proposition 5.6.24 to the decoder D, we obtain a graph G' that satisfies the requirements. The fact that G' is vertex-decoding can be observed by examining the construction of Proposition 5.6.24 (see also the second paragraph in the above proof sketch of Proposition 5.6.24). ■

We next prove Proposition 5.7.4, which says that we can increases the number of vertices of a vertex-decoding graph to any size we wish, while paying only a small cost in the parameters. This proposition will be used to ensure that the number of vertices of a decoding graph G is equal to the number of vertices of the de Bruijn graph on which we want to embed G.

Proposition 5.7.4. *There exists a polynomial time procedure that acts as follows:*

- ***Input:***
  - *A vertex-decoding graph G of size n for input circuit $\varphi : \Gamma^t \to \{0,1\}$ with ℓ vertices, alphabet Σ, rejection ratio ρ, decoding complexity s, degree bound d, and smoothness γ.*
  - *A number $\ell' \in \mathbb{N}$ such that $\ell' \geq \ell$ (given in unary).*
- ***Output:*** *Let $c \stackrel{\text{def}}{=} \left\lfloor \frac{\ell'}{\ell} \right\rfloor$ and let d_0 and h_0 be the constants from Fact 5.2.20. The procedure outputs a vertex-decoding graph G' of size at most $2 \cdot (c+1) \cdot d_0 \cdot n$ for input circuit φ that has exactly ℓ' vertices and also has alphabet Σ, output size $s + \operatorname{poly}\log|\Sigma|$, rejection ratio $\Omega\left(\gamma^2 \cdot \rho/d^2\right)$, degree bound $2 \cdot d_0 \cdot d$, and smoothness $\frac{1}{2} \cdot \gamma$.*

Furthermore, if G is d-regular then G' is $(2 \cdot d_0 \cdot d)$-regular and has rejection ratio $\Omega\left(\gamma^2 \cdot \rho\right)$.

Proof sketch The basic idea of the proof is as follows. Given the graph G, we construct the graph G' by replacing each vertex v of G with multiple copies of v, such that the total number of vertices becomes ℓ' as required. Each copy of v will be connected to the same edges as the original v. An assignment to G' will be required to assign the same value to all the copies of v: Clearly, if an assignment π' to G' assigns the same value to the copies of each vertex v of G, then in a way π' "behaves" like an assignment to G, and we can use the soundness of G to establish the soundness of G' with respect to π'. In order to verify that the copies of a vertex v are assigned the same value, we will put equality constraints between the copies of v. In order to save edges, the equality constraints are placed according to the edges of an expander, and the analysis goes exactly as in the proof of Proposition 5.6.24. We use the fact that G is vertex decoding in order to allow the equality constraints to decode values even though they can use only the labeling of a single vertex of G. The rest of this proof consists of the technical details of this construction, and can be found in [DM10, App. E]. ■

5.7.2 Embedding decoding graphs on de Bruijn graphs

In this section we prove the following proposition, which implies Lemma 5.6.25 (linear structure embedding for udPCPs) and is analogous to Proposition 5.4.4 (embedding of constraint graphs on de-Bruijn graphs). The proof follows the steps of Proposition 5.4.4 with the few adaptations to the setting of decoding graphs. For intuition and a high-level explanation of the proof, we refer the reader to Section 5.4 and in particular to Section 5.4.2.

Proposition 5.7.5 (Embedding Decoding Graphs on de-Bruijn Graphs)**.** *Let d_0 be the constant of Fact 5.2.20. There exists a polynomial time procedure that satisfies the following requirements:*

- ***Input:***
  - *A decoding graph G of size n for an input circuit $\varphi : \Gamma^t \to \{0,1\}$ with alphabet Σ, rejection ratio ρ, decoding complexity s, and smoothness γ.*
  - *A finite alphabet Λ such that $|\Lambda| \geq 4 \cdot d_0^2$.*
  - *A natural number m such that $|\Lambda|^m \geq 2 \cdot d_0 \cdot n/\gamma$.*
- ***Output:*** *A decoding graph G' for φ such that the following holds:*

- *The underlying graph of G' is the de Bruijn graph $\mathcal{DB}_{\Lambda,m}$.*
- *The size of G' is $|\Lambda|^{m+1}$.*
- *G' has alphabet $\Sigma^{O(m)}$.*
- *G' has rejection ratio $\Omega\left(\rho/|\Lambda|^2 \cdot m\right)$.*
- *G' has smoothness at least $\gamma' \stackrel{\text{def}}{=} \Omega\left(\frac{1}{|\Lambda|}\right)$.*
- *G' has decision complexity $s + \text{poly}(m, \log|\Sigma|)$*

Let G, Λ, and m be as in Proposition 5.7.5, and let $\varphi : \Gamma^t \to \{0,1\}$ be the input circuit of G. On input G, Λ, and m, the procedure acts as follows. The procedure first constructs a vertex-decoding graph G_1 by applying to G the procedure of Proposition 5.7.3, and then applying to the resulting graph the procedure of Proposition 5.7.4 with $\ell' = |\Lambda|^m$. It can be verified that G_1 is a vertex-decoding graph for input circuit φ with exactly $|\Lambda|^m$ vertices, alphabet $\Sigma_1 \stackrel{\text{def}}{=} \Sigma^2$, rejection ratio $\rho_1 = \Omega(\rho)$, decoding complexity $s + \text{poly}\log|\Sigma|$, and smoothness at least $\frac{1}{2}$. Furthermore, G_1 is d-regular for $d = 4 \cdot d_0^2 \leq |\Lambda|$, and is of size $d \cdot |\Lambda|^m$.

Then, the procedure identifies the vertices of G_1 with the vertices of $\mathcal{DB} = \mathcal{DB}_{\Lambda,m}$, partitions the the edges of G_1 to d matchings $\mu_1, \ldots, \mu_d$, and views those matchings as permutations on the vertices of $\mathcal{DB}$. We apply Fact 5.4.5 to each permutation μ_i resulting in a set of paths $\mathcal{P}_i$ of length $l \stackrel{\text{def}}{=} 2m$. Let $\mathcal{P} = \bigcup \mathcal{P}_i$.

Next, the procedure constructs G' in the following way. The alphabet of G' is set to be $\Sigma_1^{l \cdot d}$, viewed as $\left(\Sigma_1^l\right)^d$. If $\sigma \in \left(\Sigma_1^l\right)^d$, and $\sigma = (\sigma_1, \ldots, \sigma_d)$, we denote by $\sigma_{i,j}$ the element $(\sigma_i)_j \in \Sigma_1$. It remains to describe how to associate each edge e of G' with an index $k_e \in [k]$ and with a circuit ψ_e. To this end, we first describe in which cases a circuit ψ_e accepts, and then describe how the index k_e is chosen and what is the output of ψ_e when it accepts.

The conditions in which ψ_e accepts. Fix an edge $e' = (u,v)$ of G', and let ψ_e be the circuit associated with e. The circuit ψ_e accepts in exactly the same cases in which the constraint that corresponds to e in the proof of Proposition 5.4.4 (for constraint graphs) accepts. That is, the circuit ψ_e accepts if and only if all of the following conditions hold:

1. For every $i \in [d]$, the values $\left(\pi'(u)_{i,l}, \pi'(u)_{i,1}\right)$ satisfy the edge $\left(\mu_i^{-1}(u), u\right)$ of G.
2. It holds that $\pi'(u)_{1,1} = \ldots = \pi'(u)_{d,1}$ and that $\pi'(v)_{1,1} = \ldots = \pi'(v)_{d,1}$.
3. For every $i \in [d]$ and $j \in [l-1]$ such that u and v are the j-th and $(j+1)$-th vertices of a path in $p \in \mathcal{P}_i$ respectively, it holds that $\pi'(u)_{i,j} \neq \pi'(v)_{i,j+1}$.
4. Same as Condition 3, but when v is the j-th vertex of p and u is its $(j+1)$-th vertex.

The choice of k_e and the output of ψ_e. Fix a vertex u of G'. We describe the way we assign indices k_e to the outgoing edges of u, and the output of the circuits ψ_e. We begin by associating each of the $|\Lambda|$ outgoing edges of u in G' with one of the d outgoing edges of u in G_1. This association is done in a "balanced" way - that is, each outgoing edge of u in G_1 is associated with either $\lfloor |\Lambda|/d \rfloor$ or $\lceil |\Lambda|/d \rceil$ edges of u in G'.

Now, let e' be an outgoing edge of u in G', and suppose that it is associated with an outgoing edge e_1 of u in G_1, and that e_1 belongs to the matching μ_i. Let k_{e_1} and ψ_{e_1} be the index and circuit associated with e_1. Recall that since G_1 is vertex-decoding, there exists a function $f_{e_1} : \Sigma_1 \to \Gamma$ such that whenever

$\psi_{e_1}(a,b) \neq \perp$ it holds that $\psi_{e_1}(a,b) = f_{e_1}(a)$. We associate e' with the index k_{e_1}, and with the circuit $\psi_{e'}$ that is defined for every $a', b' \in \left(\Sigma_1^l\right)^d$ for which $\psi_{e'}(a,b) \neq \perp$ by

$$\psi_{e'}(a', b') = f_{e_1}\left((a')_{1,1}\right).$$

Note that $\psi_{e'}$ is indeed well defined, since the cases in which $\psi_{e'}$ outputs $\perp$ were defined above.

The parameters of G'. The size and alphabet of G' are immediate, and the completeness of G' can be established in the same way as in Proposition 5.4.4 (embedding of constraint graphs on de-Bruijn graphs). It can also be verified that G' has smoothness at least $\gamma' = \frac{1}{2\cdot|\Lambda|}$ using the smoothness criterion (Proposition 5.6.22) and a straightforward calculation.

It remains to analyze the rejection ratio of G'. Let π' be an assignment to G' that minimizes the ratio between the probability that a random edge of G' rejects π' (under the decoding distribution) to the decoding error of G' on π'. As in the proof of Proposition 5.4.4, we may assume that for every vertex u of $\mathcal{DB}$ it holds that $\pi'(u)_{1,1} = \ldots = \pi'(u)_{d,1}$, since otherwise we may modify π' to such an assignment that satisfies this property without increasing the rejection probability or decreasing the decoding error. Let π_1 be the assignment to G_1 defined by $\pi_1(u) = \pi'(u)_{1,1}$. Let ε be the decoding error of G_1 on π_1, and let x be the assignment to φ that achieves this decoding error. Let ε' be the decoding error of G' on π' with respect to x. We show that the rejection probability of G' on π' is at least $\Omega\left(\gamma' \cdot \rho_1 \cdot \varepsilon' / |\Lambda| \cdot m\right)$, and this will yield the required rejection ratio.

