Combinatorial PCPs with efficient verifiers[*]

Or Meir[†]

Abstract

The PCP theorem asserts the existence of proofs that can be verified by a verifier that reads only a very small part of the proof. The theorem was originally proved by Arora and Safra (J. ACM 45(1)) and Arora et al. (J. ACM 45(3)) using sophisticated algebraic tools. More than a decade later, Dinur (J. ACM 54(3)) gave a simpler and arguably more intuitive proof using alternative combinatorial techniques.

One disadvantage of Dinur's proof compared to the previous algebraic proof is that it yields less efficient verifiers. In this work, we provide a combinatorial construction of PCPs with verifiers that are as efficient as the ones obtained by the algebraic methods. The result is the first *combinatorial* proof of the PCP theorem for **NEXP** (originally proved by Babai et al., FOCS 1990), and a combinatorial construction of super-fast PCPs of Proximity for **NP** (first constructed by Ben-Sasson et al., CCC 2005).

1 Introduction

1.1 Background and Our Results

The PCP theorem [AS98, ALM$^+$98] is one of the major achievements of complexity theory. 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. The theorem has found many applications, most notably in establishing lower bounds for approximation algorithms.

The original proof of the PCP theorem by Arora et al. [AS98, ALM$^+$98] was based on algebraic techniques: Given a claim to be verified, they construct a PCP for the claim by first "arithmetizing" the claim, i.e., reducing the claim to a related "algebraic" claim about polynomials over finite fields, and then constructing a PCP for this algebraic claim. The PCP for the algebraic claim, in turn, requires an 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 the PCP theorem, since the theorem itself does not refer to algebra. Furthermore, those techniques seem to give little intuition as to why the theorem holds.

Given this state of affairs, it is an important goal to gain a better understanding of the PCP theorem and the reasons for which it holds. In her seminal paper, Dinur [Din07] has made a big step toward achieving this goal by giving an alternative proof of the PCP theorem using a combinatorial

[*] A preliminary version of this paper have appeared in FOCS 09 and as ECCC TR11-104. This research was partially supported by the Israel Science Foundation (grants No. 460/05 and 1041/08) and by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

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

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.

However, Dinur's PCP construction is still inferior to the algebraic constructions in few aspects. We believe that it is important to try to come up with combinatorial constructions of PCPs that match the algebraic constructions in those aspects, as this will hopefully advance our understanding of the PCP theorem. One such aspect, namely the soundness error of the PCPs, has been dealt with in a previous work [DM10]. In this work, we deal with a second aspect that concerns the running time of the verification procedure, to be discussed next.

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 their running time depends 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 is the first combinatorial proof of the PCP theorem for **NEXP**:

Theorem 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 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 $L \in$ **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 **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 is a combinatorial construction of PCPPs that allow super-fast verification:

Theorem 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 1.1 was already proved in [BSGH$^+$05], but we provide a combinatorial proof of this theorem.

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 1.2).

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 $\text{poly}(n)$.

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 $n^{\mathrm{poly}\log n}$. *Our second main technical contribution is showing how to modify the DR construction so as to have proofs of length* $\mathrm{poly}(n)$ *while maintaining the high efficiency of the verifiers.*

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 $n^{\mathrm{poly}\log n}$. 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 our work, 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 our work.

Organization of the paper In Section 2 we cover the relevant background for our work and give a formal statement of our main results. In Section 3 we give a high-level overview of our work. In Sections 4 and 5 we define super-fast assignment testers, which are central for our work, and develop tools for working with those testers. Finally, in Sections 6, 7, and 8, we prove our main technical results. A more detailed description of the organization of the paper is given at the end of the overview in Section 3.5.

2 Preliminaries and Our Main Results

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.

2.2 PCPs

As discussed in the introduction, the focus of this work 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 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 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 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 2.4. Note that Definition 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 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 2.6. The standard notation usually omits the running time t, and refers to the class $\mathbf{PCP}[r,q]$, which equals $\mathbf{PCP}[r,q,\mathrm{poly}(n)]$.

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 $\mathrm{poly}\log f(n)$ (this definition can be extended for functions f of many variables).

Theorem 2.7 (implicit in the PCP theorem of [BFL91, AS98, ALM$^+$98])**.** *For any admissible function $T(n) = \Omega(n)$, it holds that*

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

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 2.7, it only yields a weaker guarantee on the time complexity of the verifier:

Theorem 2.8 (Implicit in Dinur's PCP theorem [Din07])**.** *For any admissible function $T(n) = \Omega(n)$, it holds that*

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

Note that due to the difference in the time complexity, Theorem 2.7 implies Theorem 1.1 (PCP theorem for **NEXP**) as a special case for $T(n) = \exp(\mathrm{poly}(n))$, while Theorem 2.8 does not. *One contribution of this work is a combinatorial proof for Theorem 2.7.* This proof is actually an immediate corollary of our proof of Theorem 2.16 to be discussed in the next section.

2.3 PCPs of Proximity

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 1. In Section 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 2.9. A pair-language is a relation $L \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $x \in \{0,1\}^*$, we denote $L(w) \stackrel{\text{def}}{=} \{x : (w, x) \in L\}$.

Using Definition 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 $\text{poly}\left(|w|, \log|x|\right)$. 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 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 $t\left(|w|, m\right)$ steps, tosses at most $r\left(|w| + m\right)$ coins and makes at most $q\left(|w| + m\right)$ 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 $|x|$).

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 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:

[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.

- **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 $\Pr\left[V^{x,\pi}(w) \text{ rejects}\right] \geq \rho \cdot \operatorname{dist}\left(x, L(w)\right)$.

We refer to ρ as the rejection ratio of V.

Notation 2.12. Similarly to the PCP case, when we do not view constant factors as significant, we will drop ρ from the notation $\mathbf{PCPP}_{\rho}\left[r,q,t\right]$, since for every $0 < \rho_1 < \rho_2$ it holds that $\mathbf{PCPP}_{\rho_1}\left[r,q,t\right] = \mathbf{PCPP}_{\rho_2}\left[O(r), O(q), O(t)\right]$.

We turn to discuss two important features of Definition 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* $\operatorname{dist}\left(x, L(w)\right)$ 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 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 $|x| \leq p\left(n\right)$, and define a language $L' \overset{\text{def}}{=} \{w : \exists x \text{ s.t. } (w,x) \in PL\}$. Then it holds that $L' \in \mathbf{PCP}_{\rho}\left[r(n), q(n), t\left(n, p(n)\right)\right]$.*

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 2.14. The proof of Corollary 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.

2.3.2 Constructions of PCPPs and our results

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

Notation 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 $T\left(|x|, |w|\right)$, 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 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 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 2.16 implies Theorem 1.2 as a special case for $T(n) = \operatorname{poly}(n)$, while Theorem 2.17 does not, due to the difference in the time complexity. *Our main result is a combinatorial proof of Theorem 2.16.* Observe that Theorem 2.16 implies Theorem 2.7 using Corollary 2.13. We thus focus on proving Theorem 2.16.

We conclude the discussion in PCPPs by showing that Theorem 2.16 indeed implies a formal version of Theorem 1.2.

Corollary 2.18 (Special case of [BSGH+06, Proposition 2.5])**.** *Let $L \in \mathbf{NP}$, and let PL be the pair-language $\{(\lambda, x) : x \in L\}$ (where λ is the empty string). Then $PL \in \mathbf{PCPP}\left[O(\log m), O(1), \operatorname{poly}\log m\right]$ (note that here m denotes the length of x).*

Proof sketch. The main difficulty in proving Corollary 2.18 is that PL may not be decidable in polynomial time (unless $\mathbf{P} = \mathbf{NP}$), and therefore we can not use Theorem 2.16 directly. The naive solution would be to use Theorem 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 2.16 to construct a PCPP verifier V_2 for the *efficiently decidable* pair-language

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

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

[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 work.

Remark 2.19. We have stated Theorems 2.16 and 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 2.18. For details, see [BSGH$^+$06, Proposition 2.5].

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 $R_C \overset{\text{def}}{=} \frac{k}{l}$. 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 $\delta_C \overset{\text{def}}{=} \min_{c_1 \neq c_2 \in C} \{\text{dist}(c_1, c_2)\}$.

2.5 Routing networks

In our circuit decomposition method (see Sections 3.2 and 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 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 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* $\operatorname{poly}\log n$ *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 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 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 2.20, and take the union of all the resulting sets of paths. ■

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 3.4, we describe how this construction can be implemented efficiently.

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.

3.1.1 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 3.4).

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 $\tilde{O}\left(R^{c_0}\right)$, 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.

3.1.3 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 3.1.2).

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

[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.

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:

1. The assignment tester A decomposes φ, in a way to be explained in Section 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 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 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 3.4).

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 $n^{\operatorname{poly}\log n}$ 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.

3.2 Our circuit decomposition method

In this section we describe the circuit decomposition we use in Step 1 (of Section 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 3.3.2 for more details regarding how it is used.

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 3.1.3.

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 $O\left(\sqrt{n}\right) \times O\left(\sqrt{n}\right)$ 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 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 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 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 $G_{O(\sqrt{n})}$ are bounded by a constant.

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 3.1.3).

Lemma 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 3.4).

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 $R\left(s(n)\right)$ 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 $O\left(m(n)\right)$ 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 $R\left(O\left(m(n)\right)\right)$ 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 $R\left(O\left(m(n)\right)\right) \cdot R\left(s(n)\right)$ 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 $x \circ y'$.

3.3.2 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 3.3.1 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 3.3.1, 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 3.3.1 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 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 work, we define and analyze a generalization of the robustization technique of [DR06], which requires the assignment tester to have the following property:

Definition 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 (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.

3.4 Efficiency issues

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

3.4.1 Modifying the formalism

In Section 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 $\text{poly}\log n$, 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 $\text{poly}\log n$ bits. An assignment tester can therefore be defined[4] as 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.

3.4.2 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 3.3.1). Recall that those circuits are defined by $\eta_j \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m(n)} \xi_{i,j}$. 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.

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

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

3.4.3 A finer analysis of Dinur's amplification theorem

In order for our assignment testers to run in time 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 assignment 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 work have corresponding super-fast reverse listers, which turns out to be quite non-trivial in some of the constructions. See Section 5.1 for more details regarding the definition of reverse listers.

Remark 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.

3.4.4 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 work, 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 5.7 for more details.

3.4.5 Bounding the fan-in and fan-out

Throughout this work, 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

[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.

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 work, 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 5.8 for more details.

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). 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 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 work, we give a more sophisticated implementation that does preserve the running time of the reverse listers. See Section 5.4 for more details.

2. **Robustization**: As mentioned in Section 3.3.2, in this work 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).
   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 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 5.2 for more details.

3.5 Organization of the rest of the paper

In Section 4, we present the definition of super-fast assignment testers and state our main construction of assignment testers. In Section 5, we develop few generic tools that are used in our construction, but are also of independent interest, and which were mentioned in Sections 3.4.4 and 3.4.6 above. In Section 6, we prove our main theorem, relying on the decomposition method and on the tensor product lemma. In Section 7 we present our circuit decomposition method (which was sketched in Section 3.2). Finally, in Section 8, we prove the tensor product lemma, which was sketched in Section 3.3.

4 Super-Fast Assignment Testers: Definitions and Main Theorem

Recall that our final goal in this work is to construct the PCPPs that were stated in Theorem 2.16. As discussed in Section 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 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 the paper. 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 4.1, we review a DR-style definition of assignment testers, which does not support discussion of super-fast verifiers. Then, in Section 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 4.3, we state our construction of assignment testers, and prove that this construction implies the desired construction of PCPPs.

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:

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

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 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 $\rho \cdot \mathrm{dist}(x, \mathrm{SAT}(\varphi))$ 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 4.2. Note that Definition 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 2.11. It can be seen that the tested assignment x and the proof string π in Definition 4.1 correspond to the implicit input x and to the proof π in Definition 2.11, while the circuit φ corresponds to the explicit input. Furthermore, the rejection ratio ρ plays the same role as in Definition 2.11, and the outputs' number $R(n)$ is simply $2^{r(n)}$ where $r(n)$ is the randomness complexity in Definition 2.10.

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 4.1 does not support such a discussion due to the issues discussed next.

4.2.1 The size of the input circuit

The first issue that prevents Definition 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 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 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 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 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 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 $\operatorname{poly}\log|\varphi|$. 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 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 $\varphi_{n,m}^{\text{rep}}$ of size at most $\log^{d_{CL}} T(n,m)$, and there exists a Turing machine that on input (n,m) outputs $\varphi_{n,m}^{\text{rep}}$ 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.

