Short PCPs Verifiable in Polylogarithmic Time*

Eli Ben-Sasson † Oded Goldreich ‡ Prahladh Harsha § Madhu Sudan ¶

Salil Vadhan ‖

January 3, 2006

Abstract

We show that every language in NP has a probabilistically checkable proof of proximity (i.e., proofs asserting that an instance is "close" to a member of the language), where the verifier's running time is polylogarithmic in the input size and the length of the probabilistically checkable proof is only polylogarithmically larger that the length of the classical proof. (Such a verifier can only query polylogarithmically many bits of the input instance and the proof. Thus it needs oracle access to the input as well as the proof, and cannot guarantee that the input is in the language — only that it is close to some string in the language.) If the verifier is restricted further in its query complexity and only allowed q queries, then the proof size blows up by a factor of $2^{(\log n)^{c/q}}$ where the constant c depends only on the language (and is independent of q). Our results thus give efficient (in the sense of running time) versions of the shortest known PCPs, due to Ben-Sasson *et al.* (STOC '04) and Ben-Sasson and Sudan (STOC '05), respectively. The time complexity of the verifier and the size of the proof were the original emphases in the definition of holographic proofs, due to Babai *et al.* (STOC '91), and our work is the first to return to these emphases since their work.

Of technical interest in our proof is a new complete problem for NEXP based on constraint satisfaction problems with very low complexity constraints, and techniques to arithmetize such constraints over fields of small characteristic.

---

*A preliminary version of this paper appeared in *Proc. 20th IEEE Conference on Computational Complexity*, 2005 [BGHSVc]

†Computer Science Department, Technion, Haifa, Israel, and Toyota Technological Institute at Chicago. Email: `eli@eecs.harvard.edu`.

‡Department of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL. Email: `oded.goldreich@weizmann.ac.il`.

§Microsoft Research, 1065 La Avenida, Mountain View, CA 94041. Email: `pharsha@microsoft.com`

¶Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139. Email: `madhu@mit.edu`. Supported in part by NSF Award CCR-0312575.

‖Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Email: `salil@eecs.harvard.edu`. Supported in part by ONR grant N00014-04-1-0478, NSF grant CCR-0133096 and a Sloan Research Fellowship.

[0]Work done when the first, second, fourth and fifth authors were visiting Radcliffe Institute for Advanced Study and the third author was at MIT and the Toyota Technological Institute, Chicago.

Contents

1 Introduction

The study of efficient probabilistic methods for verifying proofs was initiated in the works of Babai *et al.* [BFLS] and Feige *et al.* [FGLSS] with very different motivation and emphases. The work of Babai *et al.* considered the direct motivation of verifying proofs, and especially computations, highly efficiently. Their motivation led them to emphasize the time taken by the verifier and the length of the proof in the new format. In contrast, Feige *et al.* established a dramatic connection between efficient probabilistically checkable proofs (PCPs) and the inapproximability of optimization problems. This led them to focus on the amount of randomness used by the verifier, and the number of bits of the proof that the verifier queries. Most succeeding works have focused on the latter choice of parameters, or variants thereof, and derived many strong inapproximability results for a wide variety of optimization problems (while often introducing improved PCP constructions). In contrast there has been little subsequent work on the parameters highlighted by Babai *et al.* Only a few works, specifically [PS, HS, GS, BSVW, BGHSVb, BSb], have focused on the length of the PCP, while no later work seems to have returned to the question of the extreme efficiency of the verifier. This is unfortunate because the latter efficiency parameters are significant in the context of proof-verification, and are also important in some of the applications of PCPs.

In this work we revisit the study of efficient PCP verifiers. Our work is motivated by two recent developments. First is a technical one: The original result of Babai *et al.* [BFLS] required the PCP to be larger than the classical proof by a factor of $\Omega(n^\epsilon)$ for arbitrarily small but positive ϵ, where n denotes the length of a classical proof. Recent constructions of PCPs have, however, obtained a much smaller proof length. Notably the blowup in the proof-length in the work of Ben-Sasson *et al.* [BGHSVb] is only $2^{(\log n)^\epsilon}$ where the verifier is allowed to query $O(1/\epsilon)$ bits of the proof. And if the verifier is allowed to make more queries, polylogarithmic in the length of the proof, then the blowup in the proof-length is only a polylogarithmic factor (cf. Ben-Sasson and Sudan [BSb]). These improvements in the proof length raise the question as to whether these can be accompanied with efficient verifiers, which would lead to the first strict improvements on the work of [BFLS]; that is, reducing one of the parameters (i.e., the length) without increasing the other (i.e., verification time).