Observe that by the smoothness of G_1 (resp. G'), the fraction of edges of G_1 (resp. G') that fail to decode x on π_1 (resp. π') is at least $\varepsilon_0 \overset{\text{def}}{=} \frac{1}{2} \cdot \varepsilon$ (resp. $\varepsilon_0' = \gamma' \cdot \varepsilon'$). Furthermore, the fraction of edges of G_1 that reject π_1 is at least $\rho_1 \cdot \varepsilon_0$. This implies, using the same argument as in the proof of Proposition 5.4.4, that the fraction of edges of G' that reject π' is at least $\Omega\left(\rho_1 \cdot \varepsilon_0 / |\Lambda| \cdot m\right)$.

We finish the proof by relating ε_0' with ε_0. To this end, observe that for every edge $e' = (u, v)$ of G' and its associated edge e_1 of G_1, the edge e' fails to decode x on π' (i.e. $\psi_{e'}(\pi'(u)) \notin \left\{x_{k_{e'}}, \perp\right\}$) only if e_1 fails to decode x on π_1 (i.e. $\psi_{e_1}(\pi_1(u)) \notin \left\{x_{k_{e_1}}, \perp\right\}$). Furthermore, each edge e_1 of G_1 corresponds to either $\lfloor|\Lambda|/d\rfloor$ or $\lceil|\Lambda|/d\rceil$ edges in G'. It can be verified by a straightforward calculation that this implies that $\varepsilon_0' \leq 2 \cdot \varepsilon_0$. It now follows that the fraction of edges of G' that reject π' is at least

$$\begin{aligned}
\Omega\left(\frac{\rho_1 \cdot \varepsilon_0}{|\Lambda| \cdot m}\right) &\geq \Omega\left(\frac{\rho_1 \cdot \varepsilon_0'}{|\Lambda| \cdot m}\right) \\
&\geq \Omega\left(\frac{\rho_1 \cdot \gamma'}{|\Lambda| \cdot m} \cdot \varepsilon'\right) \\
&= \Omega\left(\frac{\rho}{|\Lambda|^2 \cdot m} \cdot \varepsilon'\right).
\end{aligned}$$

The required rejection ratio follows. ■

5.8 Derandomized Parallel Repetition of Decoding Graphs with Linear Structure

In this section we prove Lemma 5.6.26 (derandomized parallel repetition for dPCPs), restated below.

Lemma (5.6.26, restated)**.** *There exist a universal constant h and a polynomial time procedure that satisfy the following requirements:*

- ***Input:***

  - *A finite field $\mathbb{F}$ of size q.*
  - *A decoding graph $G = (\mathbb{F}^m, E)$ of size n for input circuit $\varphi : \Gamma^t \to \{0,1\}$ with linear structure, alphabet Σ, rejection ratio ρ, decision complexity s, and smoothness γ.*
  - *The rejection ratio ρ of G.*
  - *A parameter $d_0 \in \mathbb{N}$ such that $d_0 < m/h^2$ and $\rho \geq h \cdot d_0 \cdot q^{-d_0/h}/\gamma$.*

- ***Output:*** *A decoding graph G' for φ such that the following holds:*
  - *G' has size $n^{O(d_0)}$.*
  - *G' has alphabet $\Sigma^{q^{O(d_0)}}$.*
  - *G' is list-decoding with soundness error $\varepsilon \overset{\text{def}}{=} h \cdot d_0 \cdot q^{-d_0/h}/\gamma$ and list size $L \overset{\text{def}}{=} q^{O(d_0)}$.*
  - *G' has the projection property.*
  - *G' has decoding complexity $q^{O(d_0)} \cdot (s + \operatorname{poly}\log|\Sigma|)$.*

The proof follows the proof of the corresponding lemma for constraint graphs (Lemma 5.3.4), with the following modification: Recall that the proof of Lemma 5.3.4 described the graph G' by describing a *verification* procedure (the E-test, Figure 5.2). Moreover, recall that the E-test works by choosing a random subspace F of edges and verifying that the edges in F are satisfied by the assignment $\Pi(F)$.

In order to describe the graph G' of Lemma 5.6.26, we describe a *decoding* procedure (the E-decoder, see Figure 5.4 below). The E-decoder is constructed by changing the E-test as follows. Whenever the E-decoder is required to decode an index $k \in [t]$, the E-decoder chooses a random edge e that is associated with k, and then chooses the subspace F to be a random subspace that contains e. The E-decoder then checks, as before, that the edges in F are satisfied by the assignment $\Pi(F)$. If one of the edges in F is unsatisfied, then the E-decoder rejects. If all the edges in F are satisfied, then the E-decoder decodes the index k by invoking the circuit ψ_e associated with e on input $\Pi(F)_{|e}$.

The intuition that underlies the construction of the E-decoder is as follows. Just as in the proof of Lemma 5.3.4, we argue that the E-decoder contains an implicit S-test, and therefore the assignment Π needs to be roughly consistent with some assignment π to G in order to be accepted. We now consider two cases:

1. If G has high decoding error on π, then by the soundness of G it holds that many of the edges of G reject π. By the sampling property of F, there are many edges in F that reject π, and therefore the E-decoder must reject with high probability.

2. If G has low decoding error on π, then due to the sampling property of F, only few of the edges in F err. In particular, since e is distributed like a random edge of F, it only errs with low probability. Thus, in this case the E-decoder decodes correctly with high probability.

Thus, in both cases the soundness error of the E-decoder is small.

5.8.1 The construction of G' and its parameters

The decoding graph G' is constructed as follows. Let $G = (\mathbb{F}^m, E)$ and d_0 be as in Lemma 5.6.26 (derandomized parallel repetition for dPCPs), and let $d_1 = h \cdot d_0$ where h is the universal constant from Lemma 5.6.26 to be chosen later. As in the proof of the corresponding lemma for constraint graphs (Lemma 5.3.4), the graph G' is bipartite, the right vertices of G' are the $2d_0$-subspaces of $\mathbb{F}^m$ (the vertex-space of G), and the left vertices of G' are the $2d_1$-subspaces of the edge space E of G. An assignment Π to G' should label each $2d_0$-subspace A of $\mathbb{F}^m$ with a function from A to Σ, and each $2d_1$-subspace F

1. Suppose that we are required to decode an index $k \in [t]$. Let $e = (u, v)$ be a uniformly distributed edge of G that is associated with k, and let ψ_e be its associated circuit.
2. Let F_L and F_R to be random d_1-subspaces of E, and let
$$B_L \stackrel{\text{def}}{=} \text{left}\,(F_L)\,, \quad B_R \stackrel{\text{def}}{=} \text{right}\,(F_R)\,, \quad F \stackrel{\text{def}}{=} F_L + F_R.$$
F_L and F_R are chosen to be uniformly and independently distributed d_1-subspaces of E conditioned on $e \in F$, $\dim(F) = 2d_1$, $\dim(B_L) = d_1$, $\dim(B_R) = d_1$, and $B_L \cap B_R = \{0\}$.
3. Let A_L and A_R be uniformly distributed d_0-subspaces of B_L and B_R respectively, and let
$$A \stackrel{\text{def}}{=} A_L + A_R.$$
4. If either $\Pi\,(F)_{|(A_L,A_R)} \neq \Pi\,(A)_{|(A_L,A_R)}$ or the assignment $\Pi\,(F)$ is rejected by of the edges in F, output $\perp$.
5. Otherwise, output ,$\psi_e\left(\Pi\,(F)_{|u}\,, \Pi\,(F)_{|v}\right)$.

Figure 5.4: The E-decoder

of E with a function that maps the endpoints of the edges in F to Σ. The edges of G' are constructed such that they simulate the action of the "E-decoder" described in Figure 5.4.

The completeness, size, and alphabet size of G' is can be verified in the same way as it was done in the proof of Lemma 5.3.4, and so is the fact that G' has the projection property. It remains to analyze the soundness of G', which is done in the following section.

5.8.2 The soundness of G'

We turn to prove that G' is list-decoding with $\varepsilon = h \cdot d_0 \cdot q^{-d_0/h}/\gamma$ and list size $L = q^{O(d_0)}$. Let Π be an assignment to G'. That is, we prove that there exists a (possible empty) list of satisfying assignments $x^1, \ldots, x^L \in \Gamma^t$ to the input circuit φ such that when given as input a uniformly distributed index $k \in [t]$, the probability that the output of the E-decoder is not in $\left\{x_k^1, \ldots, x_k^L, \perp\right\}$ is at most ε.

Consider the distribution on the edges of G' that results from letting the edge e of the E-decoder be chosen according to *the uniform distribution on the edges of* G instead of the decoding distribution of G. We will refer to the above distribution as the *G-uniform distribution of G'*. It is straightforward to show that the G-uniform distribution and decoding distribution of G' are γ-similar, by applying Claim 5.2.18 with X_1 and X_2 being the choices of e according the the G-uniform distribution and the decoding distribution, and Y_1 and Y_2 being the G-uniform distribution and decoding distribution of G' respectively. In the following proof, all the probability expressions are *not over the decoding distribution* of G', but rather over the *G-uniform distribution of G'*. We will later use the similarity between the distributions to argue that G' has small soundness error with respect to its decoding distribution.

Notation 5.8.1. We denote by $\mathcal{D}$ the random variable that equals to the output of the E-decoder. As in the proof of Lemma 5.3.4 (derandomized parallel repetition for constraint graphs), we denote by $\mathcal{T}$ the event in which the E-decoder accepts Π, so $\mathcal{T}$ is the event $\mathcal{D} \neq \perp$. Moreover, as in the proof of Lemma 5.3.4, for an assignment $\pi : \mathbb{F}^m \to \Sigma$, we denote by $\Pi\,(F) \overset{\alpha}{\approx} \pi$ the claim that for at least

$1-\alpha$ fraction of the edges e of F it holds that $\Pi(F)$ is consistent with π on both the endpoints of e, and otherwise we denote $\Pi(F) \overset{\alpha}{\not\approx} \pi$.