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 $\psi_1^{\text{rep}},\ldots,\psi_{R(n)}^{\text{rep}}$ 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 $\psi_1^{\text{rep}},\ldots,\psi_{R(n)}^{\text{rep}}$. 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 $\psi_1^{\text{rep}},\ldots,\psi_{R(n)}^{\text{rep}}$.

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 5.1).

The foregoing considerations lead to the following definition:

Definition 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.

4.2.4 Syntactic modifications

Except for the foregoing issues, we make two more syntactic modifications to Definition 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 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.

4.2.5 The final definition

We are now ready to present the definition of assignment testers with which we will work throughout the paper. In the rest of the paper, whenever we refer to "assignment testers" we always refer to the following definition, and never to Definition 4.1. The following definition differs from Definition 4.1 only in the points discussed above.

Definition 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 $(\varphi^{\text{rep}}, m, i)$, 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 $(\psi_i^{\text{rep}}, q_i)$, 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 $(\varphi^{\text{rep}}, m, i, \kappa)$, 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 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 $\rho \cdot \text{dist}\left(x, \text{SAT}(\varphi)\right)$ 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 4.9. Note that Definition 4.8 specifies an upper bound on the size of the assignment tester (i.e. the tester size t), while Definition 4.1 had no restrictions on the running time of the assignment tester.

Remark 4.10. Definition 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 5.2 and 5.7, we do not keep track of the proof length ℓ throughout this work, and the representations' sizes n^{rep} and s^{rep} are of little significance.

4.3 Main theorem

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

Theorem 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 $A_{n,n^{\text{rep}}}$.*

We now show that the main theorem implies the desired construction of PCPPs (Theorem 2.16, restated below), and thus implies the desired construction of PCPs as well (Theorem 2.7):

Theorem (2.16, restated)**.** *For any admissible function $T(n,m)$ and a pair-language PL that is decidable in time T, it holds that*

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

Proof of Theorem 2.16 based on Theorem 4.11. The proof is straightforward, and consists of reducing PL to Circuit-Value as discussed in Section 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 4.11) to the resulting representation. Details follow.

Let $\ell(n)$ denote the proof length of the assignment tester of Theorem 4.11. Let $T(n,m)$ and PL be as in Theorem 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 4.6 to M, and let $\varphi_{n,m}^{\text{rep}}$ 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 $\varphi_{n,m}^{\text{rep}}$ is of size at most $\log^{d_{CL}} T(n,m)$, where d_{CL} is the constant from Fact 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 $\varphi^{\text{rep}} = \varphi_{|w|,|x|}^{\text{rep}}$ of the circuit $\varphi = \varphi_{|w|,|x|}$. Next, V computes the representation φ_w^{rep} 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 $A = A_{|\varphi_w|,|\varphi_w^{\text{rep}}|}$ of the main theorem on φ_w^{rep}, 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

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

coin tosses, since $|\varphi_w| = O\left(T\left(n,m\right)^2\right)$ (due to Fact 4.6). Similarly, the proof length of V is $\text{poly}\left(T(n,m)\right)$.

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 4.6. The representation φ_w^{rep} can be computed from φ_w^{rep} in time $\text{poly}\left(n, \log T(n,m)\right)$, 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 $\text{poly}\left(n, \log T(n,m)\right)$ (note that $|\varphi_w| = \text{poly}\left(T(n,m)\right)$ and that $|\varphi_w^{\text{rep}}| = n + \text{poly}\log T(n,m)$). 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 4.12 ([DR06, Theorem 1.2])**.** *Same as the main theorem, but with* $R(n) = n^{\text{poly}\log n}$ *and* $t(n, n^{\text{rep}}) = n^{\text{poly}\log n} + \text{poly}\left(n, n^{\text{rep}}\right)$.

Theorem 4.13 ([Din07])**.** *Same as the main theorem, but with* $t(n, n^{\text{rep}}) = \text{poly}(n, n^{\text{rep}})$.

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 5.1, we define the auxiliary notion of "reverse listers".

- In Section 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 work we can *do not need to keep track of the proof length* of our assignment testers.
- In Section 5.3, we adapt the gap amplification technique of Dinur to the setting of super-fast assignment testers by using reverse listers.
- In Section 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 5.5, we review useful known facts on error correcting codes in which membership can be verified efficiently. Such codes are used in Sections 5.6, 5.7, 5.8, and 8.
- In Section 5.6, we present a super-fast and general variant of the known robustization technique.
- In Section 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 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.

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 5.3), as well as in the proof of the tensor product lemma (see Section 8). In addition, as we show in Section 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 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 4.8), we define the reverse lister as a circuit that has few modes of operation.

Definition 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 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.

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 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 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 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 work all of our assignment testers have super-fast reverse listers. Thus, in order to simplify the presentation of this work, 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 5.4 to reduce the proof length to $R \cdot S$.

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 $\mathrm{poly}(1/\rho)$ (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 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* $\mathrm{poly}\left(s, \frac{1}{\rho}\right) \cdot R$, *outputs' size s_0, rejection ratio ρ_0, tester size at most* $t' \overset{\mathrm{def}}{=} \mathrm{poly}\left(s, \frac{1}{\rho}\right) \cdot \left(t + \mathrm{poly}\left(s^{\mathrm{rep}}, \log\left(R \cdot n\right)\right)\right)$, *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 5.5 from the original amplification theorem of [Din07] is not very difficult, but is tedious. Since it is not the focus of our work, we have not included its full proof.

Theorem 5.5 versus the original theorem of [Din07]. As mentioned above, the main difference between the original theorem of [Din07] and our Theorem 5.5 is the analysis of the tester size. There are also three minor differences between the original theorem of [Din07] and Theorem 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 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 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 5.4 for details on composition, and Remark 6.2 for details on the aforementioned trivial assignment tester.

The relation of reverse listers to Dinur's amplification theorem. As mentioned in Section 5.1, the proof of Theorem 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 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.

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 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 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 4.8). Then, for every assignment x to φ and for every proof π it holds that

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

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 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 5.9 below.

Theorem 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 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 5.9. Theorem 5.7 works under slightly more restrictive conditions than the composition theorem of [BSGH$^+$05]. Specifically, Theorem 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 5.23) such that it will guarantee that A_1 satisfies this requirement of the composition theorem.

Remark 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 5.7 and the proofs of previous composition theorems. The full details of the proof can be found in Appendix A.

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 5.7 assumes that all the output circuits of A_1 have the same input length. This calls for the following definition:

Definition 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 a composition lemma for the case where A_2 is input-uniform (the full proof of this lemma is given in Appendix A):

Lemma 5.12 (Composition Lemma for Input Uniform Inner Testers)**.** *Same as Theorem 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 5.7 now follows as an immediate corollary of Lemma 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 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 $t' = t + \operatorname{poly}\log(n, R)$, input representation size n^{rep}, and output representation size $s^{\mathrm{rep}} + \operatorname{poly}\log(n)$.*
  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'. ■

5.5 Efficiently verifiable error-correcting codes

Throughout this work 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 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 $\operatorname{poly}\log k$ and outputs a representation H_k^{rep} of H_k. In particular, H_k^{rep} is of size at most $\operatorname{poly}\log k$.*
4. *There exists an algorithm that on input $x \in \{0,1\}^k$, computes $C_k(x)$ in time $\operatorname{poly}(k)$.*

The codes of Fact 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 5.15. We note that the codes of Fact 5.14 are stronger than what we need in order to prove the main results of this paper. In particular, we could have relaxed Requirement 2 and require only that the circuit H_k will be of size $k\operatorname{poly}\log k$, 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 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 5.14.

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 4.7). Specifically, instead of defining a block to be a subset of $[m+\ell]$, we use the following definition of a block:

Definition 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 5.17. Let $Q : [q] \to [m+\ell] \cup \{\mathsf{dummy}\}$ be a queries function (as in Definition 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 $B_1, \dots B_b$ if it holds that $q = \sum_{j=1}^{b} |B_j|$ and

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

Definition 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 $B_1, \dots, B_p$ of $[m + \ell]$ whose images form a partition of $[m + \ell]$, such that the following holds: For each $i \in [R]$ there exist $B_{j_1}, \dots, B_{j_{b'}}$ (for $b' \leq b$) such that the queries function $Q_i^{A,\varphi}$ queries $B_{j_1}, \dots, B_{j_{b'}}$.

For each $i \in [R]$, we refer to the corresponding blocks $B_{j_1}, \dots, B_{j_{b'}}$ as the blocks queried by ψ_i (where ψ_i is the i-th output circuit obtained by applying A to φ). Note that $|B_j| \leq s$ 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 5.19. We stress that the different proof blocks in Definition 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 4.8 and 5.2), we define the block access circuit as a circuit that has a few modes of operation.

Definition 5.20. Let A, R, ℓ, b, φ^{rep}, m, $\psi_1, \dots, \psi_R$, and $B_1, \dots, B_p$ be as in Definition 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 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 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 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 $t' = O(t) + b \cdot \mathrm{poly}\log(R, s, n)$, input representation size n^{rep}, and output representation size $s^{\mathrm{rep}} + b \cdot \mathrm{poly}\log(s)$.*
  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 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 5.7) for details.

Proof sketch. Below we sketch the proof of Theorem 5.23, and in particular the construction of A' and the analysis of its robustness. The full proof of Theorem 5.23 is given in Appendix B. While we do not discuss below how to implement this construction in a super-fast manner, such implementation is straightforward (but tedious), and is presented in Appendix 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 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 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 $\psi_1^{\mathrm{rob}}, \ldots, \psi_R^{\mathrm{rob}}$: 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 SAT(φ) 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 SAT(ψ_i^{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 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 $\text{SAT}(\varphi)$ 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 $\text{SAT}(\varphi)$.

Suppose that x^{dec} is close to $\text{SAT}(\varphi)$. Note that this implies that x and x^{dec} are far from each other, since by assumption x is far from $\text{SAT}(\varphi)$. 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 $\text{SAT}(\varphi)$, 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 $\text{SAT}(\varphi)$, 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 $\text{SAT}(\varphi)$), 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 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. ■

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 5.7.2), which will also be used in the proof of the tensor product lemma (in Section 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 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 $n^{\text{rep}} \overset{\text{def}}{=} \text{poly}\log(n)$ 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 $t' = O(t) + \text{poly}(\log n, n^{\text{rep}})$, 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 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 5.7.1 below, we sketch the proof of Lemma 5.25. Then, in Section 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 8. We do not provide a the full proof of Lemma 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 8).

Remark 5.26. We note that the construction of the assignment tester A' of Lemma 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 work in this context is the observation that this construction can also be used to increase the input representation size of an assignment tester.

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 5.7.2), a variant of such a universal circuit can be implemented in size $n' \cdot n^{\mathrm{rep}\prime} \cdot \mathrm{poly}\log(n')$, and has a representation of size $\mathrm{poly}\log(n')$. 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 $(\varphi^{\mathrm{rep}}, x)$. 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 $n' \cdot n^{\mathrm{rep}\prime} \cdot \mathrm{poly}\log(n')$. 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 $(\zeta^{\mathrm{rep}}, y)$), 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 $(\zeta^{\mathrm{rep}}, y, z)$ where z is an auxiliary string, and act as follows:

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

2. If y is not a satisfying assignment of ξ, then U rejects $(\zeta^{\mathrm{rep}}, y, z)$ 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 $(\varphi^{\mathrm{rep}}, x, z)$ 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 $(\varphi^{\mathrm{rep}}, x, z)$ does not verify that x is close to a satisfying assignment of φ, but rather that the whole triplet $(\varphi^{\mathrm{rep}}, x, z)$ 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 $(\varphi^{\mathrm{rep}}, x, z)$ is close to the satisfying assignment $(\varphi^{\mathrm{rep}\prime}, x, z)$of U (where $\varphi^{\mathrm{rep}\prime}$ 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 5.5. The circuit $\hat{U}$ will now accept if and only if U accepts $(\zeta^{\mathrm{rep}}, y, z)$, *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 $(\varphi^{\mathrm{rep}}, c, x, z)$ 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 $(\varphi^{\mathrm{rep}}, c, x, z)$.

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 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.