A second motivation to study efficient verifiers is an aesthetic one. To motivate this, we recall the main result of [BFLS].

> There exists a probabilistic verifier that makes oracle access to an "encoded assertion", and a "purported proof", and whose running time is only polylogarithmic in the length of the assertion and its proof, such that the proper encoding of any valid assertion has a proof that is accepted with probability one, while if a supposedly "encoded assertion" is not close to the proper encoding of a valid assertion, then no proof is accepted with probability more than half.

One might contend that the power of this result is somewhat diminished by the technical nature of the statement and in particular the need to encode theorems in error-correcting codes. Such a notion appears necessary due to the sublinear running time of the verifier. However a recent notion, proposed independently by Ben-Sasson *et al.* [BGHSVb] and Dinur and Reingold [DR], suggests a more elegant characterization to capture the power of efficient verifiers — one that is similar to work in property testing [GGR, RS]. We describe this notion, termed "probabilistically checkable proofs of proximity" (by [BGHSVb] and "assignment testers" by [DR]) below.

PCPs of Proximity. A PCP of Proximity verifier accesses a pair of oracles, one for a string x (representing the assertion "$x \in L$") and one for a proof π, and probabilistically produces a Boolean verdict with the property that true assertions x have a proof π that is always accepted while for an assertion x that is far (in relative Hamming distance) from any valid statement has no proof that is accepted with high probability (say greater than half).

On the one hand, PCPs of Proximity do not specify an error-correcting code, making a potential result less cumbersome to state. On the other hand, a universal result of the form, "every language L in NP has efficient PCPs of Proximity", does subsume the result of [BFLS], since such a result includes languages that only contain (a subset of) codewords of an error-correcting code. In principle, the techniques of [BFLS] could be converted to get such a result (i.e., providing a PCP of proximity (for NP) whose running time is polylogarithmic and where the proofs are only $n^{1+\epsilon}$-bits long), but such a statement is not explicit in the previous literature.

Our results. In this work, we derive PCPs of Proximity for every NP language. These PCP systems have *highly efficient* verifiers that match other parameters of some of the best-known PCPs. Specifically, one of our main results (see Theorem 2.5) gives a PCP of Proximity for any language $L \in \mathrm{NTIME}(T(n))$, with $\mathrm{poly}\log(T(n))$ verification time for proofs of length $T(n) \cdot \mathrm{poly}\log(T(n))$. This PCP matches the query complexity and proof length of the system of [BSb], which was proved only for NP and uses a polynomial-time verification procedure. Our second main result focuses on the case where the query complexity of the verifier is further restricted (to, say, a constant) and gives a polylogarithmic time verifier making $O(1/\epsilon)$ queries into a theorem and proof, whose proof length is $T(n) \cdot 2^{(\log T(n))^\epsilon}$, again for verifying membership in $L \in \mathrm{NTIME}(T(n))$. This PCP matches the query complexity and proof length of the system of [BGHSVb], which was proved only for NP and uses a polynomial-time verification procedure. Both results improve over [BFLS] (and [PS]), which obtains proofs of length $T(n)^{1+\epsilon}$, for any constant $\epsilon > 0$.

In terms of the length of the proof, a polylogarithmic factor is perhaps the best one can hope for, given our current inability to get tighter completeness results for non-deterministic computation: E.g., even in a classical reduction to SAT, one loses a polylogarithmic factor in the length of the proof. Thus, our first result achieves this "limit" in the length of the proof, while maintaining the smallest possible running time (i.e., a verifier examining a proof of length $T(n)$ needs at least $\log T(n)$ time to index a random location of the proof). Thus, with respect to the original parameters of [BFLS], our first result achieves limits of qualitative significance. Our second result (which also improves upon [BFLS]) is significant when query complexity is also considered and then it matches the best known PCP constructions, while maintaining efficient verification.