Our proof proceeds in two steps. We first show that there exists a (possible empty) assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that whenever the E-decoder accepts Π, it almost always does so while being roughly consistent with one of the assignments $\pi^1, \ldots, \pi^L$. We can then choose the assignments $x^1, \ldots, x^L$ to be the assignments that minimize the decoding error of $\pi^1, \ldots, \pi^L$ respectively. Next, we show that whenever Π is roughly consistent with π^i, the E-decoder either rejects Π with high probability (if π^i has high decoding error) or decodes x^i successfully with high probability (if π^i has low decoding error). Thus, the overall probability that the E-decoder fails is small.

The above strategy is made formal in the following three propositions. Let h' and c be the universal constants defined in Theorem 5.8.5 below, and let $\alpha \overset{\text{def}}{=} h' \cdot d_0 \cdot q^{-d_0/h'}$. Let $\varepsilon_0 \overset{\text{def}}{=} \varepsilon \cdot \gamma/3 = h \cdot d_0 \cdot q^{-d_0/h}/3$ and let $L = O\left(1/\varepsilon_0^c\right)$.

Proposition 5.8.2. *There exists a (possibly empty) list of assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that*

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] < 2 \cdot \varepsilon_0.$$

Proposition 5.8.3. *For every assignment $\pi : \mathbb{F}^m \to \Sigma$ on which G has decoding error at least $\varepsilon_0/2L$ it holds that* $\Pr\left[\mathcal{T} \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi\right] < \varepsilon_0/L$.

Proposition 5.8.4. *For every assignment $\pi : \mathbb{F}^m \to \Sigma$ on which G has decoding error less than $\varepsilon_0/2L$ with respect to a satisfying assignment x to the input circuit φ it holds that*

$$\Pr\left[\mathcal{D} \neq x_k \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi\right] < \varepsilon_0/L,$$

where k is the index on which the E-decoder is invoked.

Propositions 5.8.2 and 5.8.4 are proved in Sections 5.8.2.1 and 5.8.2.2 respectively. Proposition 5.8.3 can be proved in the same way as Proposition 5.5.7, by noting that due to the soundness of G, at least $\rho \cdot \varepsilon_0/2L$ of the edges of G reject π.

We now prove that G' is (L, ε)-list decoding using Propositions 5.8.2, 5.8.3, and 5.8.4. Let $\pi^1, \ldots, \pi^L$ be the assignments from Proposition 5.8.2. For each $i \in [L]$, let x^i be the assignment to φ that attains the decoding error of π^i. The decoding error of G' on Π under the G-uniform distribution of G' is as follows.

$$\begin{aligned}
\Pr\left[\mathcal{D} \notin \left\{x_k^1, \ldots, x_k^L, \perp\right\}\right] &\leq \sum_{i=1}^{L} \Pr\left[\mathcal{D} \notin \left\{x_k^1, \ldots, x_k^L, \perp\right\} \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \\
&\quad + \Pr\left[\mathcal{D} \notin \left\{x_k^1, \ldots, x_k^L, \perp\right\} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \\
&\leq \sum_{i=1}^{L} \Pr\left[\mathcal{D} \notin \left\{x_k^i, \perp\right\} \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \\
&\quad + \Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \\
&\leq \sum_{i=1}^{L} \varepsilon_0/L + 2 \cdot \varepsilon_0 \qquad (5.7) \\
&= 3 \cdot \varepsilon_0,
\end{aligned}$$

where Inequality 5.7 follows from Propositions 5.8.2 and 5.8.4. Finally, since the G-uniform distribution of G' and the decoding distribution of G' are γ-similar, it follows that the decoding error of G' on Π under the decoding distribution of G' is at most $3 \cdot \varepsilon_0/\gamma = \varepsilon$, as required. ■

5.8.2.1 Proof of Proposition 5.8.2

Recall that in order to analyze the soundness of the E-test in Proposition 5.5.6, we argued that the E-test contains an "implicit S-test", and then relied on a theorem regarding the soundness of the S-test (Theorem 5.5.4). The aforementioned theorem said that if the S-test accepts an assignment Π with some probability, then there exists an assignment π such that with some (smaller) probability, the S-test accepts Π while being consistent with the S-direct product of π. This can be thought as a "unique decoding" theorem, that decodes π from Π.

In order to prove Proposition 5.8.2 for the E-decoder, we use a similar argument, but this time we use a "list decoding" theorem for the S-test. The following theorem says that there exists a short list of assignments $\pi_1, \ldots, \pi_L$, such that it is *almost always* the case that if the S-test accepts Π, it does so while being consistent with the S-direct product of one of the assignments $\pi_1, \ldots, \pi_L$.

Theorem 5.8.5 (List-decoding soundness of the S-test)**.** *There exist universal constants $h', c \in \mathbb{N}$ such that for every $d_0 \in \mathbb{N}$, $d_1 \geq h' \cdot d_0$, and $m \geq h' \cdot d_1$, the following holds: Let $\varepsilon \geq h' \cdot d_0 \cdot q^{-d_0/h'}$, $\alpha \stackrel{\text{def}}{=} h' \cdot d_0 \cdot q^{-d_0/h'}$. Let Π be a (possibly randomized) assignment to $2d_0$-subspaces of $\mathbb{F}^m$ and to pairs of d_1-subspaces of $\mathbb{F}^m$. Then, there exists a (possibly empty) list of $L = O\left(1/\varepsilon^c\right)$ assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that*

$$\Pr\left[\Pi\left(B_1, B_2\right)_{|(A_1,A_2)} = \Pi\left(A\right)_{|(A_1,A_2)} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi\left(B_1, B_2\right) \stackrel{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right] < \varepsilon.$$

Theorem 5.8.5 is proved in Section 5.9.

We turn to prove Proposition 5.8.2 based on Theorem 5.8.5. As in the proof of Proposition 5.5.6, we begin by extending Π to pairs of independent d_1-subspaces of $\mathbb{F}^m$ in a randomized manner as follows: Given a pair of independent d_1-subspaces B_1 and B_2, we choose F_1 and F_2 to be uniformly distributed and independent d_1-subspaces of E such that $\text{left}\left(F_1\right) = B_1$ and $\text{right}\left(F_2\right) = B_2$, and set $\Pi\left(B_1, B_2\right) = \Pi\left(F_1 + F_2\right)_{|(B_1,B_2)}$.

Again as in the proof of Proposition 5.5.6, we observe that the probability that the E-decoder accepts equals to the probability that the S-test accepts the extended Π. The reason is that the subspaces B_L, B_R, A_L, A_R of the E-decoder are distributed like the subspaces B_1, B_2, A_1, A_2 of the S-test. By choosing h to be at least the constant h' we can invoke Theorem 5.8.5 (list-decoding soundness of the S-test), and conclude that there there exists a list of $L = O\left(1/\varepsilon^c\right)$ assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that for subspaces B_1, B_2, A_1, A_2 as in the S-test it holds that

$$\Pr\left[\Pi\left(B_1, B_2\right)_{|(A_1,A_2)} = \Pi\left(A\right)_{|(A_1,A_2)} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi\left(B_1, B_2\right) \stackrel{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right] < \varepsilon_0.$$

The latter inequality is equivalent to the following inequality:

$$\Pr\left[\Pi\left(F\right)_{|(B_L,B_R)} = \Pi\left(A\right)_{|(A_1,A_2)} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi\left(F\right)_{|(B_L,B_R)} \stackrel{\alpha}{\approx} \pi^i_{|(B_L,B_R)}\right] < \varepsilon_0,$$

which in turn implies the inequality

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi\left(F\right)_{|(B_L,B_R)} \stackrel{\alpha}{\approx} \pi^i_{|(B_L,B_R)}\right] < \varepsilon_0. \tag{5.8}$$

In the rest of this section we show that this implies that

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi\left(F\right) \stackrel{4\cdot\alpha}{\approx} \pi^i\right] < 2 \cdot \varepsilon_0 \tag{5.9}$$

To this end, we use Claim 5.5.9, which was proved in Section 5.5.3.1 and is restated below.

Claim (5.5.9, restated)**.** *For every fixed $2d_0$-subspace F_0 of E such that $\Pi(F_0) \overset{4\alpha}{\not\approx} \pi$, it holds that*

$$\Pr\left[\Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{|(B_L,B_R)} \middle| F = F_0\right] \leq 1/\left(q^{d_1-2}\cdot\alpha^2\right).$$

Claim 5.5.9 implies immediately the following corollary.

Corollary 5.8.6. *For every $i \in [L]$ it holds that*

$$\Pr\left[\Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi_{i|(B_L,B_R)} \middle| \nexists j \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^j\right] < 1/\left(q^{d_1-2}\cdot\alpha^2\right).$$

In order to prove Inequality 5.9, we first show that

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi^i_{|(B_L,B_R)} \middle| \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \geq \frac{1}{2}. \tag{5.10}$$

To show it, we prove an upper bound on the complement event, that is, we prove that

$$\Pr\left[\mathcal{T} \text{ and } \exists i \in [L] \text{ s.t. } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi^i_{|(B_L,B_R)} \middle| \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \leq \frac{1}{2}.$$

To see the latter inequality, observe that the right end side is upper bounded by

$$\begin{aligned}\sum_{i\in[L]} \Pr\left[\Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi^i_{|(B_L,B_R)} \middle| \nexists j \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^j\right] &\leq \sum_{i\in[L]} 1/\left(q^{d_1-2}\cdot\alpha^2\right) \\ &= L\cdot/\left(q^{d_1-2}\cdot\alpha^2\right) \\ &= O\left(1/\varepsilon_0^c\cdot\left(q^{d_1-2}\cdot\alpha^2\right)\right) \\ &\leq \frac{1}{2}.\end{aligned}$$

where the first inequality follows from Corollary 5.8.6, and the second inequality follows for sufficiently large choice of h. Now, it holds that

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi^i_{|(B_L,B_R)} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] \tag{5.11}$$

is upper bounded by

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F)_{|(B_L,B_R)} \overset{\alpha}{\approx} \pi^i_{|(B_L,B_R)}\right] < \varepsilon_0.$$

On the other hand, by writing the probability in (5.11) in conditional form and applying Inequality 5.10, we obtain that the probability in (5.11) is at least

$$\frac{1}{2}\cdot\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right].$$