5.7.2 Universal circuits

In this section we describe how to construct the universal circuits discussed in Section 5.7.1. This construction is used not only in this section, but also in Section 5.7.2 and in the proof of the tensor product lemma in Section 8. We begin by proving the following result:

Lemma 5.27. *There exists a polynomial time procedure that when given as input numbers n, $m \leq n$, and n^{rep}, outputs a representation $U^{\text{rep}} = U^{\text{rep}}_{n,n^{\text{rep}},m}$ (of size $\operatorname{poly}\log(n)$) of a circuit $U = U_{n,n^{\text{rep}},m}$ (of size $n \cdot n^{\text{rep}} \cdot \operatorname{poly}\log(n)$) 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 $(\zeta^{\text{rep}}, y, z)$ 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 $(\zeta^{\text{rep}}, y, z)$ for every string z.*

Proof Sketch. The construction of U is similar to the circuit decomposition method described in Sections 3.2 and 7, but is somewhat simpler. We first present a construction of a circuit U of size $n \cdot \operatorname{poly}(n^{\text{rep}}, \log n)$, and then explain how modify the construction to yield a circuit U of size $n \cdot n^{\text{rep}} \cdot \operatorname{poly}\log(n)$.

In order to construct a circuit U of size $n \cdot \operatorname{poly}(n^{\text{rep}}, \log n)$, 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 $n \cdot \operatorname{poly}(n^{\text{rep}}, \log n)$ by implementing a standard sorting algorithm, where the $\operatorname{poly}(n^{\text{rep}})$ 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 $n \cdot \operatorname{poly}(n^{\text{rep}})$, where the $\operatorname{poly}(n^{\text{rep}})$ 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 $n \cdot \operatorname{poly}(n^{\text{rep}}, \log n)$. In order to reduce the size of U to $n \cdot n^{\text{rep}} \cdot \operatorname{poly}\log n$, note that the $\operatorname{poly}(n^{\text{rep}})$ 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 $\tilde{O}(n^{\mathrm{rep}})$. After using this efficiency improvement, we get an implementation of U of size $n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log\,(n)$.

It is easy to verify that U has a representation of size $\mathrm{poly}\log\,(n)$, 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 $n^{\mathrm{rep}} \leq \mathrm{poly}\,(n)$ and hence $\mathrm{poly}\log\,(n^{\mathrm{rep}}) = \mathrm{poly}\log\,(n)$. The assumption that $n^{\mathrm{rep}} \leq \mathrm{poly}\,(n)$ can be made since every circuit of size n has a trivial representation of size $\mathrm{poly}\,(n)$, 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 5.27:

Corollary 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 $\hat{U}^{\mathrm{rep}} = \hat{U}^{\mathrm{rep}}_{n,n^{\mathrm{rep}},m}$ (of size $\mathrm{poly}\log\,(n, n^{\mathrm{rep}})$) of a circuit $\hat{U} = \hat{U}_{n,n^{\mathrm{rep}},m}$ (of size $n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log\,(n, n^{\mathrm{rep}})$) 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 $O(n^{\mathrm{rep}})$, 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 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.

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 7 to simplify our circuit decomposition method. Formally, we observe the following.

Lemma 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 $n' = n/n^{\text{rep}} \cdot \operatorname{poly}\log(n)$, outputs' number $R' = O(R)$, outputs' size $s' = \max\{s, O(1)\}$, rejection ratio $\rho' = \Omega(\rho)$, tester size $t' = O(t) + \operatorname{poly}(n^{\text{rep}}, \log n)$, input representation size n^{rep} and output size $s^{\text{rep}} + \operatorname{poly}\log s$.*
  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 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 5.30. Both the input representation lemma (Lemma 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 work.

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 (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 6.1, we first define the notion of circuit decomposition with matrix access, which is used in both lemmas. Then, in Section 6.2, we state the lemmas and prove the main theorem. Recall that our main theorem is the following.

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 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 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 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 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.

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 7.

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

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

  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'.*

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 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 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 6.7 (Single Iteration Lemma)**.** *Let s_0 and ρ_0 be the constants stated in the amplification theorem (Theorem 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 $n^{\mathrm{rep}} \overset{\mathrm{def}}{=} c \cdot \log^c n$ 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 $n' \overset{\mathrm{def}}{=} n^2/\mathrm{poly}\log n$ with outputs' number $O(R^2)$, outputs' size s_0, rejection ratio ρ_0, tester size at most $t' = O(t) + \mathrm{poly}\log\left(R, n\right)$, input representation size $n^{\mathrm{rep}\prime} \overset{\mathrm{def}}{=} c \cdot \log^c\left(n'\right)$ and output representation size s_0.*

  2. *A reverse lister RL' for A' of size at most t'.*

[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}.

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 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 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

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

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

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

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 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 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

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

input representation size $n^{\mathrm{rep}\prime}$ 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 (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* $A_{n,n^{\mathrm{rep}}}$*.*

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 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 6.7), and then use the input representation lemma (Lemma 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 paper, this can be established using the upper bound $\ell \le R \cdot s$ (Theorem 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 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 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 $n_{i+1} = n_i^2/d \cdot \log^d n$. In Appendix C, we show that for every i it holds that $n_i \ge \left(n_0/d \cdot 2^d \cdot \log^d n_0\right)^{2^i}$. 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

[8] Actually, since we do not need D to have matrix-access here, we can use the simpler decomposition described in Remark 6.2, and thus have an even simpler construction of A_0.

and tester size of A_k. By the single iteration lemma, there exists some constant h_R such that for each $i \geq 1$ it holds that $R_i \leq h_R \cdot (R_{i-1})^2$. It is not hard to prove by induction that

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

and therefore

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

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

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

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 $n^{\mathrm{rep}} \leq \mathrm{poly}(n)$, since every circuit of size n has a trivial representation of size $\mathrm{poly}(n)$. Let $n' = n \cdot n^{\mathrm{rep}} \cdot \mathrm{poly}\log n$, and let $n^{\mathrm{rep}\prime} = c \cdot \log^c(n')$. We construct the assignment tester $A_{n',n^{\mathrm{rep}\prime}}$, and invoke the input representation lemma (Lemma 5.25) on $A_{n',n^{\mathrm{rep}\prime}}$ to increase its input representation size to n^{rep}. We then output the resulting assignment tester as $A_{n,n^{\mathrm{rep}}}$. It is not hard to check that $A_{n,n^{\mathrm{rep}}}$ has the required parameters, and that the latter invocation of the input representation lemma is indeed legal, since the assignment tester $A_{n',n^{\mathrm{rep}\prime}}$ indeed satisfies the requirements of the input representation lemma for sufficiently large choice of c. ■

7 Circuit Decomposition Lemma

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

Lemma (6.4, Circuit decomposition lemma, restated)**.** *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 $t_D(n, n^{\mathrm{rep}})$.*

In Section 7.1, we give an overview of the proof which is more detailed than the one given in Section 3.2. Then, in Section 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 5.29).

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 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 7.1.2 we describe how to modify D into a super-fast decomposition.

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 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

$$\pi_{k_{M,g}} = \pi_{k_{N,g}} \text{ and } \pi_{k_{M,(g_1,g_2)}} = \pi_{k_{N,(g_1,g_2)}}. \tag{1}$$

Moreover, D must verify that Equality 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 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]).

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.

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 7.2.1, we describe the block structure of D. In Section 7.2.2 we describe the proof strings of D. In Section 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.

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 2.21, 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.

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 2.22, restated below.

Proposition (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 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.

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 $\tilde{O}\left(\sqrt{n}\right)$ 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 $\tilde{O}\left(\sqrt{n}\right)$.

2. $|V|+1 \le i \le |V|+\sqrt{n}$: Let $j \overset{\text{def}}{=} i-|V|$. 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 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 $h \overset{\text{def}}{=} (j-1)\cdot \max\{w_a, \sqrt{n}\}+u$ 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 $j \le \lceil m/w_a \rceil$, 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. $|V|+\sqrt{n}+1 \le i \le |V|+2\cdot\sqrt{n}$: Let $j \overset{\text{def}}{=} i-|V|-\sqrt{n}$. 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 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 $\tilde{O}(\sqrt{n})$. It is also not hard to see that both the circuit decomposition D and the reverse lister RL can be implemented in size $\operatorname{poly}\log n + O(n^{\text{rep}})$, 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 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.

8 Tensor Product Lemma

In this section we prove the general version of the tensor product lemma, restated below.

Lemma (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 $n_A^{\text{rep}}$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:*

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

  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*

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

  input representation size n_D^{rep}, and output representation size $s_A^{\text{rep}} + \operatorname{poly}\log(s_A)$.

  2. *An reverse lister RL' for A' of size at most t'.*

This section is organized as follows. In Section 8.1 we recall the ideas that underlie the construction of the assignment tester A', which were explained in Section 3, and sketch the technical complications that arise when realizing those ideas. In Section 8.2, we discuss a robustness property for decompositions, which differs from the notion of expected robustness defined in Section 5.4, and which is used in the proof of the tensor product lemma. Next, in Section 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 8.4, we complete the proof of the tensor product lemma using the intermediate assignment tester.

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: 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 $\eta_{i_A} \stackrel{\text{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D.i_A}$, 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. 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 8.1.1 and 8.1.2.

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 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 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 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 8.1.3 below.

8.1.2 Implementing A_I efficiently

We turn to discuss the efficiency of the implementation of A_I. As was mentioned in Section 3.4, 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 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 $(x \circ \pi_D)_{|Q_{i_D}^D}$. 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 $(x \circ \pi_D)_{|Q_{i_D}^D}$. 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 $\psi_{i_D}^{\mathrm{rep}}$ 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 $\psi_{i_D}^{\mathrm{rep}}$ to the κ-th input gate of ξ_{i_D,i_A}.

2. We redirect the queries of ξ_{i_A} to y to $(x \circ \pi_D)_{|Q_{i_D}^D}$, 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 $(x \circ \pi_D)_{|Q_{i_D}^D}$.

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 $(\psi_{i_D}^{\mathrm{rep}}, (x \circ \pi_D)_{|Q_{i_D}^D})$ satisfies U.

Next, we construct and output the circuits $\eta_1, \ldots, \eta_{R_A}$, which are defined as before by $\eta_{i_A} \overset{\mathrm{def}}{=} \bigwedge_{i_D=1}^{R_D} \xi_{i_D,i_A}$. 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 $\eta_{i_A}^{\mathrm{rep}}$ of a circuit η_{i_A} as follows. Suppose that the representation $\eta_{i_A}^{\mathrm{rep}}$ 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 $\eta_{i_A}^{\mathrm{rep}}$ simply invokes the representation $\xi_{i_A}^{\mathrm{rep}}$ 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 $\psi_{i_D}^{\mathrm{rep}}$. To this end, $\eta_{i_A}^{\mathrm{rep}}$ invokes D to compute $\psi_{i_D}^{\mathrm{rep}}$, 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 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 8.2. We mention that due to technical considerations that were discussed in Section 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.

8.1.3 Showing that the queries of A_I are contained in columns

Recall that in Section 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 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.

8.2 Robustization of decompositions with matrix access

As discussed in Section 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 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

$$\operatorname{dist}\left((x \circ \pi)_{|Q_i}, \operatorname{SAT}(\psi_i)\right) \geq \rho \cdot \operatorname{dist}(x, \operatorname{SAT}(\varphi)). \tag{2}$$

Remark 8.4. The difference between *existential* robustness and *expected* robustness (of Definition 5.6) is that the definition of existential robustness requires Equation (2) to hold for some $i \in [R]$, while the definition of expected robustness requires Equation (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 5.6 to transform decompositions into existentially robust ones. We actually use the following variant of the procedure of Section 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 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 $t' = O(t) + b \cdot$ $\text{poly}\log(R, s, n, \ell)$, input representation size $n^{\text{rep}\prime} = n^{\text{rep}}$, and output representation size $s^{\text{rep}\prime} = s^{\text{rep}} + b \cdot \text{poly}\log(s)$.*
  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 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 8.5 in the construction of the intermediate assignment tester A_I (Proposition 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 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 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 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 $B'_1,\dots,B'_p$ of D' to be equal to the blocks $B_1,\dots,B_p$ respectively. It remains to define a partition $B'_{p+1},\dots,B'_{p'}$ 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 $B'_{p+1},\dots,B'_{p'}$ 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. ■

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 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 5.11), and we will later obtain this property by applying the generic transformation that was described in Lemma 5.13 in Section 5.4.