Techniques. Naturally, our efficient PCPs of Proximity are based upon the prior works of [BGHSVb] and [BSb]. However, we stress that efficiency (i.e., fast verification time, let alone polylogarithmic verification time) is not an immediate corollary of having low query complexity. Indeed the FGLSS verifier [FGLSS] invests polynomial time to compute low-degree extensions of its input. The Polishchuk-Spielman verifier [PS] invests polynomial time routing a permutation from n sources to n sinks in a sorting network. And most known PCP constructions use recursive composition, where the time to compose PCPs is lower-bounded by the query complexity of the ingredient PCPs, which can be prohibitively large too. Such operations abound in the recent constructions of PCPs including those of [BGHSVb] and [BSb] leading to several barriers in any immediate translation. These complications force us to tackle some new problems and our solutions yield some new ingredients that may be of independent interest.

First, we give a new problem that is NEXP-complete under nearly linear-time reductions. This problem may be described as a generalized graph coloring problem (color the vertices of a graph subject to arbitrary constraints on the color of a vertex given its name and the colors of its neighbors). We show that it is NEXP hard to color an (exponentially large) deBruijn graph with a constant number of colors, where the coloring constraint function (determining the validity of the coloring of a vertex and its neighbors, depending on the name of the vertex) is described by an extremely low-complexity function; namely, an $\mathbf{NC}_1$ circuit. Moreover, the reduction from an instance of NTIME($T(n)$) yields a deBruijn graph of size $T(n) \cdot \log^{O(1)} T(n)$. Both the construction of such a low-complexity function (of the coloring constraint) and such an efficient reduction may be of independent interest. At a high level, our reduction works by embedding the nearly linear-time oblivious Turing machine simulation of Pippenger and Fischer [PF] on a deBruijn graph.

Next, we describe a general arithmetization technique that converts low-complexity functions into their low-degree extensions that are computable by small algebraic circuits, even when the degree of the extension is very large. This part uses heavily the structure of large finite fields of small characteristic, and may be of independent interest. Applying the arithmetization to our NEXP-complete coloring problem gives a family of nearly linear-time reductions from NTIME(T) to algebraic problems.

Finally, we extend standard notions of compositions to verifiers that are specified implicitly, so as to obtain by composition, efficient verifiers whose running time complexity can be much smaller than the query complexity of some of the ingredients in the composition.

Organization of this paper: In Section 2 we present the main definitions underlying our work, and provide a formal statement of our main results. In Section 3 we provide an overview of the proofs of our main results. The proof themselves appear in the rest of the paper, and their organization is described in Section 3.3.

2 Definitions and Main Results

We follow the general formulation of PCPs of Proximity (PCPPs), as appearing in [BGHSVb, DR]. In this formulation, the input comes in two parts (x, y), where x is given explicitly to the verifier and y is given as an oracle. (In addition, the verifier is given access to a proof oracle.) The verifier is allowed to read x in its entirety, but its queries to y are counted as part of its query complexity (i.e., together with the queries to the proof oracle). Natural special cases, where either $y = \lambda$ or $x = |y|$ (i.e., x is the length of y in binary), will be discussed below.

Definition 2.1 (Restricted PCPP) *Let $r, q : \mathbb{Z}^+ \to \mathbb{Z}^+$ and $t : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$. An (r, q, t)-*restricted PCPP verifier *is a probabilistic machine that, given a string x* (called the explicit input) *and a number K* (in binary) *as well as oracle access to an* implicit input $y \in \Sigma^K$ *and to a* proof oracle $\pi \in \Sigma^*$, *tosses $r(|x| + K)$ coins, queries the oracles (y, π) for a total of $q(|x| + K)$ symbols, runs in time $t(|x|, K)$, and outputs a Boolean verdict in* $\{\texttt{accept}, \texttt{reject}\}$.

We stress that we deviate from the standard treatments in not requiring the verifier to run in polynomial time, but rather considering an explicit time bound, denoted t. Furthermore, this time bound is expressed as a function of two parameters: the length of the explicit part of the input (i.e.,

x) and the length of the implicit part of the input (i.e., y). The reason for separating the effect of the two parts is that, by definition, the verifier must read the entire explicit input x and hence takes time at least linear in its length but it can run for time that is polylogarithmic in $|y| = K$ (just K is needed for reading $|y|$ and indexing into y). In fact, obtaining such running-time is the focus of the current work. Other complexity measures (and parameters) are expressed (as usual) as a function of the sum of these two parts (i.e., the length of the entire input (x, y)). Recall that our interest in the randomness complexity stems from its effect on the *proof length*: the ("effective") length of the proof oracle of a $(r, q, \cdot)$-restricted PCPP verifier is at most $\ell(m) = 2^{r(m)} \cdot q(m)$.