By combining the two last bounds, we obtain that

$$\Pr\left[\mathcal{T} \text{ and } \nexists i \in [L] \text{ s.t. } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi^i\right] < 2\cdot\varepsilon_0,$$

as required. ■

5.8.2.2 Proof of Proposition 5.8.4

Fix an assignment $\pi : \mathbb{F}^m \to \Sigma$ on which G has decoding error less than $\varepsilon_0/2L$ with respect to a satisfying assignment x of the input circuit φ. We prove that $\Pr\left[D \neq x_k \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi\right] < \varepsilon_0/L$ Let us denote by $\mathcal{E}_1$ the event in which $\Pi(F) \overset{4\cdot\alpha}{\approx} \pi$ and by $\mathcal{E}_2$ the event in which F contains less than $\varepsilon_0/3L$ fraction of edges on which G fails to decode x on π. We will prove that

$$\Pr[\mathcal{D} \neq x_k \text{ and } \mathcal{E}_1] = \Pr\left[\mathcal{D} \neq x_k \text{ and } \Pi(F) \overset{4\cdot\alpha}{\approx} \pi\right] < \varepsilon_0/L.$$

It holds that

$$\Pr[\mathcal{D} \neq x_k \text{ and } \mathcal{E}_1] = \Pr[\mathcal{D} \neq x_k \text{ and } \mathcal{E}_1 \text{ and } \mathcal{E}_2] + \Pr[\psi(a,b) \neq x_k \text{ and } \mathcal{E}_1 \text{ and } \neg\mathcal{E}_2].$$

We upper bound both terms on the right hand side. The second term is clearly upper bounded by $\Pr[\neg\mathcal{E}_2]$. The latter probability can be shown to be at most $O\left(L^2/q^{2\cdot d_1-2}\cdot\varepsilon_0^2 + \cdot d_1/q^{m-2\cdot d_1}\right)$, using the fact that F samples well the edges of G, and more specifically using an argument similar to the one used in the proof of Proposition 5.5.7. For sufficiently large choice of h, the latter expression is upper bounded by $\varepsilon/3L$.

We turn to upper bound the probability $\Pr[\mathcal{D} \neq x_j \text{ and } \mathcal{E}_1 \text{ and } \mathcal{E}_2]$. This probability is upper bounded by the probability $\Pr[\mathcal{D} \neq x_j | \mathcal{E}_1 \text{ and } \mathcal{E}_2]$. Now, let F_0 be any $2d_1$-subspace of E such that $\Pi(F_0) \overset{4\cdot\alpha}{\approx} \pi_i$ and such that the fraction of edges of F_0 that fail to decode x on π is at most $2\varepsilon_0/3L$. Let us consider the probability $\Pr[\mathcal{D} \neq x_j | F = F_0]$. Observe that conditioned on the choice $F = F_0$, the edge e chosen by the E-test is uniformly distributed among the edges of F. Observe that e fails to decode x only if one of the endpoints of e is inconsistent with π or if e is one of the edges in F that fail to decode x on π. The probability of the first case is at most $4\cdot\alpha \leq \varepsilon_0/3L$ (where the latter inequality holds for sufficiently large choice of h), and the probability of the second case is at most $\varepsilon_0/3L$. It therefore holds that

$$\Pr[\mathcal{D} \neq x_k \text{ and } \mathcal{E}_1 \text{ and } \mathcal{E}_2] \leq \Pr[\mathcal{D} \neq x_j | F = F_0] \leq \varepsilon_0/3L + \varepsilon_0/3L \leq 2\varepsilon_0/3L.$$

All in all, it holds that $\Pr[\mathcal{D} \neq x_k \text{ and } \mathcal{E}_1]$ is at most $2\varepsilon_0/3L + 3\cdot\varepsilon_0/3L = \varepsilon_0/L$, as required. ■

5.9 The Analysis of the Specialized Direct Product Test

In this section we provide the analysis of the S-test and prove Theorems 5.5.4 and 5.8.5, which are the theorems on the soundness of the S-test that are used in Sections 5.5.3.1 and 5.8.2.1 respectively. The proof proceeds in two steps. First, in Section 5.9.1, we define and analyze an intermediate direct product test, which we call the P^2-test. Then, in Section 5.9.2, we reduce the analysis of the S-test to that of the P^2-test.

For the rest of this section, we let $\mathbb{F}$ be a finite field of size q and let $d_0, d_1 \in \mathbb{N}$.

5.9.1 The P^2-test

In this section we define and analyze the P^2-test. Informally, the P^2-test consists of two P-tests that are performed simultaneously. Details follow.

Given two strings $\pi_1, \pi_2 : \mathbb{F}^m \to \Sigma$, we define their *$P^2$-direct product* Π (with respect to $d_0, d_1 \in \mathbb{N}$) as follows: Π assigns each pair of d_0-subspaces (A_1, A_2) the pair of functions $(\pi_{1|A_1}, \pi_{2|A_2})$, and assigns each pair of d_1-subspaces (B_1, B_2) to the pair of functions $(\pi_{1|B_1}, \pi_{2|B_2})$. We consider the task of testing whether a given assignment Π is the P^2-direct product of some pair of strings $\pi_1, \pi_2 : \mathbb{F}^m \to \Sigma$. That

1. Choose two uniformly distributed d_1-subspaces B_1, B_2 of $\mathbb{F}^m$.
2. Choose two uniformly distributed d_0-subspaces $A_1 \subseteq B_1$, $A_2 \subseteq B_2$.
3. Accept if and only if $\Pi(B_1, B_2)_{|(A_1,A_2)} = \Pi(A_1, A_2)$.

Figure 5.5: The P^2-test

is, we are given an assignment Π , and in order to check whether Π is a P^2-direct product, we invoke the P^2-test, described in Figure 5.5.

It is easy to see that if Π is a P^2-direct product then the P^2-test always accepts. Again, it can be shown that if Π is "far" from being a P^2-direct product, then the P^2-test rejects with high probability, and that this holds even if Π is a randomized assignment. Formally, we have the following result.

Theorem 5.9.1 (Soundness of the P^2-test)**.** *There exist universal constants $h, c \in \mathbb{N}$ such that the following holds: Let $\varepsilon \geq h \cdot d_0 \cdot q^{-d_0/h}$, $\alpha \overset{\text{def}}{=} h \cdot d_0 \cdot q^{-d_0/h}$. Assume that $d_1 \geq h \cdot d_0$, $m \geq h \cdot d_1$. Suppose that an assignment Π passes the P^2-test with probability at least ε. Then, there exist two assignments π_1 and π_2 to $\mathbb{F}^m$ such that for B_1, B_2, A_1, A_2, distributed as in the P^2-test it holds that*

$$\Pr\left[\Pi(B_1, B_2)_{|(A_1,A_2)} = \Pi(A_1, A_2) \quad \text{and } \Pi(A_1, A_2) \overset{\alpha}{\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right) \quad \text{and } \Pi(B_1, B_2) \overset{\alpha}{\approx} \left(\pi_{1|B_1}, \pi_{2|B_2}\right)\right]$$

is at least $\Omega(\varepsilon^c)$.

In the rest of this section we prove Theorem 5.9.1. We denote by $\mathcal{P}$ the event in which the P^2-test accepts, that is, that $\Pi(B_1, B_2)_{|(A_1,A_2)} = \Pi(A_1, A_2)$. The core of the proof is the following lemma:

Lemma 5.9.2. *There exist universal constants $h', c' \in \mathbb{N}$ such that the following holds: Let $\varepsilon \geq h' \cdot d_0 \cdot q^{-d_0/h'}$, $\alpha' \overset{\text{def}}{=} h' \cdot d_0 \cdot q^{-d_0/h'}$. Assume that $d_1 \geq h' \cdot d_0$, $m \geq h' \cdot d_1$. If Π passes the P^2-test with probability at least ε then there exists an assignment $\pi_2 : \mathbb{F}^m \to \Sigma$ such that*

$$\Pr\left[\mathcal{P} \quad \text{and } \Pi(A_1, A_2)_{|A_2} \overset{\alpha'}{\approx} \pi_{2|A_2} \quad \text{and} \quad (B_1, B_2)_{|B_2} \overset{\alpha'}{\approx} \pi_{2|B_2}\right] \geq \Omega(\varepsilon^{c'}),$$

and symmetrically, there exists a function $\pi_1 : \mathbb{F}^m \to \Sigma$ such that

$$\Pr\left[\mathcal{P} \quad \text{and } \Pi(A_1, A_2)_{|A_1} \overset{\alpha'}{\approx} \pi_{1|A_1} \quad \text{and} \quad (B_1, B_2)_{|B_1} \overset{\alpha'}{\approx} \pi_{1|B_1}\right] \geq \Omega(\varepsilon^{c'}).$$

We prove Lemma 5.9.2 in Section 5.9.1.1. We turn to derive Theorem 5.9.1 from Lemma 5.9.2.

Proof of Theorem 5.9.1. The following proof is for the case where Π is not randomized, but it can be easily extended to the case where Π is randomized (see Remark 5.9.4 for details). We will choose h to be larger than the constant h' of Lemma 5.9.2, so we can apply this lemma. Let $\pi_2 : \mathbb{F}^m \to \Sigma$ be the assignment guaranteed by Lemma 5.9.2, and let Π' be an assignment that is obtained from Π as follows:

1. For every pair (A_1, A_2) for which $\Pi(A_1, A_2)_{|A_2} \overset{\alpha'}{\approx} \pi_{2|A_2}$, set $\Pi'(A_1, A_2) = \Pi(A_1, A_2)$.
2. For every other pair (A_1, A_2), set $\Pi'(A_1, A_2) = \perp$, where $\perp$ is some special value on which the test never accepts.
3. Set the pairs (B_1, B_2) similarly.