Proposition 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:*

$$\begin{aligned} n_A &\geq s_D \cdot s_D^{\mathrm{rep}} \cdot \operatorname{poly}\log(s_D) \\ n_A^{\mathrm{rep}} &\geq \operatorname{poly}\log(s_D) \end{aligned}$$

- ***Output:***

1. *An assignment tester A_I for circuits of size n_D with outputs' number $R_I \overset{\mathrm{def}}{=} R_A$, outputs' size $s_I \overset{\mathrm{def}}{=} O(R_D \cdot s_A)$, rejection ratio $\Omega(\rho_D \cdot \rho_A)$, tester size*

$$t_I \overset{\mathrm{def}}{=} O(t_D + s_A \cdot t_A) + s_A \cdot \operatorname{poly}\left(s_A^{\mathrm{rep}}\right) + \operatorname{poly}\left(s_D^{\mathrm{rep}}\right) + \operatorname{poly}\log(R_D, s_D, n_D, R_A, s_A)$$

input representation size n_D^{rep}, and output representation size

$$s_I^{\mathrm{rep}} \overset{\mathrm{def}}{=} O(t_D + s_A \cdot \log s_D) + \operatorname{poly}\left(s_D^{\mathrm{rep}}\right) + \operatorname{poly}\log(R_D, s_D, n_D, R_A, s_A).$$

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 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 8.9. Throughout this section, we use the family of error correcting codes $\{C_k\}_{k=1}^{\infty}$ whose existence was stated in Fact 5.14 in Section 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 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 $\hat{U} = \hat{U}_{s_D, s_D^{\mathrm{rep}}, q_D}$ be the augmented universal circuit of Corollary 5.28, and recall that $\hat{U}$ has size $s_D \cdot s_D^{\mathrm{rep}} \cdot \operatorname{poly}\log(s_D) \leq n_A$ and has representation $\hat{U}^{\mathrm{rep}}$ of size $\operatorname{poly}\log(s_D) \leq n_A^{\mathrm{rep}}$. 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 $C(\zeta^{\mathrm{rep}})$ 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 $\hat{U}$.

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 $\ell_I \overset{\text{def}}{=} \ell_D + R_D \cdot (\ell_U + \ell_A)$ 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 $z^{i_D} \circ \pi_A^{i_D}$.

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 8.1.3, the output circuit η_{i_A} is defined by $\eta_{i_A} \overset{\text{def}}{=} \bigwedge_{i_D \in [R_D]} \xi_{i_D, i_A}$, 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.

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 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 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 $\eta_{i_A}^{\text{rep}}$. Recall that $\eta_{i_A}^{\text{rep}}$ computes the following functionality: the representation $\eta_{i_A}^{\text{rep}}$ takes as input the index of a gate g of η_{i_A} and an index h, and $\eta_{i_A}^{\text{rep}}$ 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 $\psi_{i_D}^{\text{rep}}$ or its encodings. In this case, the representation $\eta_{i_A}^{\text{rep}}$ 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 $\eta_{i_A}^{\text{rep}}$ the descriptions of all the representations $\psi_{i_D}^{\text{rep}}$ for every $i_D \in [R_D]$. However, this would cause the representation $\eta_{i_A}^{\text{rep}}$ to be of size at least R_D, which is too large.
   We therefore use the following alternative solution: We hardwire into $\eta_{i_A}^{\text{rep}}$ the input representation φ^{rep} and the decomposition D itself. Then, whenever $\eta_{i_A}^{\text{rep}}$ needs the description of $\psi_{i_D}^{\text{rep}}$ for any $i_D \in [R_D]$, the representation $\eta_{i_A}^{\text{rep}}$ invokes D on φ^{rep} in order to generate $\psi_{i_D}^{\text{rep}}$. We stress that the fact that $\eta_{i_A}^{\text{rep}}$ can compute the description of $\psi_{i_D}^{\text{rep}}$ using D is a key point in our construction of A_I, and is one of the central ideas of this work. What is actually happening here is that we use the fact that the circuits $\psi_1^{\text{rep}}, \ldots, \psi_{i_D}^{\text{rep}}$ are in a way "similar" and "uniform", in the sense that they can all be generated using D.
   We note that when $\eta_{i_A}^{\text{rep}}$ needs the *encoding* of the description of $\psi_{i_D}^{\text{rep}}$, it computes the description of $\psi_{i_D}^{\text{rep}}$ and encodes it via C, which can be done using a circuit of size $\text{poly}\left(\left|\psi_{i_D}^{\text{rep}}\right|\right) = \text{poly}\left(s_D^{\text{rep}}\right)$.

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 $\eta_{i_A}^{\text{rep}}$ 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 $\eta_{i_A}^{\text{rep}}$.

3. Suppose that g is one of the input gates of η_{i_A}. The representation $\eta_{i_A}^{\text{rep}}$ 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 our work 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 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 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 8.11).

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 on 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 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

$$|\mathbf{RevList}_{A_I,\varphi}(k)| = \sum_{h=1}^{\beta \cdot r_a} \left| \mathbf{RevList}_{A,\hat{U}}(\sigma_h) \right|$$

[9] We also use this fact in order to upper bound the proof length, as in Section 5.2, but this is done only for convenience and can be avoided.

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 $i \overset{\text{def}}{=} i_A$).

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 $v \overset{\text{def}}{=} (h-1) \cdot \left|\mathbf{RevList}_{A,\hat{U}}(\sigma_1)\right| + v'$.

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.

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 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

$$s_I^{\text{rep}} \overset{\text{def}}{=} O\left(t_D + s_A \cdot \log s_D\right) + \text{poly}\left(s_D^{\text{rep}}\right) + \text{poly}\log\left(R_D, s_D, n_D, R_A, s_A\right).$$

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 poly $\left(s_D^{\text{rep}}\right)$ 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 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

$$\begin{aligned} t_I &= s_I^{\mathrm{rep}} + O\left(s_A \cdot t_A\right) + s_A \cdot \mathrm{poly}\left(s_A^{\mathrm{rep}}\right) \\ &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \mathrm{poly}\left(s_A^{\mathrm{rep}}\right) + \mathrm{poly}\left(s_D^{\mathrm{rep}}\right) + \mathrm{poly}\log\left(R_D, s_D, n_D, R_A, s_A\right) \end{aligned}$$

To see it, note that the actions of A_I on index i_A consist of computing the representation $\xi_{i_A}^{\mathrm{rep}}$(which accounts for a term of $O(t_A)$), computing the description of ξ_{i_A} from $\xi_{i_A}^{\mathrm{rep}}$ (which is the reason for the $s_A \cdot \mathrm{poly}\left(s_A^{\mathrm{rep}}\right)$ 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 $\left(x \circ \pi_D\right)_{|Q_{i_D}^{D,\varphi}}$ is $(\rho_D \cdot \varepsilon)$-far from satisfying ψ_{i_D}. Now, let $c = C(\psi_{i_D}^{\mathrm{rep}})$ be encoding of the binary description of $\psi_{i_D}^{\mathrm{rep}}$, and consider the following assignment y to $\hat{U}$:

$$y \overset{\mathrm{def}}{=} \psi_{i_D}^{\mathrm{rep}} \circ \underbrace{c \circ \cdots \circ c}_{\alpha} \circ \underbrace{\left(x \circ \pi_D\right)_{|Q_{i_D}^{D,\varphi}} \circ \ldots \circ \left(x \circ \pi_D\right)_{|Q_{i_D}^{D,\varphi}}}_{\beta} \circ z^{i_D},$$

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 $y \circ \pi_A^{i_D}$. 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.

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 are technical, and contain 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 8.5 to D, thus obtaining an existentially robust decomposition D^{rob}.
2. The procedure applies the transformation of Lemma 5.13 to A, thus obtaining an input-uniform assignment tester A^{uni}.
3. The procedure applies Proposition 8.7 to D^{rob} and A^{uni} to construct the intermediate assignment tester A_I.
4. The procedure applies Theorem 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 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 8.5, it holds that D^{rob} is a circuit decomposition for circuits of size n_D with outputs' number $R_{D^{\mathrm{rob}}} \stackrel{\mathrm{def}}{=} 2 \cdot R_D$, outputs' size $s_{D^{\mathrm{rob}}} \stackrel{\mathrm{def}}{=} O(b \cdot s_D)$, existential robustness $\rho_{D^{\mathrm{rob}}} \stackrel{\mathrm{def}}{=} \Omega\left(1/b\right)$, tester size $t_{D^{\mathrm{rob}}} \stackrel{\mathrm{def}}{=} O\left(t_D\right) + b \cdot \operatorname{poly}\log\left(R_D, s_D, n_D\right)$, input representation size n_D^{rep}, and output representation size $s_{D^{\mathrm{rob}}}^{\mathrm{rep}} \stackrel{\mathrm{def}}{=} s_D^{\mathrm{rep}} + b \cdot \operatorname{poly}\log\left(s_D\right)$. Furthermore, D has b'-matrix access for some arbitrary b'.

2. By Lemma 5.13, it holds that A^{uni} is an assignment tester for circuits of size n_A with with outputs' number $R_{A^{\mathrm{uni}}} = 2 \cdot R_A$, outputs' size $s_{A^{\mathrm{uni}}} \stackrel{\mathrm{def}}{=} O(s_A)$, rejection ratio $\frac{1}{4} \cdot \rho_A$, tester size $t_{A^{\mathrm{uni}}} = t_A + \operatorname{poly}\log\left(n_A, R_A\right)$, input representation size n_A^{rep}, and output representation size $s_{A^{\mathrm{uni}}}^{\mathrm{rep}} = s_A^{\mathrm{rep}} + \operatorname{poly}\log(n_A)$.

3. By Proposition 8.7, it holds that A_I is an assignment tester for circuits of size n_D with outputs' number $R_I \stackrel{\mathrm{def}}{=} R_{A^{\mathrm{uni}}} = O(R_A)$, outputs' size $s_I \stackrel{\mathrm{def}}{=} O(R_{D^{\mathrm{rob}}} \cdot s_{A^{\mathrm{uni}}}) = O(R_D \cdot s_A)$, rejection ratio $\Omega\left(\rho_{D^{\mathrm{rob}}} \cdot \rho_{A^{\mathrm{uni}}}\right) = \Omega(\rho_A/b)$, tester size
$$\begin{aligned} t_I &\stackrel{\mathrm{def}}{=} O\left(t_{D^{\mathrm{rob}}} + s_{A^{\mathrm{uni}}} \cdot t_{A^{\mathrm{uni}}}\right) + s_{A^{\mathrm{uni}}} \cdot \operatorname{poly}\left(s_{A^{\mathrm{uni}}}^{\mathrm{rep}}\right) \\ &\quad + \operatorname{poly}\left(s_{D^{\mathrm{rob}}}^{\mathrm{rep}}\right) + \operatorname{poly}\log\left(R_{D^{\mathrm{rob}}}, s_{D^{\mathrm{rob}}}, n_{D^{\mathrm{rob}}}, R_{A^{\mathrm{uni}}}, s_{A^{\mathrm{uni}}}\right) \\ &= O\left(t_D + s_A \cdot t_A\right) + s_A \cdot \operatorname{poly}\left(s_A^{\mathrm{rep}}, \log n_A, \log R_A\right) \\ &\quad + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + b \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right), \end{aligned}$$
input representation size n_D^{rep}, and output representation size
$$\begin{aligned} s_I^{\mathrm{rep}} &\stackrel{\mathrm{def}}{=} O\left(t_{D^{\mathrm{rob}}} + s_{A^{\mathrm{uni}}} \cdot \log s_{D^{\mathrm{rob}}}\right) + \operatorname{poly}\left(s_{D^{\mathrm{rob}}}^{\mathrm{rep}}\right) + \operatorname{poly}\log\left(R_{D^{\mathrm{rob}}}, s_{D^{\mathrm{rob}}}, n_D, R_{A^{\mathrm{uni}}}, s_{A^{\mathrm{uni}}}\right) \\ &= O\left(t_D + s_A \cdot \log\left(s_D \cdot b\right)\right) + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right). \end{aligned}$$
Furthermore, A_I has $s_{A^{\mathrm{uni}}}$-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 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 5.23, it holds that $A_{I'}$ is an assignment tester for circuits of size n_D with outputs' number $R_{I'} \stackrel{\mathrm{def}}{=} 2 \cdot R_I = O(R_A)$, outputs' size $s_{I'} \stackrel{\mathrm{def}}{=} O(s_A \cdot s_I) = O\left(R_D \cdot s_A^2\right)$, robustness $\rho_{I'} \stackrel{\mathrm{def}}{=} \Omega\left(\rho_I / s_{A^{\mathrm{uni}}}\right) = \Omega\left(\rho_A / b \cdot s_A\right)$, tester size
$$\begin{aligned} t_{I'} &\stackrel{\mathrm{def}}{=} O\left(s_{A^{\mathrm{uni}}} \cdot t_I\right) + s_{A^{\mathrm{uni}}} \cdot \operatorname{poly}\log\left(R_I, s_I, n_D\right) \\ &= O\left(s_A \cdot t_D + s_A^2 \cdot t_A\right) + \operatorname{poly}\left(s_A, s_A^{\mathrm{rep}}, s_D^{\mathrm{rep}}, b\right) \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, n_A\right), \end{aligned}$$
input representation size n_D^{rep}, and output representation size
$$\begin{aligned} s_{I'}^{\mathrm{rep}} &\stackrel{\mathrm{def}}{=} s_I^{\mathrm{rep}} + s_{A^{\mathrm{uni}}} \cdot \operatorname{poly}\log\left(s_I\right) \\ &= O\left(t_D\right) + \operatorname{poly}\left(s_D^{\mathrm{rep}}, b, \log s_D\right) + s_A \cdot \operatorname{poly}\log\left(R_D, s_D, n_D, R_A, s_A, b\right). \end{aligned}$$