In view of the above, PCPPs refer to languages consisting of pairs of strings (where the elements in these pairs refer to the two parts of the input in Definition 2.1). Thus, we define a pair language to be subset of $\Sigma^* \times \Sigma^*$. For a pair language L and $x \in \Sigma^*$, we define $L_x \triangleq \{y \in \Sigma^* : (x, y) \in L\}$. We usually use the notations $n = |x|$, $K = |y|$ and $m = n + K$.

We will be interested in PCPP verifiers that cannot afford to read their implicit input y in its entirety. Such verifiers will not be able to exactly verify membership of (x, y) in a language L, but will rather test that y is "close" to L_x. Unless stated otherwise, we use the relative Hamming distance as our distance measure between $x, x' \in \Sigma^n$, denoted $\delta(x, x') = |\{i : x_i \neq x'_i\}|/n$. For $x \in \Sigma^n$ and $S \subseteq \Sigma^n$, we define $\delta(x, S) = \min_{x' \in S}\{\delta(x, x')\}$. The string x is said to be δ-far from (resp., δ-close to) S if $\delta(x, S) > \delta$ (resp., $\delta(x, S) \leq \delta$).

Definition 2.2 (PCPP for Pair Languages) *For functions $r, q : \mathbb{Z}^+ \to \mathbb{Z}^+$, $t : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$, $s, \delta : \mathbb{Z}^+ \to [0, 1]$, a pair language $L \subseteq \Sigma^* \times \Sigma^*$ is in $\mathrm{PCPP}_{s,\delta}[r, q, t]$ if there exists an (r, q, t)-restricted verifier V with the following properties:*

- **Completeness:** *If $(x, y) \in L$ then there exists a π such that $\Pr_R[V^{y,\pi}(x, |y|; R)\ \text{accepts}] = 1$, where $V^{y,\pi}(x, |y|; R)$ denotes the decision of V on input $(x, |y|)$, oracle access to (y, π) and coin tosses R.*

- **Soundness:** *If (x, y) is such that y is $\delta(|x| + |y|)$-far from $L_x \cap \Sigma^{|y|}$, then for every π it holds that $\Pr_R[V^{y,\pi}(x, |y|; R)\ \text{accepts}] \leq s(|x| + |y|)$.*

If we specialize Definition 2.2 to pair languages where the implicit input is the empty string λ (and constrain the verifier to polynomial time), then we obtain the standard definition of PCPs.

Definition 2.3 (PCP) *A language L is in $\mathrm{PCP}_s[r, q]$ if there exists a function $t(n, K) = t(n, 0) = n^{O(1)}$ and a constant $\delta < 1$ such that the pair language $L' = L \times \{\lambda\}$ is in $\mathrm{PCPP}_{s,\delta}[r, q, t]$.*

On the other hand, if we specialize Definition 2.2 to pair languages where the explicit input only specifies the length of the implicit input (and constrain the verifier again to polynomial time), then we obtain verifiers that can check, in *polylogarithmic time*, whether a string given as oracle is close to being in some ("pure") language. The special case where this language contains error-correcting encodings of some NP-set was studied in [BFLS]. We generalize their definition as follows:

Definition 2.4 (Efficient PCPP for pure languages) *A language L is in $\text{eff-}\mathrm{PCPP}_{s,\delta}[r, q]$ if there exists a function $t(n, K) = t(0, K) = (\log K)^{O(1)}$ such that the pair language $L' = \{\lambda\} \times L$ is in $\mathrm{PCPP}_{s,\delta}[r, q, t]$.*

More generally, we may define efficient PCPP as ones having a verifier that runs in time polynomial in $|(x,K)|$; that is, having time complexity $t(n,K) = (n \log K)^{O(1)}$.