The probability ε' that the assignment Π' passes the P^2-test is at least $\Omega(\varepsilon^{c'})$ by the definition of π_2. By choosing h to be sufficiently larger than the corresponding constants of Lemma 5.9.2, we can make sure that ε' satisfies the requirements of Lemma 5.9.2. Therefore, we can deduce by Lemma 5.9.2 that there exists an assignment $\pi_1 : \mathbb{F}^m \to \Sigma$ such that

$$\Pr\left[\mathcal{P} \text{ and } \Pi'(A_1, A_2)_{|A_1} \overset{\alpha'}{\approx} \pi_{1|A_1} \text{ and } \Pi'(B_1, B_2)_{|B_1} \overset{\alpha'}{\approx} \pi_{1|B_1}\right] \geq \Omega((\varepsilon')^{c'}) = \Omega(\varepsilon^{(c')^2}).$$

We now choose $c = (c')^2$. Since the test never accepts when Π' answers $\perp$, we deduce that

$$\Pr\left[\mathcal{P} \text{ and } \Pi(A_1, A_2) \overset{\alpha'}{\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right) \text{ and } \Pi(B_1, B_2) \overset{\alpha'}{\approx} \left(\pi_{1|B_1}, \pi_{2|B_2}\right)\right] \geq \Omega(\varepsilon^c).$$

Choosing h such that $\alpha \geq \alpha'$ completes the proof. ■

Remark 5.9.3. Technically speaking, our use of the special value $\perp$ requires formal justification, since when defining the P^2-test and stating Lemma 5.9.2 we did not allow the use of such a special symbol. To this end, we observe that the use of $\perp$ can be implemented as follows: Let $\Sigma' = \Sigma \cup \{\perp_A, \perp_B\}$, where $\perp_A, \perp_B$ are symbols outside Σ. We first observe that Lemma 5.9.2 works just as well if we replace the alphabet Σ with the modified alphabet Σ', since Lemma 5.9.2 is oblivious to the choice of the alphabet. Now, whenever we wish to set $\Pi'(A_1, A_2) = \perp$ in the proof of Theorem 5.9.1, we actually set $\Pi'(A_1, A_2)$ to be the pair of functions that map all the vectors of A_1 and A_2 respectively to the symbol $\perp_A$. We deal with the case of $\Pi'(B_1, B_2) = \perp$ similarly, this time using the symbol $\perp_B$. It remains to observe that when assigning $\Pi'(A_1, A_2)$ this way, the P^2-test will always reject $\Pi'(A_1, A_2)$, since the assignment Π' never assigns pairs (B_1, B_2) with the symbol $\perp_A$. The same holds for the case of $\Pi'(B_1, B_2) = \perp$.

Remark 5.9.4. If Π is randomized, then the definition of Π' in the foregoing proof should be slightly changed to consider the internal randomness of Π. That is, we define Π' to be a randomized assignment, and obtain it from Π as follows. For every pair (A_1, A_2) and every internal randomness ω of Π, let us denote by (a_1, a_2) the output of Π on (A_1, A_2) and randomness ω. We define the output of Π' on (A_1, A_2) and randomness ω to be (a_1, a_2) if $a_2 \overset{\alpha'}{\approx} \pi_{2|A_2}$, and define it to be $\perp$ otherwise. The definition for pairs (B_1, B_2) is again similar.

5.9.1.1 The proof of Lemma 5.9.2

We prove Lemma 5.9.2 only for the assignment π_2, and the conclusion π_1 can be proved analogously. The proof proceeds in three steps. First, we rely on Theorem 5.2.2 (soundness of the P-test) to find for each pair of A_1, B_1 a direct product function that agrees (on average) with a good fraction of $\Pi(A_1, \cdot)$ and $\Pi(B_1, \cdot)$. Then, we show that for each A_1 separately, the number of distinct such functions is bounded. Next, we show that there is a single function π such that the probability that the test accepts and $\Pi(A_1, A_2)_{|A_2} \approx \pi_{|A_2}$ is non-negligible (A priori there could have been a different π for each A_1). Finally, we extend the latter result for d_1-subspaces B_1, B_2. Let h_1 be the universal constant whose existence is guaranteed in Theorem 5.2.2, and let α_1 be the corresponding value from Theorem 5.2.2.

Step 1. Consider the bipartite graph corresponding to the P-test, that is, the graph whose left vertices are d_0-subspaces and whose right vertices are d_1-subspaces, and such that a d_0-subspace A_1 is connected to a d_1-subspace B_1 by an edge if and only if $A_1 \subseteq B_1$. . We label an edge (A_1, B_1) by $\pi : \mathbb{F}^m \to \Sigma$ if

$$\Pr_{A_2, B_2}\left[\mathcal{P} \text{ and } \Pi(B_1, B_2)_{|B_2} \overset{\alpha_1}{\approx} \pi_{|B_2} \text{ and } \Pi(A_1, A_2)_{|A_2} \overset{\alpha_1}{\approx} \pi_{|A_2}\right] \geq \Omega\left(\varepsilon^4\right).$$

If no such π exists then do not label the edge.

Fix A_1, B_1. We will choose the universal constant h' to be at least $2 \cdot h_1$. If the probability of passing the P^2-test conditioned on A_1, B_1 is at least $\varepsilon/2$, then we claim that the edge is labeled. Indeed, define an assignment $\Pi_{(A_1,B_1)}$ by

$$\Pi_{(A_1,B_1)}(A_2) = \Pi\left(A_1, A_2\right)_{|A_2} \quad \text{and } \Pi_{(A_1,B_1)}(B_2) = \Pi\left(B_1, B_2\right)_{|B_2}.$$

If $\Pi_{(A_1,B_1)}$ passes the P-test with probability at least $\varepsilon/2$, then by Theorem 5.2.2 (soundness of the P-test) there is an assignment π as needed (since $h' \geq 2 \cdot h_1$).

Furthermore, observe that by averaging at least $\varepsilon/2$ of the edges (A_1, B_1) have conditional success at least $\varepsilon/2$, so (A_1, B_1) is labeled.

Step 2. Fix B_1 and let $L(B_1)$ be the labels on edges touching B_1. Consider the following "pruning" process: arbitrarily choose a label $\pi \in L(B_1)$ and remove all elements in $L(B_1)$ that are within relative Hamming distance $3\alpha_1$ of π. Repeat until no more labels can be removed. Let $L'(B_1)$ denote the remaining set of labels. The set $L'(B_1)$ has the following properties

- Every pair of labels in $L'(B_1)$ are at least $3\alpha_1$ apart, and
- Every $f \in L(B_1)$ is $3\alpha_1$-close to some label in $L'(B_1)$.

We prove that $|L'(B_1)| \leq O(1/\varepsilon^4)$, using an argument in the spirit of the Johnson bound: Suppose $L'(B_1) = \{\pi_1, \pi_2, \ldots\}$ is non-empty. For every $\pi_i \neq \pi_j \in L'(B)$ let us denote

$$\begin{aligned} p_i &\stackrel{\text{def}}{=} \Pr_{B_2}\left[\Pi\left(B_1, B_2\right)_{|B_2} \stackrel{\alpha_1}{\approx} \pi_{i|B_2}\right] \\ p_{i,j} &= \Pr_{B_2}\left[\Pi\left(B_1, B_2\right)_{|B_2} \stackrel{\alpha_1}{\approx} \pi_{i|B_2} \ \text{ and } \Pi\left(B_1, B_2\right)_{|B_2} \stackrel{\alpha_1}{\approx} \pi_{j|B_2}\right]. \end{aligned}$$

By the definition of the labels π_i, we know that for some universal constant η it holds that $p_i \geq \eta \cdot \varepsilon^4$ for every π_i. We upper bound the fractions $p_{i,j}$: We know that for every $\pi_i \neq \pi_j$ it holds that $\pi_i \stackrel{3\cdot\alpha_1}{\not\approx} \pi_j$. It follows that

$$\begin{aligned} p_{i,j} &\leq \Pr_{B_2}\left[\pi_{i|B_2} \stackrel{2\cdot\alpha_1}{\approx} \pi_{j|B_2}\right] \\ &\leq 1/\left(q^{d_1-2} \cdot \left(\alpha_1 - q^{-d_1}\right)^2\right) \\ &\leq \frac{1}{2} \cdot \eta^2 \cdot \varepsilon^8, \end{aligned}$$

where the second inequality follows by Lemma 5.2.4 (subspace-point sampler) and the third inequality holds for sufficiently large choice of h'. Now, by the inclusion-exclusion principle that

$$\begin{aligned} \sum_i p_i - \sum_{i \neq j} p_{i,j} &\leq 1 \\ |L'(B_1)| \cdot \left(\eta \cdot \varepsilon^4\right) - \frac{1}{2}\left|L'(B_1)\right|^2 \cdot \left(\frac{1}{2} \cdot \eta^2 \cdot \varepsilon^8\right) &\leq 1. \end{aligned}$$

The last inequality immediately implies that $|L'(B_1)| \leq 2/\left(\eta \cdot \varepsilon^4\right) = O(1/\varepsilon^4)$.

We define $L(A_1)$ similarly, and prune it to $L'(A_1)$. Imagine now choosing a random $\pi_{A_1} \in L'(A)$ for each A_1 and a random $\pi_{B_1} \in L'(B_1)$ for each B_1. An edge (A_1, B_1) is called alive if it is labeled by a function π that is $3\alpha'$-close to both π_{A_1} and π_{B_1}. We expect at least $1/\left|L'(A)\right|\left|L'(B)\right| = \Omega(\varepsilon^8)$ fraction of edges to be alive. Fix a choice of π_{A_1} and π_{B_1} for each A_1 and B_1 in a way that attains this expectation.