5. Finally, by the composition theorem (Theorem 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

$$\begin{aligned} t' &\overset{\text{def}}{=} O\left(t_{I'} + t_A\right) + \text{poly}\log\left(n_D, R_{I'}, \ell_{I'}, R_A, \ell_A\right) \\ &= O\left(s_A \cdot t_D + s_A^2 \cdot t_A\right) + \text{poly}\left(s_A, s_A^{\text{rep}}, s_D^{\text{rep}}, b\right) \cdot \text{poly}\log\left(R_D, s_D, n_D, R_A, s_A, n_A\right), \end{aligned}$$

input representation size n_D^{rep}, and output representation size $s^{\text{rep}\prime} \overset{\text{def}}{=} s_A^{\text{rep}} + \text{poly}\log(s_A)$. 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 $s_{I'}^{\text{rep}}$.

The required result follows.

Remark 8.11. We would like to highlight the fact that the output representation size s_I^{rep} 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 $s_{I'}^{\text{rep}}$ 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 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 6.

Acknowledgement. The author is grateful to Oded Goldreich for many useful discussions and ideas.

References

[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.

[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.

[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.

[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.

[Cam98] Peter J. Cameron. *Combinatorics: Topics, Techniques, Algorithms.* Cambridge University Press, Cambridge CB2 2RU, MA, USA, 1998.

[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.

[Lei92] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

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

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *STOC*, pages 194–203, 1994.

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[Sze99] Mario Szegedy. Many-valued logics and holographic proofs. In *ICALP*, pages 676–686, 1999.

A Proof of the Composition Lemma

In this appendix we provide the full proof of the composition lemma (Lemma 5.12). This proof follows the proof of the composition theorem of [BSGH$^+$05, Section 7], except for the construction of the reverse lister, which is new. Below, we recall the definition of input-uniform assignment tester as well as Lemma 5.12.

Definition (5.11, restated)**.** 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 $k \in [m]$.

Lemma (5.12, Composition Lemma for Input Uniform Inner Testers, restated)**.** *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"* input-uniform *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 $R' \overset{\mathrm{def}}{=} R_1 \cdot R_2$, outputs' size s_2, rejection ratio $\rho' \overset{\mathrm{def}}{=} \rho_1 \cdot \rho_2$, tester size $t' \overset{\mathrm{def}}{=} O(t_1 + t_2) + \mathrm{poly}\log(n, R_1, \ell_1, R_2, \ell_2)$, input representation size n^{rep}, and output representation size s_2^{rep}.*
2. *A reverse lister RL' for A' of size at most t'.*

In the rest of this appendix, we define A' and RL', and analyze their parameters.

The proof strings of A'. Let φ be a circuit of size n and let $x \in \mathrm{SAT}(\varphi)$. For every $i_1 \in [R_1]$, let ψ_{i_1} be i_1-th output circuit of A_1 when invoked on input circuit φ, and let $Q_{i_1}^{A_1,\varphi}$ be the corresponding queries function. We describe a proof string π' that convinces A' that x satisfies φ. The proof π' consists of two parts:

1. The first part of π' is the proof π_1 that convinces A_1 that x satisfies φ.
2. The second part of π' is a collection of proofs $\{\pi_{2,i_1}\}_{i_1 \in [R_1]}$, where π_{2,i_1} is the proof that convinces A_2 that $(x \circ \pi_1)_{|Q_{i_1}^{A_1,\varphi}}$ satisfies ψ_{i_1}.

Let ℓ_1 and ℓ_2 denote the proof lengths of A_1 and A_2 respectively. Observe that the assignment tester A' has proof length $\ell' = \ell_1 + R_1 \cdot \ell_2$. Since we wish to be able to compute the place of each string π_{2,i_1} within π' efficiently, we construct π' such that that for every $i_1 \in [R_1]$, the string π_{2,i_1} is found between the coordinates $m + \ell_1 + (i_1 - 1) \cdot \ell_2 + 1$ and $m + \ell_1 + i_1 \cdot \ell_2$.

The assignment tester A'. We describe the action of A' in circuit mode and in query mode separately. Let φ be a circuit of size n over m inputs and let φ^{rep} be a representation of φ of size at most n^{rep}. In *circuit mode*, on input $(\varphi^{\mathrm{rep}}, m, i)$ for $i \in [R']$, the assignment tester A' acts as follows:

1. A' views the input i, which is a number between 1 and $R_1 \cdot R_2$, as representing a pair of numbers (i_1, i_2) for $i_1 \in [R_1]$ and $i_2 \in [R_2]$.

2. The algorithm invokes A_1 on input $(\varphi^{\mathrm{rep}}, m, i_1)$ in circuit mode. The output is a representation $\psi_{i_1}^{\mathrm{rep}}$ of a circuit ψ_{i_1} of size s_1 over q_1 inputs. Note that by our assumption on A_1, the input length q_1 does not depend on i_1

3. The algorithm invokes A_2 on input $\left(\psi_{i_1}^{\mathrm{rep}}, q_1, i_2\right)$, resulting in a representation $\xi_{i_1,i_2}^{\mathrm{rep}}$ of a circuit ξ_{i_1,i_2} of size s_2 over q_{i_1,i_2} inputs.

4. The algorithm outputs the representation $\xi_{i_1,i_2}^{\mathrm{rep}}$.

We turn to describe the action of A' in query mode. In order to streamline the description, we fix an assignment x to φ and proof string π', and consider the action of A' when querying $x \circ \pi'$. When A' is required to compute the queries function $Q_i^{A',\varphi}$ that corresponds to ξ_{i_1,i_2} on input κ, the tester A' performs the following steps:

1. A' computes $\kappa_1 \overset{\text{def}}{=} Q_{i_2}^{A_2,\psi_{i_1}}(\kappa)$, which is the κ-th query of the i_2-th output circuit of A_2 on input circuit ψ_{i_1}. The index κ_1 is a coordinate of the string $(x \circ \pi_1)_{|Q_{i_1}^{A_1,\varphi}} \circ \pi_{2,i_1}$. We refer to $(x \circ \pi_1)_{|Q_{i_1}^{A_1,\varphi}}$ as the "first part" and to π_{2,i_1} as the "second part".
2. If $\kappa_1 \leq q_1$, then κ_1 falls in the first part. In such case, A' computes $Q_{i_1}^{A_1,\varphi}(\kappa_1)$, which is the corresponding coordinate of $x \circ \pi_1$, and outputs it.
3. If $\kappa_1 > q_1$, then κ_1 falls in the second part. In such case, A' computes $\kappa_2 = \kappa_1 - q_1$, which is the offset of κ_1 within the string π_{2,i_1}. Then, A' computes $m + \ell_1 + (i_1 - 1) \cdot \ell_2$, which is the offset of the string π_{2,i_1} within $x \circ \pi'$, and outputs $m + \ell_1 + (i_1 - 1) \cdot \ell_2 + \kappa_2$.

The parameters of A'. Clearly, A' is an assignment tester for circuits of size n and has outputs' number $R' = R_1 \cdot R_2$, outputs' size s_2, input representation size n^{rep} and output representation size s_2^{rep}. It is can also be seen that A' has tester size $t' = O(t_1 + t_2) + \text{poly}\log(n, R_1, \ell_1, R_2, \ell_2)$.

It remains to show that A' has rejection ratio $\rho' = \rho_1 \cdot \rho_2$. Let φ be circuit of size n, and let $\psi_1, \ldots, \psi_{R_1}$ and $\xi_{1,1}, \ldots, \xi_{R_1,R_2}$ be as before. Let $x \in \{0,1\}^m$ be an assignment to φ, let π' be a proof string for A', and let π_1 and $\{\pi_{2,i_1}\}_{i_1 \in [R_1]}$ be as before. We show that the fraction of the circuits ξ_{i_1,i_2} that reject $x \circ \pi'$ is at least $\rho' \cdot \text{dist}(x, \text{SAT}(\varphi))$.

Let I_1 and I_2 be random variables uniformly distributed in $[R_1]$ and in $[R_2]$ respectively. Then

$$\begin{aligned}
\Pr_{I_1,I_2}[\xi_{I_1,I_2} \text{ rejects}] &= \mathbb{E}_{I_1}\left[\Pr_{I_2}[\xi_{I_1,I_2} \text{ rejects}]\right] \\
\text{(Rejection ratio of } A_2) \quad &\geq \mathbb{E}_{I_1}\left[\rho_2 \cdot \text{dist}\left((x \circ \pi_1)_{|Q_{i_1}^{A_1,\varphi}}, \text{SAT}(\psi_{i_1})\right)\right] \\
&= \rho_2 \cdot \mathbb{E}_{I_1}\left[\text{dist}\left((x \circ \pi_1)_{|Q_{i_1}^{A_1,\varphi}}, \text{SAT}(\psi_{i_1})\right)\right] \\
\text{(Robustness of } A_1) \quad &\geq \rho_2 \cdot \rho_1 \cdot \text{dist}(x, \text{SAT}(\varphi)),
\end{aligned}$$

as required.

The reverse lister RL'. We conclude with describing the reverse lister RL'. Before going into the description, recall that since all the output circuits of A_1 have the same input length q_1, and since A_2 is input-uniform, it holds that the size of the reverse list $\mathbf{RevList}_{A,\psi_{i_1}}(\kappa_1)$ is the same for all circuits ψ_{i_1} and all coordinates $\kappa_1 \in [q_1]$. Let us denote the size of all those reverse lists by z.

In all of its modes of operation, the reverse lister RL' computes z by invoking the reverse lister RL_2 on one of the circuits ψ_{i_1} of size s_1 over q_1 inputs (again, z does not depend on ψ_{i_1}). The next steps of RL' depend on its mode of operation, and we describe them separately for each mode:

1. **Counting mode:** Recall that in this mode, RL' is invoked on input $(\varphi^{\text{rep}}, m, k)$ for $k \in [m + \ell']$ and is required to output $\left|\mathbf{RevList}_{A',\varphi}(k)\right|$. We consider two cases:

   (a) If $k \leq m + \ell_1$, then RL' invokes RL_1 in order to find $|\mathbf{RevList}_{A_1,\varphi}(k)|$, the number of output circuits of A_1 that query k, and outputs $|\mathbf{RevList}_{A_1,\varphi}(k)| \cdot z$. Note this is indeed the correct result, since the coordinate k is queried $|\mathbf{RevList}_{A_1,\varphi}(k)|$ times by the output circuits $\psi_1,\ldots,\psi_{R_1}$ of A_1, and for each such circuit ψ_{i_1}, the coordinate k is queried z times by the corresponding output circuits $\xi_{i_1,1},\ldots,\xi_{i_1,R_2}$ of A'.

(b) Otherwise, k is a coordinate of a proof π_{2,i_1} of A_2. In this case, RL' computes i_1 and the offset k_2 of k within π_{2,i_1}. Next, RL' invokes A_1 to compute $\psi_{i_1}^{\text{rep}}$. Finally, RL' invokes RL_2 on $\psi_{i_1}^{\text{rep}}$ to compute $\left|\mathbf{RevList}_{A_2,\psi_{i_1}}(k_2)\right|$, and outputs it.