Recall that most PCP results (only) refer to NP, but many of them can be scaled up to $\mathrm{NTIME}(T)$ for any $T : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that $T(m) < \exp(\mathrm{poly}(n))$. Note that such a scaling requires, as per Definition 2.3, that the verifier run in polynomial-time (rather than in time polynomial in T). The few works [PS, HS, GS, BSVW, BGHSVb, BSb] that focus on the length of the PCP are an exception: their result refers to NP and do *not* extend to $\mathrm{NTIME}(T)$, because the resulting verifiers run in time polynomial in T (rather than polynomial in its input length). Obtaining such (polynomial-time) extensions is the goal of the current paper.

Our results: The first main result of this paper is a PCP verifier for $\mathrm{NTIME}(T)$ with query and randomness complexities analogous to those in [BSb]. Essentially, for every $L \in \mathrm{NTIME}(T)$, where $T(n) < \exp(\mathrm{poly}(n))$, we present a PCP for L using proof length $T(n) \cdot \mathrm{poly}\log T(n)$ verifiable in time $\mathrm{poly}\log T$, generalizing the results in [BSb] which refers to the case $T(n) = \mathrm{poly}(n)$. We stress that our verifier runs in time polynomial in n, $\log K$ and $\log T$, and this should be contrasted with the $\mathrm{poly}(T)$-time verifier implicit in works as [GS, BSVW, BGHSVb, BSb] (as well as in [PS, HS]) which refer explicitly only to the case $T(n) = \mathrm{poly}(n)$. More generally, we have:

Theorem 2.5 (Efficient PCPPs with short proofs) *Suppose that L is a pair language in $\mathrm{NTIME}(T)$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. Then, for every constant $s > 0$, we have $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$, for*

- *Proximity parameter $\delta(m) = 1/\mathrm{poly}\log T(m)$,*
- *Randomness complexity $r(m) = \log_2 T(m) + O(\log\log T(m))$,*
- *Query complexity $q(m) = \mathrm{poly}\log T(m)$,*
- *Verification time $t(n,K) = \mathrm{poly}(n, \log K, \log T(n+K))$.*

In particular, we obtain PCPPs for pure languages in NP (i.e., $L' = \{\lambda\} \times L$) that meet the query and randomness complexities of [BSb], while using a verifier that runs in time that is polylogarithmic in its (implicit) input. Likewise, we obtain PCPs for languages in NEXP, with a randomness and query complexities that generalize the PCPs of [BSb] (which refer only to languages in NP).

Our second main result provides similar efficiency improvements to the PCPs of [BGHSVb]. The proofs are somewhat longer than in Theorem 2.5, but the number of queries is much smaller.

Theorem 2.6 (Efficient PCPPs with small query complexity]) *Suppose that L is a pair language in $\mathrm{NTIME}(T)$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. Then, for every function $\epsilon : \mathbb{Z}^+ \to (0,1)$ such that $\epsilon(m) \geq \log\log\log T(m)/2\log\log T(m)$ and every two constants $s, \delta > 0$, we have $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$, for*

- *Randomness complexity $r(m) = \log_2 T(m) + A_\epsilon(m)$, where $m = n + K$ and $A_\epsilon(m) = O\left(\log T(m)^{2\epsilon(m)}\right) + \left(\frac{1}{\epsilon(m)} + \log T(m)^{\epsilon(m)}\right) \cdot \log\log T(m)$,*
- *Query complexity $q(m) = O(1/\epsilon(m))$,*
- *Verification time $t(n,K) = \mathrm{poly}(n, \log K, \log T(n+K))$.*

In particular, we obtain PCPPs for pure languages in NP (i.e., $L' = \{\lambda\} \times L$) that meet the query and randomness complexities of [BGHSVb], while using a verifier that runs in time that is polylogarithmic in its (implicit) input. Likewise, we obtain PCPs for languages in NEXP, with a randomness–query complexity trade-off that generalize the PCPs of [BGHSVb] (which refer only to languages in NP).

Two special cases of interest are:

1. Letting $\epsilon(m)$ be an arbitrarily small constant, yields query complexity $O(1/\epsilon(m))$ and randomness complexity $\log_2 T(m)+\log^{2\epsilon} T(m)$, which in turn means proof length $T(m)\cdot\exp(\log^{2\epsilon} T(n))$.