Step 3. Let $\mathcal{D}_1$ be the distribution of choosing a random d_1-subspace B_1 and two neighbors A_1, A_1' of it in the graph. Let $\mathcal{D}_2$ be the distribution of choosing two d_0-spaces A_1, A_1' independently and a random B_1 that is a common neighbor of them in the graph. The statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is small:

Claim 5.9.5. *For every $\kappa \in \mathbb{N}$, if the constant h' is sufficiently large then the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are δ-close for $\delta < \varepsilon^{24}/\kappa$.*

We defer the proof of this claim to Section 5.9.1.2. Now choose a random triplet A_1, A_1', B_1 according to $\mathcal{D}_1$. We lower bound the probability that both edges (A_1, B_1) and (A_1', B_1) are alive. This certainly holds if (i) $\Omega(\varepsilon^8)$ fraction of the edges adjacent to B are alive, and (ii) both edges (A_1, B_1) and (A_1', B_1) are alive. Part (i) holds with probability $\Omega(\varepsilon^8)$ and conditioned on this, Part (ii) holds with probability at least $\Omega(\varepsilon^{16})$. Altogether

$$\Pr_{(B_1,A_1,A_1')\sim\mathcal{D}_1}\left[(A_1,B_1),(A_1',B_1) \text{ are both alive}\right] = \Omega(\varepsilon^{24}).$$

Finally, if we let δ be the statistical distance of $\mathcal{D}_1$ and $\mathcal{D}_2$, and apply Claim 5.9.5 with sufficiently large choices of κ and h', then we have that

$$\Pr_{(B_1,A_1,A_1')\sim\mathcal{D}_2}\left[(A_1,B_1),(A_1',B_1) \text{ are both alive}\right] \geq \Omega(\varepsilon^{24}) - \delta = \Omega(\varepsilon^{24}).$$

Now fix A_1 such that the above holds when conditioning on A_1. This means that for at least $\Omega(\varepsilon^{24})$ fraction of the d_0-subspaces A_1' there exists a d_1-subspace B_1 such that both the edges (A_1, B_1) and (A_1', B_1) are alive. For each such A_1', it holds that the label of (A_1', B_1) is $3\alpha_1$-close to π_{B_1}, which in turn is $3\alpha_1$-close to the label of the edge (A_1, B_1), which is $3\alpha_1$-close to π_{A_1}. Thus, the label of (A_1', B_1) is is $9\alpha_1$-close to π_{A_1}. Let us denote by $\pi_{\left(A_1',B_1\right)}$ the label of the edge (A_1', B_1). Recall that by the definition of $\pi_{\left(A_1',B_1\right)}$ it holds that

$$\Pr_{A_2,B_2}\left[\mathcal{P} \text{ and } \Pi\left(A_1',A_2\right)_{|A_2} \overset{\alpha_1}{\approx} \pi_{\left(A_1',B_1\right)|A_2}\right] \geq \Omega\left(\varepsilon^4\right). \tag{5.12}$$

Since $\pi_{\left(A_1',B_1\right)} \overset{9\cdot\alpha_1}{\approx} \pi_A$ it holds by Lemma 5.2.4 (subspace-point sampler) that for a uniformly distributed d_0-subspace A_2:

$$\Pr_{A_2}\left[\pi_{\left(A_1',B_1\right)|A_2} \overset{10\cdot\alpha_1}{\not\approx} \pi_{A_1|A_2}\right] \leq \frac{1}{q^{d_0-2}\cdot\left(\alpha_1 - q^{-d_0}\right)^2}.$$

The latter expression can be made smaller than any constant times ε^4 by choosing h' to be sufficiently large. By subtracting that expression from Inequality 5.12, we obtain that

$$\Pr_{A_2,B_2}\left[\mathcal{P} \text{ and } \Pi\left(A_1',A_2\right)_{|A_2} \overset{\alpha_1}{\approx} \pi_{\left(A_1',B_1\right)|A_2} \text{ and } \pi_{\left(A_1',B_1\right)|A_2} \overset{10\cdot\alpha_1}{\approx} \pi_{A_1|A_2}\right] \geq \Omega\left(\varepsilon^4\right).$$

By letting $\pi_2 = \pi_{A_1}$ and choosing $c' = 28$, we have by the triangle inequality

$$\Pr_{A_1',A_2}\left[\mathcal{P} \text{ and } \Pi\left(A_1',A_2\right)_{|A_2} \overset{11\cdot\alpha_1}{\approx} \pi_{2|A_2}\right] \geq \Omega(\varepsilon^{24})\cdot\Omega\left(\varepsilon^4\right) = \Omega(\varepsilon^{c'}). \tag{5.13}$$

Step 4. It remains to show that the assignment Π agrees with π_2 on a non-negligible fraction of the B's. To this end, we observe that

$$\Pr\left[\mathcal{P} \text{ and } \Pi\left(A_1,A_2\right)_{|A_2} \overset{11\cdot\alpha_1}{\approx} \pi_{2|A_2} \middle| \Pi\left(B_1,B_2\right)_{|B_2} \overset{12\cdot\alpha_1}{\not\approx} \pi_{2|B_2}\right] \leq \frac{1}{q^{d_0-2}\cdot\left(\alpha_1/2\right)^2}. \tag{5.14}$$

To see it, note that it suffices to prove that

$$\Pr\left[\Pi\left(B_1,B_2\right)_{|A_2} \overset{11\cdot\alpha_1}{\approx} \pi_{2|A_2} \middle| \Pi\left(B_1,B_2\right)_{|B_2} \overset{12\cdot\alpha_1}{\not\approx} \pi_{2|B_2}\right] \le \frac{1}{q^{d_0-2}\cdot\left(\alpha_1-q^{-d_0}\right)^2} \le \frac{1}{q^{d_0-2}\cdot\left(\alpha_1/2\right)^2}.$$

The latter inequality is an immediate corollary of Lemma 5.2.4 (subspace-point sampler).

Now, by choosing h' to be sufficiently large so that the upper bound in Inequality 5.14 is sufficiently smaller than $\varepsilon^{c'}$, and by combining Inequality 5.13 with Inequality 5.14, we obtain that

$$\Pr\left[\mathcal{P} \text{ and } \Pi\left(A_1,A_2\right)_{|A_2} \overset{11\cdot\alpha_1}{\approx} \pi_{2|A_2} \text{ and } \Pi\left(B_1,B_2\right)_{|B_2} \overset{12\cdot\alpha_1}{\approx} \pi_{2|B_2}\right] \ge \Omega(\varepsilon^{c'}).$$

By setting h' such that $\alpha' \ge 12\cdot\alpha_1$ this concludes the proof of Lemma 5.9.2. ■

5.9.1.2 Proofs of Auxiliary Claim

Proof of Claim 5.9.5. Fix $\kappa \in \mathbb{N}$. In order to prove the claim, consider the event J which holds if and only if A and A' are independent. We argue that

$$\mathcal{D}_1 \overset{\delta/2}{\approx} \mathcal{D}_1|J = \mathcal{D}_2|J \overset{\delta/2}{\approx} D_2.$$

The fact that $\mathcal{D}_1|J = \mathcal{D}_2|J$ is exactly Proposition 5.2.4. We show that $\mathcal{D}_1 \overset{\delta/2}{\approx} \mathcal{D}_1|J$ and $\mathcal{D}_2 \overset{\delta/2}{\approx} \mathcal{D}_2|J$. The statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_1|J$ (respectively, $\mathcal{D}_2$ and $\mathcal{D}_2|J$) is exactly the probability that the event J does not occur under $\mathcal{D}_1$ (respectively $\mathcal{D}_2$). It follows immediately from Proposition 5.2.16 that $\Pr_{\mathcal{D}_1}[\neg J] \le 2\cdot d_0/q^{d_1-2\cdot d_0}$ and $\Pr_{\mathcal{D}_2}[\neg J] \le 2\cdot d_0/q^{m-2\cdot d_0}$. Both the latter expressions can indeed be made smaller than ε^{24}/κ by choosing sufficiently large h', as required. ■

5.9.2 The proof of Theorems 5.5.4 and 5.8.5

In the rest of this section we prove Theorems 5.5.4 and 5.8.5.

Theorem (5.5.4, the soundness of the S-test, restated)**.** *There exists a universal constants $h, c \in \mathbb{N}$ such that the following holds: Let $\varepsilon \ge h\cdot d_0\cdot q^{-d_0/h}$, $\alpha \overset{\text{def}}{=} h\cdot d_0\cdot q^{-d_0/h}$. Assume that $d_1 \ge h\cdot d_0$, $m \ge h\cdot d_1$. Suppose that a (possible randomized) assignment Π passes the S-test with probability at least ε. There exists an assignment $\pi : \mathbb{F}^m \to \Sigma$ for which the following holds. Let B_1, B_2 be uniformly distributed and independent d_1-subspaces of $\mathbb{F}^m$, let A_1 and A_2 be uniformly distributed d_0-subspaces of B_1 and B_2 respectively, and denote $A = A_1 + A_2$. Then:*

$$\Pr\left[\Pi\left(B_1,B_2\right)_{|(A_1,A_2)} = \Pi\left(A\right)_{|(A_1,A_2)} \text{ and } \Pi\left(B_1,B_2\right) \overset{\alpha}{\approx} \pi_{|(B_1,B_2)}\right] = \Omega\left(\varepsilon^{c}\right).$$

Remark 5.9.6. Note that in the foregoing restatement of Theorem 5.5.4 we denote the first universal constant by h, while in its original statement it was denoted by h'.

The intuition that underlies the proof is the following. Consider an adversary the chooses the proof Π. Since the S-test essentially contains a P^2-test, the adversary must choose the assignment Π such that for random d_0-subspaces A_1 and A_2, the assignment $\Pi\left(A_1+A_2\right)_{|(A_1,A_2)}$ is consistent with two assignments π_1, π_2 on A_1, A_2 respectively. On the other hand, given the sum A_1+A_2, the adversary can not deduce the choices of A_1 and A_2, and therefore he must label both of A_1 and A_2 with the same assignment in order to make the S-test accept. We conclude that π_1 and π_2 must be essentially the same. Details follow.

Let h' be the universal constant whose existence guaranteed in Theorem 5.9.1 (soundness of the P^2-test), and let α' be the corresponding value from Theorem 5.9.1. We choose c to be the same constant as in Theorem 5.9.1, and will choose the universal constant h to be at least h'.