2. **Retrieval mode:** Recall that in this mode, RL' is invoked on input $(\varphi^{\text{rep}},m,k,v)$ for $k\in[m+\ell']$ and $v\in\left[\left|\mathbf{RevList}_{A',\varphi}(k)\right|\right]$, and is required to output the v-th element of $\mathbf{RevList}_{A',\varphi}(k)$. We consider two cases:

   (a) Suppose that $k\le m+\ell_1$. Recall that in such case it holds that $\left|\mathbf{RevList}_{A',\varphi}(k)\right| = |\mathbf{RevList}_{A_1,\varphi}(k)|\cdot z$. The reverse lister RL' begins by viewing v as a pair of indices (v_1,v_2) where $v_1\in[|\mathbf{RevList}_{A_1,\varphi}(k)|]$ and $v_2\in[z]$. Then, RL' invokes RL_1 in order to find the v_1-th element (i_1,κ_1) of $\mathbf{RevList}_{A_1,\varphi^{\text{rep}}}(k)$. Next, RL' invokes A_1 to compute ψ_1^{rep}. Finally, RL' invokes RL_2 on $\psi_{i_1}^{\text{rep}}$ to find the v_2-th element (i_2,κ_2) of $\mathbf{RevList}_{A_2,\psi_{i_1}}(\kappa_1)$ and outputs (i',κ_2), where i' is the index of ξ_{i_1,i_2}.

   (b) Suppose that $k>m+\ell_1$, so k is a coordinate of a proof π_{2,i_1} of A_2. In this case, RL' computes i_1 and the offset k_2 of k within π_{2,i_1}. Next, RL' invokes A_1 to compute $\psi_{i_1}^{\text{rep}}$. Finally, RL' and then invokes RL_2 to compute the v-th element (i_2,κ_2) of $\mathbf{RevList}_{A_2,\psi_{i_1}}(k_2)$, computes the index i' of ξ_{i_1,i_2}, and outputs (i',κ_2).

3. **Reverse retrieval mode:** In this mode, RL' is given an element (i,κ) of $\mathbf{RevList}_{A',\varphi}(k)$, and is required to retrieve its index v within $\mathbf{RevList}_{A',\varphi}(k)$.. The reverse lister RL' views the index $i\in[R_1\cdot R_2]$ as a pair of indices (i_1,i_2) where $i_1\in[R_1]$ and $i_2\in[R_2]$. Then, we consider two cases:

   (a) Suppose that $k\le m+\ell_1$. The reverse lister RL' begins its computation by invoking A_1 in circuit mode to compute $\psi_{i_1}^{\text{rep}}$, and then invokes A_2 in query mode to find $\kappa_1 = Q_{i_2}^{A_2,\psi_{i_1}}(\kappa)$. Note that κ_1 is the κ-th query of ξ_{i_1,i_2} inside the input of ψ_{i_1}. Next, RL' invokes RL_1 to find the index v_1 such that (i_1,κ_1) is the v_1-th element of $\mathbf{RevList}_{A_1,\varphi}(k)$, and invokes RL_2 to find the index v_2 such that (i_2,κ) is the v_2-th element of $\mathbf{RevList}_{A_2,\psi_{i_1}}(\kappa_1)$. Finally, RL' translates the pair (v_1,v_2) to a number in $[|\mathbf{RevList}_{A_1,\varphi}(k)|\cdot z] = \left[\left|\mathbf{RevList}_{A',\varphi}(k)\right|\right]$, and outputs it.

   (b) Suppose that $k>m+\ell_1$, so k is a coordinate of a proof π_{2,i_1} of A_2. In this case, RL' computes i_1 and the offset k_2 of k within π_{2,i_1}. Next, RL' invokes A_1 to compute $\psi_{i_1}^{\text{rep}}$. Finally, RL' invokes RL_2 on $\psi_{i_1}^{\text{rep}}$ to find the index v such that (i_2,κ) is the v-th element of $\mathbf{RevList}_{A_2,\psi_{i_1}}(k_2)$, and outputs v.

It should be clear that RL' is indeed of size at most

$$\begin{aligned} t' &= O(t_1+t_2)+\operatorname{poly}\log(n,R_1,\ell_1,R_2,\ell_2) \\ &= O(t_1+t_2)+\operatorname{poly}\log(n,R_1,s_1,R_2,s_2), \end{aligned}$$

as required.

B Proof of the Robustization Theorem

In this appendix, we provide the full details of the proof of the robustization theorem:

Theorem (5.23, restated)**.** *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 $\Omega\left(\rho/b\right)$*, outputs' number* $2 \cdot R$*, outputs' size $O(b \cdot s)$, tester size $t' = O\left(t\right) + b \cdot \operatorname{poly}\log\left(R, s, n\right)$, input representation size n^{rep}, and output representation size $s^{\text{rep}} + b \cdot \operatorname{poly}\log\left(s\right)$.*
  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.

This appendix is organized as follows: In Section B.1 we describe the super-fast implementation of the assignment tester A', in Section B.2 we describe the super-fast implementation of the reverse lister RL', and in Section B.3 we show that A' is robust.

Remark B.1. Throughout this appendix we use the family of error correcting codes $\{C_k\}_{k=1}^{\infty}$ whose existence was stated in Fact 5.14 in Section 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 $l_k = O(k)$. Moreover, recall that for every $k \in \mathbb{N}$, there exists a circuit H_k of size $O(k)$, which takes as input strings $x \in \{0,1\}^k$ and $w \in \{0,1\}^{l_k}$, and accepts if and only if $w = C(x)$. Finally, recall that for every $k \in \mathbb{N}$, the circuit H_k has a representation H_k^{rep} of size $\operatorname{poly}\log k$, and that this representation can be computed in time $\operatorname{poly}\log k$. Again, with slight abuse of notation, we drop k and write H instead of H_k whenever k clear from the context

B.1 The construction of A'

Let φ be a circuit of size n over m inputs that has representation φ^{rep} of size n^{rep}. In this section we describe the action of the assignment tester A' on input circuit φ. As discussed in Section 5.6, the assignment tester A' outputs two types of circuits: the robustized circuits $\psi_1^{\text{rob}}, \ldots, \psi_R^{\text{rob}}$ and the consistency circuits $\psi_1^{\text{con}}, \ldots, \psi_R^{\text{con}}$.

The following description of A' consists of three parts: In Section B.1.1 we describe the proof strings of A', in Section B.1.2 we describe the construction of the robustized circuits and the corresponding queries functions, and in Section B.1.3 we describe the consistency circuits and the corresponding queries functions.

The numbering of the output circuits. Recall that by the definition of assignment testers, each output circuit of A' should be given an index between 1 to $R' = 2 \cdot R$. To this end, we choose the convention that the index of the i-th robustized circuit ψ_i^{rob} is i, and the index of the i-th consistency circuit ψ_i^{con} is $R + i$.

B.1.1 The proof strings of A'

Let φ be a circuit of size n over m inputs, and let ℓ denote the proof length of A. Let $x \in \mathrm{SAT}(\varphi)$. We describe a proof string π' that convinces A' that x satisfies φ. Recall that A has block access, and let $B_1, \ldots, B_p$ be the partition of $[m + \ell]$ to blocks that corresponds to φ. The proof π' consists of two parts: The first part is a proof π that convinces A that x satisfies φ. The second part consists of strings $E_1, \ldots, E_p$, where the j-th string E_j is the encoding $C\left((x \circ \pi)_{|B_j}\right)$.

In order to allow A' to access the strings E_j efficiently, we need to place them inside π' in a way that given j, it will be easy to compute the coordinates of E_j inside π'. To this end, recall that by Definition 5.18, for every $j \in [p]$ it holds that $|B_j| \leq s$. Thus, the length of each encoding E_j is upper-bounded by $l_s = O(s)$. Now, for each $j \in [p]$, we place the encoding E_j to between the coordinate $m + \ell + l_s \cdot (j - 1) + 1$ and the coordinate $m + \ell + l_s \cdot (j - 1) + l_{|B_j|}$. The coordinates between $m + \ell + l_s \cdot (j - 1) + l_{|B_j|} + 1$ and $m + \ell + l_{|s|} \cdot j$ will not be used.

For the rest of this appendix, we denote the proof length of A' by ℓ'.

B.1.2 The robustized circuits $\psi_1^{\mathrm{rob}}, \ldots, \psi_R^{\mathrm{rob}}$

Let $i \in [R]$. In what follows, we describe the output circuit ψ_i^{rob} and the queries function $Q_i^{\mathrm{rob}} \overset{\mathrm{def}}{=} Q_i^{A',\varphi}$ that are obtained from the action of A' on φ^{rep}, and explain how A' computes them. For the convenience of the following description, we fix some tested assignment x and some proof string π', and describe the action of ψ_i^{rob} when acting on a x and π'. Furthermore, we view π' as consisting of a string π of length ℓ and of a collection of additional strings E_j of length $l_{|B_j|}$ that are arranged as described in Section B.1.1.

Sketch. Assume that $i \leq R$. Let $B_{j_1}, \ldots, B_{j_{b'}}$ be the blocks queried by ψ_i (where $b' \leq b$). As described in Section 5.6, the circuit ψ_i^{rob} reads the strings $(x \circ \pi)_{|B_{j_1}}, \ldots, (x \circ \pi)_{|B_{j'_b}}$ and their purported encodings $E_{j_1}, \ldots, E_{j_{b'}}$, and verifies that:

1. ψ_i accepts the strings $(x \circ \pi)_{|B_{j_1}}, \ldots, (x \circ \pi)_{|B_{j_{b'}}}$.
2. The strings $E_{j_1}, \ldots, E_{j_{b'}}$ are the correct encodings of $(x \circ \pi)_{|B_{j_1}}, \ldots, (x \circ \pi)_{|B_{j_{b'}}}$ respectively.

Moreover, as mentioned in Section 5.6, the circuit ψ'_i reads each of the encodings $E_{j_1}, \ldots, E_{j_{b'}}$ several times, such that each of them constitutes about the same fraction of the input of ψ_i^{rob}. The circuit ψ_i^{rob} also verifies for each $h \in [b']$ that the different copies of E_{j_h} are identical (the reason for this check is discussed in Remark B.2 below).

Finally, recall that the "furthermore" part of the robustization theorem requires that all output circuits of A' have the same input length. In order to achieve this property, we make some additional dummy queries such that the input length of ψ_i^{rob} becomes $q' \overset{\mathrm{def}}{=} 2 \cdot b \cdot l_s$.

Details. We turn to give a more detailed description of ψ_i^{rob} and Q_i^{rob}. We view the input of ψ_i^{rob} as partitioned to b chunks, each of of length $2 \cdot l_s$. Each of the first b' chunks correspond to one of the blocks $B_{j_1}, \ldots, B_{j_{b'}}$, while the rest of the chunks consist of dummy queries. For each

$h \in [b']$, the h-th chunk consists of two parts, each of length l_s: The first part consists of the string $(x \circ \pi)_{|B_{j_h}}$ and of additional dummy queries that complete the length of this part to l_s. The second part consists of $\left\lfloor l_s / l_{|B_{j_h}|} \right\rfloor$ copies of the string E_{j_h}, and of additional dummy queries that complete the length of this part to l_s. Note that the input length of ψ_i^{rob} is indeed $q' = b \cdot 2 \cdot l_s$.

We move to describe the circuit ψ_i^{rob}. The circuit ψ_i^{rob} computes the AND function of several circuits, which are listed below:

1. The first circuit is the circuit ψ_i. That is, the circuit ψ_i^{rob} checks that ψ_i is satisfied by $(x \circ \pi)_{|B_{j_1}}, \ldots, (x \circ \pi)_{|B_{j_{b'}}}$.

2. For each chunk h that corresponds to a block B_{j_h}, there is a circuit that checks that all the copies of E_{j_h} are the same.

3. For each chunk h that corresponds to a block B_{j_h}, there is a copy of the circuit H, which checks that the first copy of E_{j_h} in the chunk is the correct encoding of $(x \circ \pi)_{|B_{j_h}}$ via the code C.

The implementation of the representation of ψ_i^{rob} is straightforward, and so is its computation by A'. We now explain how A' computes the queries function Q_i^{rob}. On input $\kappa \in [q']$, the assignment tester A' begins by computing the indices $j_1, \ldots, j_{b'}$ of the blocks queried by ψ_i, by using the Circuit to Blocks mode of BA. The assignment tester A' proceeds by computing the index h of the chunk to which the index κ belongs, the offset v of κ within this chunk, and the index j_h of the block B_{j_h}. Note that v and h can be computed efficiently. Next, A' considers the following cases:

1. If $v \leq |B_{j_h}|$, then κ belongs to $(x \circ \pi)_{|B_{j_h}}$. In such case, the assignment tester A' invokes BA in Block to Coordinate mode to determine the v-th coordinate of B_{j_h}, and outputs the result.