2. Setting $\epsilon(m) = \log\log\log T(m)/2\log\log T(m)$, yields query complexity $o(\log\log T(m))$ and randomness complexity is $\log_2 T(m) + o((\log\log T(m))^2)$, which in turn means proof length $T(m) \cdot \exp(o(\log\log T(m))^2)$.

3 Overview of our proofs

Our main results are obtained by applying a common collection of ideas to two previous PCP constructions. Specifically, Theorem 2.5 is obtained by constructing an efficient verifier that is patterned after the verifier of [BSb], while Theorem 2.6 is obtained based on the work of [BGHSVb]. Here we describe the main ideas used to get our improvements.

For starters, we focus on the construction of PCPP for pure languages (i.e., PCPP that only refer to an implicit input, and no explicit input). In both the aforementioned constructions, there is a main construct (a "robust" PCPP) that is composed with itself (double-logarithmically) many times. Two issues arise. The first issue is to obtain such a main construct (i.e., a robust PCPP with adequate query and randomness complexities) that supports polylogarithmic time (rather than polynomial-time) verification. Loosely speaking, this requires a more efficient reduction from NP (or NTIME(T)) to an algebraic constraint satisfaction problem (CSP) (of the type used in [BGHSVb] and [BSb], resp.). In particular, we obtain a succinct representation (of polylogarithmic length) of the constraints. The second issue is the use of the proof composition paradigm in a context (indeed ours) where one cannot afford verification time that is as high as the query complexity of the intermediate verifiers used in the construction. (Needless to say, the verification time will be lower-bounded by the query complexity of the final verifier.) Loosely speaking, addressing this issue requires working with succinct representations of the sequence of queries and decision predicate of the intermediate verifiers. When proving Theorem 2.6 we introduce a general formulation (of so-called "verifier specifications") supporting this process, whereas the proof of Theorem 2.5 capitalizes on properties of the special algebraic problem used in [BSb]. Below, we present more detailed overviews of the two proofs, starting with the proof of Theorem 2.5.

Let us start by justifying our focus on PCPPs for pure languages (or, equivalently, PCPPs without explicit inputs). Recall that our final goal is to obtain PCPPs for general languages (e.g., PCPPs for circuit-value where the circuit is given as explicit input and the assignment is an implicit input). Instead, for sake of simplicity, we wish to carry out the construction when only referring to PCPPs that have no explicit input. We cannot just move the explicit input x to the implicit part (i.e., replace the implicit input y by (x, y)), because this will not maintain the desired guarantees (i.e., that y is close to some $\widehat{y}$ such that $(x, \widehat{y})$ that is in the language since soundness only guarantees that (x, y) is close to some $(\widehat{x}, \widehat{y})$ is in the language, where it may be that $x \neq \widehat{x}$..) Instead, we should incorporate in the implicit input an error correcting encoding of the explicit input, That is, for a language L, rather than verifying that the implicit input y is close to some $\widehat{y}$ such that $(x, \widehat{y}) \in L$, we verify that the implicit input $(\mathrm{ECC}(x), y)$ is close to some $\mathrm{ECC}\widehat{x}, \widehat{y})$ where ECC is an error-correcting code and the two components $\mathrm{ECC}(x)$ and y are given equal weight when measuring Hamming distance. (For details, see Lemma 7.5.)

3.1 Proof of Theorem 2.5

The proof modifies the construction of Ben-Sasson and Sudan [BSb]. We thus start by describing the verifier of [BSb], hereafter referred to as the BS-verifier.

The BS-Verifier. The first step in the construction of the BS-verifier reduces the problem at hand to an instance of a "Constraint Satisfaction Problem on a De-Bruijn graph": that is, a problem where the goal is to color the vertices of a DeBruijn graph such that the coloring of any single vertex is "consistent" with the coloring of its neighbors. Consistency is given by a list of legal values for every neighborhood, and varies from neighborhood to neighborhood. Thus, an instance of the problem is represented by such a sequence of sets (or constraints), where each set represents the legal values for a given vertex and its neighbors. The second step in the construction of the BS-verifier consists of an arithmetization of the DeBruijn-CSP, resulting in a "univariate algebraic CSP": a problem where the goal is to determine if there exists a low-degree univariate polynomial A over a finite field $\mathbb{F}$ such that applying a given "local" operator C to A results in a polynomial $B = C(A)$ that is zero on a prespecified set $H \subseteq \mathbb{F}$. Thus, the operator C specifies an instance of this problem, and is determined from the constraints of the DeBruijn-CSP by a straightforward univariate interpolation. The third step in the construction of the BS-verifier is designing a verifier for the univariate algebraic CSP. A special ingredient in this verifier is a recursive procedure to verify whether a low-degree (univariate) polynomial B, given by a (possibly slightly corrupted) table of its values, is zero on every $\alpha \in H$. This recursive verification constitutes a special-purpose proof composition technique. (It is special-purpose in the sense that it refers to PCPPs for a specific language rather than all of NP.)