Fix an assignment Π that passes the S-test with probability at least ε. We define a new assignment Π' that assigns values to pairs of d_0-subspaces and to pairs of d_1-subspaces of $\mathbb{F}^m$ (not necessarily independent) by choosing $\Pi'(B_1, B_2)$ (respectively $\Pi'(A_1, A_2)$) to be equal to $\Pi(B_1, B_2)$ (respectively $\Pi(A_1 + A_2)$) if B_1 and B_2 (respectively A_1 and A_2) are independent, and choosing Π' to be arbitrary otherwise. Observe that the assignment Π' passes the P^2-test whenever B_1 and B_2 are independent and Π passes the S-test. Furthermore, the probability that two uniformly distributed d_1-subspaces B_1 and B_2 of $\mathbb{F}^m$ are not independent is at most $d_1/q^{m-2\cdot d_1}$ by Proposition 5.2.16, and therefore Π' passes the P^2-test with probability at least $\varepsilon - d_1/q^{m-2\cdot d_1}$. For a sufficiently large choice of h, the latter probability is at least $\Omega(\varepsilon)$, and also matches the requirements of Theorem 5.9.1 (soundness of the P^2-test), so we can apply this theorem. It follows that there exist assignments $\pi_1, \pi_2 : \mathbb{F}^m \to \Sigma$ such that for uniformly distributed (not necessarily independent) B_1, B_2, $A_1 \subseteq B_1$, $A_2 \subseteq B_2$ it holds that

$$\begin{aligned} & \Pr[\Pi'(B_1, B_2)_{|(A_1,A_2)} = \Pi'(A_1, A_2) \\ & \quad \text{and } \Pi'(A_1, A_2) \overset{\alpha'}{\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right) \\ & \quad \text{and } \Pi'(B_1, B_2) \overset{\alpha'}{\approx} \left(\pi_{1|B_1}, \pi_{2|B_2}\right)] \\ = {} & \Omega(\varepsilon^c). \end{aligned} \tag{5.15}$$

The probability that B_1 and B_2 are not independent is at most $d_1/q^{m-2\cdot d_1}$, and the latter expression can be made smaller than any constant factor times ε^c by choosing h to be sufficiently large. Thus, Inequality 5.15 also holds for uniformly distributed *independent* B_1 and B_2. We now argue that

Claim 5.9.7. *For sufficiently large choice of h, it holds that $\pi_1 \overset{5\cdot\alpha'}{\approx} \pi_2$.*

We defer the proof of Claim 5.9.7 to the end of this section. We turn to prove the theorem. By Inequality 5.15 it holds for uniformly distributed and independent d_1-subspaces B_1 and B_2 of $\mathbb{F}^m$ that

$$\Pr\left[\Pi'(B_1, B_2)_{|(A_1,A_2)} = \Pi'(A_1, A_2) \quad \text{and } \Pi(B_1, B_2) \overset{\alpha'}{\approx} \left(\pi_{1|B_1}, \pi_{2|B_2}\right)\right] \geq \Omega(\varepsilon^c).$$

By Claim 5.9.7 it holds that $\pi_1 \overset{5\cdot\alpha'}{\approx} \pi_2$. Since B_2 is a uniformly distributed d_1-subspace of $\mathbb{F}^m$, this implies by Lemma 5.2.4 (subspace-point sampler) that

$$\Pr\left[\pi_{1|B_2} \overset{6\cdot\alpha'}{\approx} \pi_{2|B_2}\right] \geq 1 - \frac{1}{q^{d_1-2}\cdot(\alpha' - q^{-d_1})^2} \geq 1 - \frac{1}{q^{d_1-2}\cdot(\alpha'/2)^2}.$$

We conclude that

$$\begin{aligned} & \Pr\left[\Pi'(B_1, B_2)_{|(A_1,A_2)} = \Pi'(A_1, A_2) \quad \text{and } \Pi(B_1, B_2) \overset{7\cdot\alpha'}{\approx} \left(\pi_{1|B_1}, \pi_{1|B_2}\right)\right] \\ \geq {} & \Pr\left[\Pi'(B_1, B_2)_{|(A_1,A_2)} = \Pi'(A_1, A_2) \quad \text{and } \Pi(B_1, B_2) \overset{\alpha'}{\approx} \left(\pi_{1|B_1}, \pi_{2|B_2}\right) \quad \text{and } \pi_{1|B_2} \overset{6\cdot\alpha'}{\approx} \pi_{2|B_2}\right] \\ = {} & \Omega(\varepsilon^c) - \frac{1}{q^{d_1-2}\cdot(\alpha'/2)^2} \\ = {} & \Omega(\varepsilon^c), \end{aligned}$$

where the last equality holds for sufficiently large choice of h. the theorem now follows by defining $\pi = \pi_1$ and setting h to be sufficiently large such that $\alpha = 7\cdot\alpha'$. ■

Proof of Claim 5.9.7. For the sake of contradiction, assume that $\pi_1 \overset{5\cdot\alpha'}{\not\approx} \pi_2$. Let A be a uniformly distributed $2 \cdot d_0$-subspace A of $\mathbb{F}^m$ and let A_1 and A_2 be uniformly distributed and independent d_0-subspaces of A. By Lemma 5.2.4, it holds that

$$\Pr\left[\pi_{1|A} \overset{4\cdot\alpha'}{\not\approx} \pi_{2|A}\right] \geq 1 - \frac{1}{q^{2\cdot d_0 - 2} \cdot \left(\alpha' - q^{-2d_0}\right)^2} \geq 1 - \frac{1}{q^{2\cdot d_0 - 2} \cdot \left(\alpha'/2\right)^2}.$$

If $\pi_{1|A} \overset{4\cdot\alpha'}{\not\approx} \pi_{2|A}$ then by the triangle inequality it either holds that $\Pi(A) \overset{2\cdot\alpha'}{\not\approx} \pi_{1|A}$ or that $\Pi(A) \overset{2\cdot\alpha'}{\not\approx} \pi_{2|A}$. Since A_1 is a uniformly distributed d_0-subspace of A, it holds by Lemma 5.2.4 (subspace-point sampler) that

$$\Pr\left[\Pi(A)_{|A_1} \overset{\alpha'}{\not\approx} \pi_{1|A_1} \middle| \Pi(A) \overset{2\cdot\alpha'}{\not\approx} \pi_{1|A}\right] \geq 1 - \frac{1}{q^{2\cdot d_0 - 2} \cdot \left(\alpha'/2\right)^2}.$$

A similar claim can be made for π_2 and A_2. Now, if either $\Pi(A)_{|A_1} \overset{\alpha'}{\not\approx} \pi_{1|A_1}$ or $\Pi(A)_{|A_2} \overset{\alpha'}{\not\approx} \pi_{2|A_2}$ then by definition it holds that $\Pi(A)_{|(A_1,A_2)} \overset{\alpha'}{\not\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right)$. We conclude that

$$\Pr\left[\Pi(A)_{|(A_1,A_2)} \overset{\alpha'}{\not\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right) \middle| \pi_{1|A} \overset{4\cdot\alpha'}{\not\approx} \pi_{2|A}\right] \geq 1 - \frac{1}{q^{2\cdot d_0 - 2} \cdot \left(\alpha'/2\right)^2},$$

and therefore by lifting the conditioning and substituting $A = A_1 + A_2$ we obtain that for a uniformly distributed and independent d_0-subspaces A_1 and A_2 of $\mathbb{F}^m$ it holds that

$$\Pr\left[\Pi(A_1 + A_2)_{|(A_1,A_2)} \overset{\alpha'}{\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right)\right] \leq \frac{2}{q^{2\cdot d_0 - 2} \cdot \left(\alpha'/2\right)^2}.$$

On the other hand, by the definition of Π', Inequality 5.15 implies that for uniformly distributed and independent d_0-subspaces A_1 and A_2 of $\mathbb{F}^m$ it holds that

$$\Pr\left[\Pi(A_1 + A_2)_{|(A_1,A_2)} \overset{\alpha'}{\approx} \left(\pi_{1|A_1}, \pi_{2|A_2}\right)\right] \geq \Omega\left(\varepsilon^c\right).$$

By choosing h to be sufficiently large, the latter lower bound can be made larger than $2/\left(q^{2\cdot d_0 - 2} \cdot (\alpha')^2\right)$, and this is a contradiction. ■

Theorem 5.9.8 (5.8.5, list-decoding soundness of the S-test, restated)**.** *There exist universal constants $h, c \in \mathbb{N}$ such that for every $d_0 \in \mathbb{N}$, $d_1 \geq h \cdot d_0$, and $m \geq h \cdot d_1$, the following holds: Let $\varepsilon \geq h \cdot d_0 \cdot q^{-d_0/h}$, $\alpha \overset{\text{def}}{=} h \cdot d_0 \cdot q^{-d_0/h}$. Let Π be a (possibly randomized) assignment to $2d_0$-subspaces of $\mathbb{F}^m$ and to pairs of d_1-subspaces of $\mathbb{F}^m$. Then, there exists a (possibly empty) list of $L = O\left(1/\varepsilon^c\right)$ assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that*

$$\Pr\left[\Pi(B_1, B_2)_{|(A_1,A_2)} = \Pi(A)_{|(A_1,A_2)} \quad \text{and} \quad \nexists i \in [L] \ \ s.t. \ \Pi(B_1, B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right] < \varepsilon$$

Remark 5.9.9. Note that in the foregoing restatement of Theorem 5.8.5 we denote the first universal constant by h, while in its original statement it was denoted by h'.

The basic idea of the proof is as follows. We apply Theorem 5.5.4 to Π, thus "decoding" from it an assignment π^1. We then remove from Π the places at which it roughly agrees with π^1, resulting in an assignment Π^2. If the assignment Π^2 is accepted by the S-test with probability less than ε, then we are finished - the required list of assignments in this case consists only of π^1. Otherwise, the assignment Π^2

is accepted by the S-test with probability at least ε, and we can therefore "decode" a second assignment π^2 from Π^2. Next, we remove from Π^2 the places at which it roughly agrees with π^2, resulting in an assignment Π^3. We proceed in this manner, each time obtaining new assignments Π^i and π^i, until the conclusion of Theorem 5.8.5 holds.

We prove Theorem 5.8.5 only for non-randomized assignments Π, but the proof can easily be extended to randomized assignments, see Remark 5.9.11 for details. We choose the constants h and c to be the same as in Theorem 5.5.4. If the S-test accepts Π with probability less than ε then the theorem holds vacuously. We thus assume that the S-test accepts Π with probability at least ε. We show that for $L = O(1/\varepsilon^c)$ there exist assignments $\pi^1, \ldots, \pi^L : \mathbb{F}^m \to \Sigma$ such that

$$\begin{aligned} & \Pr\left[\Pi(B_1,B_2)_{|(A_1,A_2)} = \Pi(A)_{|(A_1,A_2)}\right] \\ & -\Pr\left[\Pi(B_1,B_2)_{|(A_1,A_2)} = \Pi(A)_{|(A_1,A_2)} \text{ and } \exists i \in [L] : \Pi(B_1,B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right] \\ \leq\ & \varepsilon. \end{aligned} \tag{5.16}$$

We construct the assignments $\pi^1, \ldots, \pi^L$ as follows. We begin by applying Theorem 5.5.4 to Π, obtaining the assignment π^1, and set $\Pi^1 \overset{\text{def}}{=} \Pi$. Then, for each $i \geq 1$ we define an assignment Π^{i+1} as follows.