2. If $v > |B_{j_h}|$ but $v \leq l_s$ then κ is a dummy query, and A' outputs dummy.

3. If $v > l_s$ but $v \leq l_s + \left\lfloor l_s / l_{|B_{j_h}|} \right\rfloor \cdot l_{|B_{j_h}|}$, then this means that κ belongs to one of the copies of E_{j_h}. In such case, the circuit Q_i' computes the offset u of v within the copy of E_{j_h} (by using the formula $u = ((v - l_s - 1) \bmod l_{|B_{j_h}|}) + 1)$, and outputs $m + \ell + (j_h - 1) \cdot l_s + u$ (recall that E_{j_h} starts at the coordinate $m + \ell + (j_h - 1) \cdot l_s + 1$ of π').

4. If $v > l_s + \left\lfloor l_s / l_{|B_{j_h}|} \right\rfloor \cdot l_{|B_{j_h}|}$, then κ is a dummy query, and A' outputs dummy.

This concludes the computation of $Q_i^{\mathrm{rob}}(\kappa)$.

Remark B.2. At first glance, it may seem odd that the circuit ψ_i^{rob} needs to check that the copies of each E_j are equal, since it will never be invoked on an input in which those copies are not equal. However, recall that the definition of robustness requires that the input of ψ_i^{rob} will be far (in expectation) from every satisfying assignment of ψ_i'. Now, had the circuit ψ_i^{rob} not checked that the copies of E_j are equal, it would have had satisfying assignments in which those copies are not equal. Such satisfying assignments could violate the aforementioned robustness requirement, even though they will never occur as actual inputs of ψ_i^{rob}. This is the reason that the equality check between the copies of each E_j is required.

B.1.3 The consistency circuits $\psi_1^{\text{con}}, \ldots, \psi_R^{\text{con}}$

Let $B_{1,\ldots,}B_a$ be the assignment blocks. We associate each assignment block with $z \overset{\text{def}}{=} \lfloor R/a \rfloor$ consistency circuits, and the remaining (R mod a) consistency circuits are dummy circuits that always accept. If ψ_i^{con} is a consistency circuit that is associated with an assignment block B_j, then ψ_i^{con} queries $x_{|B_j}$ and E_j, and checks that E_j is the correct encoding of $x_{|B_j}$. Furthermore, ψ_i^{con} queries $x_{|B_j}$ and E_j multiple times and makes some additional dummy queries such that the input length of ψ_i^{con} becomes $q' = 2 \cdot b \cdot l_s$.

We now provide a more detailed description. Recall that $z \overset{\text{def}}{=} \lfloor R/a \rfloor$. The consistency circuit ψ_i^{con} is associated with the assignment block B_j for $j = \lceil i/z \rceil$. If $j > a$, then ψ_i^{con} is the dummy circuit that always accepts and makes no queries. For the rest of this section, on we only consider the case where $j \leq a$.

The input of ψ_i^{con} consists of two halves, each of length $b \cdot l_s$. The first half consists of $\lfloor b \cdot l_s / |B_j| \rfloor$ consecutive copies of $x_{|B_j}$, where the remaining coordinates are dummy coordinates. The second half consists of $\lfloor b \cdot l_s / |E_j| \rfloor$ consecutive copies of E_j, where the remaining coordinates are dummy coordinates. The circuit ψ_i^{con} accepts if and only if the following conditions hold:

1. All the purported copies of $x_{|B_j}$ are equal.
2. All the purported copies of E_j are equal.
3. The first purported copy of E_j is the correct encoding via C of the first purported copy of $x_{|B_j}$ (this is checked using the circuit H).

The reason for the first two conditions is as described in Remark B.2 It is easy to construct an efficient representation for ψ_i^{con} given the width of B_j, and this width can be determined using the circuit BA.

We conclude by describing the computation of the queries function Q_i^{con}. On input $\kappa \in [q']$ in query mode, the assignment tester A' acts as follows:

1. Suppose that $\kappa \leq b \cdot l_s$, in which case κ belongs to the first half of the input of ψ_i^{con}. If $\kappa > \lfloor b \cdot l_s / |B_j| \rfloor \cdot |B_j|$ then A' outputs dummy. Otherwise, A' computes the offset $v \overset{\text{def}}{=} (\kappa - 1) \mod |B_j| + 1$, and outputs $B_j(v)$ (which can be computed using the Block to Coordinate mode of BA).
2. Suppose that $\kappa \leq b \cdot l_s$, in which case κ belongs to the second half of the input of ψ_i^{con}. Let $\kappa' = \kappa - b \cdot l_s$. If $\kappa' > \lfloor b \cdot l_s / |E_j| \rfloor \cdot |E_j|$ then A' outputs dummy. Otherwise, A' computes the offset $v \overset{\text{def}}{=} (\kappa' - 1) \mod |E_j| + 1$, and outputs $m + \ell + (j-1) \cdot l_s + v$.

This concludes the computation of $Q_i^{\text{con}}(\kappa)$.

B.2 The reverse lister RL'

We describe the implementation of RL'. Fix a representation φ^{rep} of a circuit φ over m inputs, and suppose that RL' is invoked on φ^{rep} and on a coordinate $k \in [m + \ell']$. Below, we describe the action of RL' on φ^{rep}, m, and k.

We first consider the case in which $k \leq m + \ell$, i.e., k belongs to the tested assignment x or to the proof string π of the original assignment tester A. Let j denote the index of the block B_j to which the coordinate k belongs, let a denote the number of assignment blocks, and let $z \overset{\text{def}}{=} \lfloor R/a \rfloor$ be as in Section B.2. Observe that both j and a can be computed efficiently using BA. We turn to describe the behavior of RL' in each of mode of operation:

1. **Counting mode:** In this mode, RL' is required to retrieve $\left|\mathbf{RevList}_{A',\varphi}(k)\right|$. The reverse lister RL' begins by using RL to compute $|\mathbf{RevList}_{A,\varphi}(k)|$. If $k > m$, then RL' outputs $|\mathbf{RevList}_{A,\varphi}(k)|$. Otherwise, RL' uses BA to compute $|B_j|$, and outputs

$$|\mathbf{RevList}_{A,\varphi}(k)| + \lfloor z \rfloor \cdot \lfloor b \cdot l_s / |B_j| \rfloor .$$

The reason is that every coordinate $k \in [m+\ell]$ is queried by robustized circuits ψ_i^{rob} exactly the same number of times that it is queried by the original output circuits ψ_i of A, and that every assignment coordinate $k \in [m]$ is also queried by $\lfloor z \rfloor$ consistency circuits, for $\lfloor b \cdot l_s / |B_j| \rfloor$ times by each consistency circuit.

2. **Retrieval:** In this mode, RL' is required to retrieve the v-th element of $\mathbf{RevList}_{A',\varphi}(k)$ for some index $v \in \left[\left|\mathbf{RevList}_{A',\varphi}(k)\right|\right]$. We consider two cases:

   (a) If $v \leq |\mathbf{RevList}_{A,\varphi}(k)|$, then v corresponds to a query by a robustized circuit. In this case, RL' invokes RL to compute the v-th element (i,κ) of $\mathbf{RevList}_{A,\varphi}(k)$. Next, RL' uses BA to determine the index u such that k is the u-th coordinate of B_j, and the index h such that B_j is the h-th block that is queried by ψ_i - if B_j is queried by ψ_i multiple times, then index h is chosen according to the copy of B_j to which κ belongs. Finally, RL' sets $\kappa' \stackrel{\mathrm{def}}{=} (2 \cdot l_s) \cdot h + u$ and outputs (i,κ')

   (b) If $v > |\mathbf{RevList}_{A,\varphi}(k)|$, then $k \leq m$ and v corresponds to a query by a consistency circuit. Recall that there are $\lfloor z \rfloor \cdot \lfloor b \cdot l_s / |B_j| \rfloor$ such queries. Let $v' = v - |\mathbf{RevList}_{A,\varphi}(k)|$. The reverse lister now views v' as pair of indices (i,h), where $i \in [z]$ and $h \in [\lfloor b \cdot l_s / |B_j| \rfloor]$. Next, the reverse lister RL' invokes BA to compute the offset u of k within the block B_j. Finally, RL' computes $\kappa \stackrel{\mathrm{def}}{=} (h-1)\cdot |B_j| + u$, and outputs the pair $(R+i,\kappa)$ (Recall that $R+i$ is the index of the i-th consistency circuit).

3. **Reverse Retrieval:** In this mode, RL' is given an element (i,κ) of $\mathbf{RevList}_{A',\varphi}(k)$, and is required to retrieve its index v within $\mathbf{RevList}_{A',\varphi}(k)$. We consider two cases:

   (a) $i \leq R$: In this case, i is the index of a robustized circuit. The reverse lister RL' begins by computing the indices of $j_1,\ldots,j_{b'}$ of the blocks that are queried by ψ_i, by using the Circuit to Blocks mode of BA. Then, RL' computes the place κ_0 of k within the input of the output circuit ψ_i of A. The place κ_0 can be computed from the widths of $B_{j_1},\ldots,B_{j_{b'}}$, and from the index of the chunk to which κ belongs. Finally, RL' invokes RL to compute the index of (i,κ_0) within $\mathbf{RevList}_{A,\varphi}(k)$ and outputs it.

   (b) $i > R$: In this case, let $i' = i - R$, and note that i is the index of $\psi_{i'}^{\mathrm{con}}$. The reverse lister RL' begins by computing $\iota \stackrel{\mathrm{def}}{=} (i'-1) \bmod z + 1$, which is the index of $\psi_{i'}^{\mathrm{con}}$ among the consistency circuits that are associated with B_j. Next, RL' computes $u = \lceil \kappa / |B_j| \rceil$, which is the copy of $x_{|B_j}$ in the input of $\psi_{i'}^{\mathrm{con}}$ to which κ belongs. Finally, RL' views the pair of numbers $\iota \in [z]$ and $u \in [\lfloor b \cdot l_s / |B_j| \rfloor]$ as a single number $v \in [z \cdot \lfloor b \cdot l_s / |B_j| \rfloor]$ and outputs it.

It remains to deal with the case where $k > m+\ell$, which means that k belongs to some encoding E_j. The implementation of RL' in this case is very similar to the previous case, with the main differences being the following:

1. The reverse lister RL' begins by finding a coordinate $k' \leq m+\ell$ that belongs to the block B_j that is encoded by E_j. Then, whenever we used the list $\mathbf{RevList}_{A,\varphi}(k)$ in the case of $k \leq m+\ell$, we now use the list $\mathbf{RevList}_{A,\varphi}(k')$.

2. Robustized circuits query multiple copies of each encoding E_j, and this can be dealt with in the same way we deal with the fact that consistency circuits query multiple copies of each $x_{|B_j}$.

This concludes the construction of RL'.

B.3 The robustness of A'

In this section, we show that A' is robust, and in particular has robustness $\Omega(\rho/b)$. Let φ be a circuit of size n over m inputs, let φ^{rep} be a representation of φ, let $\psi_1^{\mathrm{rob}}, \ldots, \psi_R^{\mathrm{rob}}$ and $\psi_1^{\mathrm{con}}, \ldots, \psi_R^{\mathrm{con}}$ be the output circuits of A' when invoked on φ^{rep}, and let $Q_1^{\mathrm{rob}}, \ldots, Q_R^{\mathrm{rob}}$ and $Q_1^{\mathrm{con}}, \ldots, Q_R^{\mathrm{con}}$ be the corresponding query functions. Similarly, let $\psi_1, \ldots, \psi_R$ and $Q_1, \ldots, Q_R$ be the output circuits and query functions of A when invoked on φ^{rep}. Let $x \in \{0,1\}^m$ be an assignment to φ, let $\pi' \in \{0,1\}^{\ell'}$ a proof string for A', and let $\varepsilon_x = \mathrm{dist}(x, \mathrm{SAT}(\varphi))$. We show that either

$$\mathbb{E}_{i\in[R]}\left[\mathrm{dist}\left((x\circ\pi')_{|Q_i^{\mathrm{rob}}}, \mathrm{SAT}\left(\psi_i^{\mathrm{rob}}\right)\right)\right] \geq \Omega\left(\rho\cdot\varepsilon_x/b\right), \tag{3}$$

or

$$\mathbb{E}_{i\in[R]}\left[\mathrm{dist}\left((x\circ\pi')_{|Q_i^{\mathrm{con}}}, \mathrm{SAT}\left(\psi_i^{\mathrm{con}}\right)\right)\right] \geq \Omega\left(\rho\cdot\varepsilon_x\right), \tag{4}$$

and this will imply the robustness of A'. In the following argument, recall that we denote the relative distance of the codes $\{C_k\}_{k=1}^{\infty}$ by δ_C.