It turns out each of these three steps relies on the fact that the resulting BS-verifier is allowed polynomial-time computations. For example, given an instance y (of the original problem), the constraints in the DeBruijn-CSP are determined in $\mathrm{poly}(|y|)$-time, and each constraint depends on the entire y. Seeking polylogarithmic verification time, we need to find an alternative reduction and an adequate arithmetization.

Getting an efficient BS-type verifier. Recall that given an implicit input y and we need to verify membership in some (adequate universal) language L. Referring to the first step in the construction, we wish to transform y into a succinct representation of an instance of DeBruijn-CSP, but need to do so without knowing the entire y. In such a succinct DeBruijn-CSP, the constraint associated with a vertex v (i.e., placing conditions on the coloring of v and its neighbors) can be computed in time $\mathrm{poly}(|v|)$ possibly using oracle access to y. Furthermore, for subsequent arithmetization, even such efficient computation is not sufficient; we require the constraint to be computed extremely efficiently, e.g., by an $\mathbf{NC}_1$ circuit (applied to the vertex name). To this end, we define Succinct DeBruijn-CSP (see Definition 4.3 (where we use the term generalized coloring)) in a way that makes such efficient computation a requirement; and reduce the universal problem to this problem (see Theorem 4.4). This reduction revisits a classical reduction of general Turing machine computations to Turing machine computations on oblivious machines due to Pippenger and Fischer [PF]. This replaces the first step in the BS-construction.

Next we jump to the *third step* of the BS-construction, namely of verifying that a univariate polynomial B, given by a (slightly corrupted) table of the associated function $B : \mathbb{F} \to \mathbb{F}$, is zero on a given set $S \subseteq \mathbb{F}$. In order to perform this verification, the BS-verifier considers the polynomial $Z_S(x) \triangleq \prod_{e \in S}(x - e)$, and evaluates $Z_S(r)$ at a random $r \in \mathbb{F}$. The BS-verifier performs this computation in the straightforward way, taking $O(|S|)$ field operations, which turns out to be polynomially related to the length of the (implicit) input (i.e., y). For our purposes such running-time is too expensive; recall, we need a verifier running in polylogarithmic (in $|y|$) time. In particular, we wish to evaluate Z_S in $\mathrm{poly}\log|\mathbb{F}|$ time. To this end we exploit the fact that we (as designers of the PCP) have (almost full) control on the choice of the set S for which the verifier needs to evaluate the polynomial Z_S. We now use the fact that if $\mathbb{F}$ has small (i.e., constant) characteristic (e.g., two), and S is a linear subspace of $\mathbb{F}$ (where we view $\mathbb{F}$ as a vector space), then the polynomial Z_S is $\log|S|$-sparse (i.e., has only $\log|S|$ terms) and thus Z_S can be evaluated in $\mathrm{poly}\log|S|$ field operations. (The relevant algebraic facts are described and proved in Section 5.1.) We mention that the computational advantages of working with linear subspaces of finite fields is one of the main contributions of this work (even though the underlying algebraic facts are well-known and were used, though not computationally, in [BSb, BGHSVb]).