1. For every pair of d_1-subspaces B_1, B_2 such that $\Pi^i(B_1,B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}$, we set $\Pi^{i+1}(B_1,B_2) = \perp$, where $\perp$ is a special symbol that the test always rejects. This is our formal way of "removing" $\Pi^i(B_1,B_2)$.
2. For every pair of d_1-subspaces B_1, B_2 such that $\Pi^i(B_1,B_2) \overset{\alpha}{\not\approx} \pi^i_{|(B_1,B_2)}$, we set $\Pi^{i+1}(B_1,B_2) = \Pi^i(B_1,B_2)$.
3. For every $2d_0$-subspace A, we set $\Pi^{i+1}(A) = \Pi^i(A)$.

Now, observe that

$$\begin{aligned} & \Pr\left[\Pi^{i+1}(B_1,B_2)_{|(A_1,A_2)} = \Pi^{i+1}(A)_{|(A_1,A_2)}\right] \\ =\ & \Pr\left[\Pi^{i}(B_1,B_2)_{|(A_1,A_2)} = \Pi^{i}(A)_{|(A_1,A_2)}\right] \\ & -\Pr\left[\Pi^{i}(B_1,B_2)_{|(A_1,A_2)} = \Pi^{i}(A)_{|(A_1,A_2)} \wedge \Pi^i(B_1,B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right], \end{aligned} \tag{5.17}$$

since we must have $\Pi^{i+1}(B_1,B_2)_{|(A_1,A_2)} \neq \Pi^{i+1}(A)_{|(A_1,A_2)}$ whenever $\Pi^{i+1}(B_1,B_2)_{|(A_1,A_2)} = \perp$, and the latter occurs whenever $\Pi^i(B_1,B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}$. If $\Pr\left[\Pi^{i+1}(B_1,B_2)_{|(A_1,A_2)} = \Pi^{i+1}(A)_{|(A_1,A_2)}\right] < \varepsilon$ then we set $L = i$ and finish the construction. Otherwise, we construct π^{i+1} by applying Theorem 5.5.4 to the assignment Π^{i+1} and setting π^{i+1} to be the resulting assignment.

It is easy to prove by induction that for every $i \in [L]$ it holds that

$$\begin{aligned} & \Pr\left[\Pi^{i+1}(B_1,B_2)_{|(A_1,A_2)} = \Pi^{i+1}(A)_{|(A_1,A_2)}\right] \\ =\ & \Pr_{A \subseteq B}\left[\Pi(B_1,B_2)_{|(A_1,A_2)} = \Pi(A)_{|(A_1,A_2)}\right] \\ & -\Pr_{A \subseteq B}\left[\Pi(B_1,B_2)_{|(A_1,A_2)} = \Pi(A)_{|(A_1,A_2)} \text{ and } \exists i \in [L] : \Pi_i(B_1,B_2) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right]. \end{aligned} \tag{5.18}$$

The proof of the Equality 5.18 goes essentially by summing over the probabilities of events of the form

$$\Pi^i(B_1,B_2)_{|(A_1,A_2)} = \Pi^i(A)_{|(A_1,A_2)} \text{ and } \Pi^i(B_1,B_2) \overset{\alpha}{\approx} \pi^j_{|(B_1,B_2)} \text{ and } \nexists j < i \text{ s.t. } \Pi^j(B_1,B_2) \overset{\alpha}{\approx} \pi^j_{|(B_1,B_2)},$$

for different values of i.

Finally, by combining Equality 5.18 with the fact that

$$\Pr\left[\Pi^{L+1}\left(B_1,B_2\right)_{|(A_1,A_2)} = \Pi^{L+1}\left(A\right)_{|(A_1,A_2)}\right] < \varepsilon,$$

it follows that the assignments $\pi^1,\ldots,\pi^L$ satisfy Inequality 5.16. To see that $L = O\left(1/\varepsilon^c\right)$, observe that for each i we have that

$$\Pr\left[\Pi^{i}\left(B_1,B_2\right)_{|(A_1,A_2)} = \Pi^{i}\left(A\right)_{|(A_1,A_2)} \text{ and } \Pi^{i}\left(B_1,B_2\right) \overset{\alpha}{\approx} \pi^i_{|(B_1,B_2)}\right] = \Omega\left(\varepsilon^c\right).$$

By Equality 5.17, this implies that the acceptance probability of Π^{i+1} is smaller than the acceptance probability of Π^i by at least ε^c, and therefore that the number of iterations can be at most $O\left(1/\varepsilon^c\right)$, as required.

Remark 5.9.10. As in the proof of Theorem 5.9.1 (soundness of the P^2-test), the use of the special symbol $\perp$ requires formal justification. This can be done as explained in Remark 5.9.3.

Remark 5.9.11. As in the proof of Theorem 5.9.1 (soundness of the P^2-test), if Π is randomized, then for each i the definition of Π^{i+1} should be slightly changed to consider the internal randomness of Π^i. That is, we define Π^{i+1} to be a randomized assignment, and obtain it from Π as follows. For every pair (B_1,B_2) and every internal randomness ω of Π^i, let us denote by (b_1,b_2) the output of Π_i on (B_1,B_2) and randomness ω. We define the output of Π^{i+1} on (B_1,B_2) and randomness ω to be $\perp$ if $(b_1,b_2) \overset{\alpha'}{\approx} \pi^i_{|(B_1,B_2)}$, and define it to be (b_1,b_2) otherwise. The definition for $2d_0$-spaces A can be changed similarly to include the internal randomness of Π^i. ■

Bibliography

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

[AL96] Sanjeev Arora and Carsten Lund. *Hardness of Approximations*. PW Publishing, 1996.

[ALM$^+$98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and intractability of approximation problems. *Journal of ACM*, 45(3):501–555, 1998. Preliminary version in FOCS 1992.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checkable proofs: A new characterization of NP. *Journal of ACM volume*, 45(1):70–122, 1998. Preliminary version in FOCS 1992.

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.

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

[Bab85] László Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[BGLR93] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alexander Russell. Efficient probabilistically checkable proofs and applications to approximations. In *STOC*, pages 294–304, 1993.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

[BSGH$^+$05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. pages 120–134, 2005.

[BSGH$^+$06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SIAM Journal of Computing*, 36(4):120–134, 2006.

[BSHLM09] Eli Ben-Sasson, Prahladh Harsha, Oded Lachish, and Arie Matsliah. Sound 3-query PCPPs are long. *TOCT*, 1(2), 2009.

[BSS06] Eli Ben-Sasson and Madhu Sudan. Robust locally testable codes and products of codes. *Random Struct. Algorithms*, 28(4):387–402, 2006. Preliminary version in APPROX-RANDOM 2004.

[BSS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008. Preliminary version in STOC 2005.

[BSV09a] Eli Ben-Sasson and Michael Viderman. Composition of semi-ltcs by two-wise tensor products. In *APPROX-RANDOM*, pages 378–391, 2009.

[BSV09b] Eli Ben-Sasson and Michael Viderman. Tensor products of weakly smooth codes are robust. *Theory of Computing*, 5(1):239–255, 2009.

[Cam98] Peter J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, Cambridge CB2 2RU, MA, USA, 1998.

[DFK$^+$99] Irit Dinur, Eldar Fischer, Guy Kindler, Ran Raz, and Shmuel Safra. PCP characterizations of NP: Towards a polynomially-small error-probability. In *STOC*, pages 29–40, 1999.

[DG08] Irit Dinur and Elazar Goldenberg. Locally testing direct product in the low error range. In *FOCS*, pages 613–622, 2008.

[DH09] Irit Dinur and Praladh Harsha. Composition of low-error 2-query PCPs using decodable PCPs. In *FOCS*, 2009.

[Din07] Irit Dinur. The PCP Theorem by gap amplification. *Journal of ACM*, 54(3):241–250, 2007. Preliminary version in STOC 2006.

[DM10] Irit Dinur and Or Meir. Derandomized parallel repetition of structured PCPs. In *IEEE Conference on Computational Complexity*, pages 16–27, 2010. Full version can be obtained as ECCC TR10-107.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards combinatorial proof of the PCP theorem. *SIAM Journal of Computing*, 36(4):155–164, 2006.

[DSW06] Irit Dinur, Madhu Sudan, and Avi Wigderson. Robust local testability of tensor products of ldpc codes. In *APPROX-RANDOM*, pages 304–315, 2006.

[FGL$^+$96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[FK95] Uriel Feige and Joe Kilian. Impossibility results for recycling random bits in two-prover proof systems. In *STOC*, pages 457–468, 1995.

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

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

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[Gol08] Oded Goldreich. Probabilistic proof systems: A primer. *Foundations and Trends in Theoretical Computer Science*, 3(1):1–91, 2008.

[GS00] Oded Goldreich and Shmuel Safra. A combinatorial consistency lemma with application to proving the PCP theorem. *SIAM J. Comput.*, 29(4):1132–1154, 2000.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost linear length. *Journal of ACM*, 53(4):558–655, 2006. Preliminary version in FOCS 2002, pages 13-22.

[IJKW08] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: simplified, optimized, and derandomized. In *STOC*, pages 579–588, 2008.

[IKW09] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. New direct-product testers and 2-query PCPs. In *STOC*, pages 131–140, 2009.

[Kho06] Subhash Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006.

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

[Lei92] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

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

[LPS88] Alexander Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[Mei09] Or Meir. Combinatorial PCPs with efficient verifiers. In *FOCS*, 2009. Full version is available as ECCC TR11-104.

[Mei10a] Or Meir. Combinatorial pcps with short proofs. 2010. In preparation.

[Mei10b] Or Meir. IP = PSPACE using error correcting codes. *Electronic Colloquium on Computational Complexity (ECCC)*, (137), 2010.

[MR08] Dana Moshkovitz and Ran Raz. Two query PCP with sub-constant error. In *FOCS*, 2008. Full version is available as ECCC TR08-071.

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

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

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *STOC*, pages 194–203, 1994.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

[Raz98] Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

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

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

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

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

[Sze99] Mario Szegedy. Many-valued logics and holographic proofs. In *ICALP*, pages 676–686, 1999.

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, pages 162–176, 1977.

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