We view the proof string π' as consisting of a proof string π of A and of a collection of purported encodings E_j. For every $j \in [p]$, let E_j^{dec} be the codeword of C that is nearest to E_j. Let $x^{\mathrm{dec}} \in \{0,1\}^m$ and $\pi^{\mathrm{dec}} \in \{0,1\}^{\ell}$ be the assignment of φ and proof string of A that are encoded by the encodings E_j^{dec}, i.e., x^{dec} and π^{dec} satisfy $E_j^{\mathrm{dec}} = C\left(\left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|B_j}\right)$ for every block B_j. Note that x^{dec} and π^{dec} are well defined, since the blocks $B_1, \ldots, B_p$ are disjoint. Our argument now proceeds as described in Section 5.6: We argue that if $\mathrm{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) \geq \frac{1}{2}\cdot\varepsilon_x$, then $x\circ\pi'$ must be far (in expectation) from satisfying the robustized circuits ψ_i^{rob} (so Inequality 3 holds), and that otherwise $x\circ\pi'$ must be far (in expectation) from satisfying the consistency circuits ψ_i^{con} (so Inequality 4 holds).

The case where $\mathrm{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) \geq \frac{1}{2}\cdot\varepsilon_x$. Suppose that $\mathrm{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) \geq \frac{1}{2}\cdot\varepsilon_x$. We prove that Inequality 3 holds. By the rejection ratio of A, it holds for at least $\rho\cdot\frac{1}{2}\cdot\varepsilon_x$ of the indices $i \in [R]$ that the circuit ψ_i of A rejects $\left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|Q_i}$. We prove that for each such index i it holds that

$$\mathrm{dist}\left((x\circ\pi')_{|Q_i^{\mathrm{rob}}}, \mathrm{SAT}(\psi_i^{\mathrm{rob}})\right) \geq \delta_C/8b,$$

and this will imply Inequality 3, as required.

Fix $i \in [R]$ such that the circuit ψ_i of A rejects $\left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|Q_i}$. Let w' be a string nearest to $(x\circ\pi')_{|Q_i^{\mathrm{rob}}}$ that satisfies ψ_i^{rob}. Let w be the input to ψ_i that is obtained by projecting w' to $B_{j_1}, \ldots, B_{j_{b'}}$. Clearly, $w \neq \left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|Q_i}$, so there exists $\kappa \in [q_i]$ such that w and $\left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|Q_i}$ differ on the coordinate κ. Let B_{j_h} be the block to which κ belongs, i.e., B_{j_h} is the unique block such that $Q_i(\kappa) \in B_{j_h}$.

Recall that $(x\circ\pi')_{|Q_i^{\mathrm{rob}}}$ contains $\lfloor l_s/|B_{j_h}|\rfloor$ copies of E_{j_h}, and observe that the corresponding part of w' contains $\lfloor l_s/|B_{j_h}|\rfloor$ copies of some codeword $E_j^{w'}$ of C (here we use the fact that ψ_i^{rob} checks that the different copies of E_j are equal, see Remark B.2). Furthermore, recall that E_j^{dec} is the codeword of C nearest to E_j. Since w and $\left(x^{\mathrm{dec}}\circ\pi^{\mathrm{dec}}\right)_{|Q_i}$ differ on the coordinate κ, it holds that $E_j^{w'} \neq E_j^{\mathrm{dec}}$. This implies that $E_j^{w'}$ and E_j^{dec} are δ_C-far from each other, and therefore E_j is $\delta_C/2$-far from $E_j^{w'}$. Finally, it is easy to verify that the copies of E_j form at least $\frac{1}{3b}$-fraction of the input of ψ_i^{rob}, and therefore

$$\operatorname{dist}\left(\left(x \circ \pi'\right)_{|Q_i^{\mathrm{rob}}}, \mathrm{SAT}(\psi_i^{\mathrm{rob}})\right) = \operatorname{dist}\left(\left(x \circ \pi'\right)_{|Q_i^{\mathrm{rob}}}, w'\right) \geq \delta_C/3b.$$

This concludes the proof of Inequality 3.

The case where $\operatorname{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) < \frac{1}{2} \cdot \varepsilon_x$**.** Suppose that $\operatorname{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) < \frac{1}{2} \cdot \varepsilon_x$. We prove that Inequality 4 holds. By the triangle inequality, it holds that

$$\operatorname{dist}\left(x, x^{\mathrm{dec}}\right) \geq \varepsilon_x - \operatorname{dist}\left(x^{\mathrm{dec}}, \mathrm{SAT}(\varphi)\right) \geq \frac{1}{2} \cdot \varepsilon_x.$$

Let $B_1, \ldots, B_a$ be the assignment blocks, and recall that $B_1, \ldots, B_a$ are all of the same width. Furthermore recall that the number of non-dummy coordinates in each assignment block B_j is between $|B_j|/3$ and $|B_j|$. Given the latter fact, it is not hard to prove that

$$\mathbb{E}_{j \in [a]}\left[\operatorname{dist}(x_{|B_j}, x^{\mathrm{dec}}_{|B_j})\right] \geq \frac{1}{3} \cdot \operatorname{dist}\left(x, x^{\mathrm{dec}}\right) \geq \frac{1}{6} \cdot \varepsilon_x.$$

We now prove that for each $j \in [a]$ and each consistency circuit ψ_i^{con} that is associated with B_j, it holds that

$$\operatorname{dist}\left(\left(x \circ \pi'\right)_{|Q_i^{\mathrm{con}}}, \mathrm{SAT}(\psi_i^{\mathrm{con}})\right) \geq \Omega\left(\operatorname{dist}(x_{|B_j}, x^{\mathrm{dec}}_{|B_j})/b\right) \tag{5}$$

and this will imply Inequality 4, as required.

We turn to prove Inequality 5. Fix $j \in [a]$ and consistency circuit ψ_i^{con} that is associated with h. Recall that $(x \circ \pi')_{|Q_i^{\mathrm{con}}}$ consists of $\lfloor b \cdot l_s / |B_j| \rfloor$ copies of $x_{|B_j}$ and of $\lfloor b \cdot l_s / |E_j| \rfloor$ copies of E_j. Let w' be a string nearest to $(x \circ \pi')_{|Q_i^{\mathrm{con}}}$ that is accepted by ψ_i^{con}. Note that w' consists of $\lfloor b \cdot l_s / |B_j| \rfloor$ copies of a string $w \in \{0,1\}^{|B_j|}$ and of $\lfloor b \cdot l_s / |E_j| \rfloor$ copies of the encoding $C(w)$. Now, we consider two possible cases:

1. $w = x^{\mathrm{dec}}$: In this case, it holds that $x_{|B_j}$ and w differ on $\operatorname{dist}(x_{|B_j}, x^{\mathrm{dec}}_{|B_j})$ fraction of bits. Since the copies of $x_{|B_j}$ form at least $(1/3)$ fraction of the input of ψ_i^{con}, it follows that

$$\operatorname{dist}\left(\left(x \circ \pi'\right)_{|Q_i^{\mathrm{con}}}, \mathrm{SAT}(\psi_i^{\mathrm{con}})\right) \geq \frac{1}{3} \cdot \operatorname{dist}(x_{|B_j}, x^{\mathrm{dec}}_{|B_j}),$$

and hence Inequality 5 holds.

2. $w \neq x^{\mathrm{dec}}$: In this case, it holds that $C(w) \neq E_j^{\mathrm{dec}}$. Now, recall that E_j^{dec} is a codeword of C that is nearest to E_j, and therefore E_j and $C(w)$ must differ on at least $\delta_C/2$ fraction of coordinates. Since the copies of E_j form at least one third of the input of ψ_i^{con}, it follows that

$$\operatorname{dist}\left(\left(x \circ \pi'\right)_{|Q_i^{\mathrm{con}}}, \mathrm{SAT}(\psi_i^{\mathrm{con}})\right) \geq \frac{1}{6} \cdot \delta_C,$$

and hence Inequality 5 holds (recall that δ_C is a universal constant that is independent of $\operatorname{dist}(x_{|B_j}, x^{\mathrm{dec}}_{|B_j})$).

This concludes the proof of Inequality 4.

C Lower bound on input size in the proof of the main theorem

Recall that in the proof of the main theorem we had a sequence $\{n_i\}_{i=0}^{\infty}$ defined by $n_{i+1} = n_i^2/d \cdot \log^d n$ for some constant d. In this appendix, we prove that for every i it holds that $n_i \geq \left(n_0/d \cdot 2^d \cdot \log^d n_0\right)^{2^i}$. It is not hard to see that for every $i \geq 1$ it holds that $n_i \leq n_0^{2^i}$. Now, we first prove by induction that for every $i \geq 1$ it holds that

$$n_i \geq n_0^{2^i} / \left(d \cdot \log^d n_0\right)^{2^i - 1} \cdot 1/2^{d \cdot \sum_{j=0}^{i-1} 2^j \cdot (i-1-j)}$$

The induction step goes as follows:

$$\begin{array}{rcl}
n_i & = & n_{i-1}^2 / \left(d \cdot \log^d (n_{i-1})\right) \\
(\text{Since } n_{i-1} \leq n_0^{2^{i-1}}) & \geq & n_{i-1}^2 / \left(2^{d \cdot (i-1)} \cdot d \cdot \log^d (n_0)\right) \\
(\text{The induction hypothesis}) & \geq & \left(n_0^{2^{i-1}} / \left(d \cdot \log^d n_0\right)^{2^{i-1}-1} \cdot 1/2^{d \cdot \sum_{j=0}^{i-2} 2^j \cdot (i-2-j)}\right)^2 \\
 & & \cdot 1/ \left(2^{d \cdot (i-1)} \cdot d \cdot \log^d (n_0)\right) \\
 & = & n_0^{2^i} / \left(d \cdot \log^d n_0\right)^{2^i - 2} \cdot 1/2^{d \cdot \sum_{j=1}^{i-1} 2^j \cdot (i-1-j)} \\
 & & \cdot 1/ \left(2^{d \cdot (i-1)} \cdot d \cdot \log^d (n_0)\right) \\
 & = & n_0^{2^i} / \left(d \cdot \log^d n_0\right)^{2^i - 1} \cdot 1/2^{d \cdot \sum_{j=0}^{i-1} 2^j \cdot (i-1-j)}
\end{array}$$

We now prove that $1/2^{d \cdot \sum_{j=0}^{i-1} 2^j \cdot (i-1-j)} \geq 1/2^{d \cdot 2^i}$, and this will imply the required result. To this end, it suffices to prove that for every $m \geq 0$ it holds that

$$\sum_{j=0}^{m} 2^j \cdot (m-j) \leq 2^{m+1}$$

We first rewrite the sum as follows:

$$\sum_{j=0}^{m} 2^j \cdot (m-j) = \sum_{j=0}^{m} 2^{m-j} \cdot j = 2^{m-1} \sum_{j=0}^{m} j \cdot 2^{-(j-1)}$$

So it remains to show that $\sum_{j=0}^{m} j \cdot 2^{-(j-1)} \leq 4$. Since this should hold for every m, we actually need to show that $\sum_{j=0}^{\infty} j \cdot 2^{-(j-1)} \leq 4$. The latter inequality is folklore, and is proved as follows. Consider the function

$$f(x) = \sum_{j=0}^{\infty} x^j,$$

and at its derivative

$$f'(x) = \sum_{j=0}^{\infty} j \cdot x^{j-1}.$$

Observe that the series that we wish to upper bound is exactly $f'(\frac{1}{2})$. Next, observe that for $x \in (0,1)$, it holds that

$$f(x) = \frac{1}{1-x},$$

and therefore the derivative of f in $(0,1)$ can also be written as

$$f'(x) = \frac{1}{(1-x)^2}.$$

Thus, the series we wish to bound converges to $f'(\frac{1}{2}) = \frac{1}{(1-\frac{1}{2})^2} = 4$, as required.