Finally we move to the *second step* of the BS-construction, where we transform DeBruijn-graph CSP to (univariate) algebraic CSPs. It is shown in [BSb] how to embed[1] the DeBruijn graph into a Cayley-like graph over any sufficiently large field of characteristic two (where the vertices of the Cayley-like graph are elements of the field and adjacency is given by a constant number of affine functions over the field). We use the same embedding, and arithmetize the constraint function over the same embedding. For this part, we need to transform the "constraint" function C, where $C(v, \cdots)$ describes the constraint on the neighborhood of the vertex v, into a polynomial of moderately low-degree that can be computed by a very small circuit over $\mathbb{F}$. More specifically, if we let $S \subseteq \mathbb{F}$ be the image of the embedding, then we would like the polynomials to have degree $\tilde{O}(|S|)$, while the size of the circuits should be $\mathrm{poly}\log|S|$. This is a non-trivial challenge, since all we know about the function C is that it is a small depth circuit when its input is viewed as a sequence of bits, whereas now we want to view the input as an element of $S \subseteq \mathbb{F}$ and perform only $\mathbb{F}$ operations on it.

Once again we bring in the fact that S is selected to be a linear subspace of $\mathbb{F}$. We also use the fact that the bits of the natural representations of $v \in S$ are projection functions, which in turn are linear maps of S to $\mathbb{F}$. We prove and use the fact that, when S is a linear subspace of $\mathbb{F}$, any linear map $f : S \to \mathbb{F}$ can be represented by a $(\log|S|)$-sparse polynomial $\widehat{f} : \mathbb{F} \to \mathbb{F}$ of degree $|S|$ that extends f (see Proposition 5.1). This implies that any bit in the natural representation of $v \in S$ can be computed efficiently by a small algebraic circuit of low-degree. We conclude that any small-depth small-size circuit can be arithmetized naturally to get a small-degree small-algebraic circuit (see Theorem 5.5). Thus, we get a low-degree polynomial that is computed by a small algebraic circuit that represents, for every $v \in S$, the constraint associated with v's neighborhood.

[1] The notion of embedding used here is that of an injective homomorphism, where a vertex u is mapped to $f(u)$ such that the existence of a directed edge $u \to v$ in the image graph implies that $f(u) \to f(v)$ is an edge of the graph used for embedding. Note that f is not necessarily surjective and that non-edges need not map to non-edges.

3.2 Proof of Theorem 2.6

The proof modifies the construction of Ben-Sasson *et al.* [BGHSVb]. We thus start by describing the verifier of [BGHSVb], hereafter referred to as the BGHSV-verifier. Recall that the BGHSV-verifier has lower query complexity than the BS-verifier, though it utilizes slightly longer proofs. These features are inherited by Theorem 2.6 (as compared to Theorem 2.5).

The BGHSV verifier. The BGHSV-verifier is built by repeated composition of an atomic verifier, which we'll call the Basic-Verifier. The ingredients going into the Basic-Verifier are similar to the ingredients of the BS-verifier: i.e., there is a reduction of SAT to DeBruijn-CSP; a reduction of DeBruijn-CSP to an algebraic problem (though this time, the reduction is to problems involving multivariate polynomials), and finally a construction of a (robust) PCPP verifier for the algebraic problem.

Getting an efficient BGHSV-type verifier. Employing similar (and sometimes, the same) ideas as those described in Section 3.1, we can improve the running time of this verifier also, and make it comparable to its query complexity. Unfortunately this falls (well) short of our goals of polylogarithmic time verification. This is because we later employ composition to reduce the query complexity of the PCP system. However the query complexity at the Basic-verifier could be as large as $\sqrt{K}$, for theorems of size K. The problem is that composition does not reduce the running time of the composed verifier: the running time of the composed verifier is the sum of the running times of the ingredient verifiers.

Thus the main additional challenge in reducing the time-complexity of the BGHSV-verifier is in redesigning the ingredients of PCP composition such that the verifiers used in composition have significantly smaller running times than their query complexity! We do so in the usual spirit of "implicit" computations: Rather than building a circuit that describes the computations of the Basic-verifier, we describe its computations by Turing machines. Rather than listing all the queries that the Basic-verifier would make, we describe a function that when given an index i, returns the i^{th} query that the verifier would make (if allowed to run fully). Put together this gives a specification of a verifier rather than the actual verifier itself. This idea of a verifier specification is defined formally in Section 7 (see Definition 7.1). We then describe how composition works for verifier specifications (see Lemma 7.3). Finally, we show how to construct an adequate verifier specification based on the techniques described above. Combining all of these gives a proof of Theorem 2.6.

3.3 Organization of the presentation of the proofs

In Section 4 we show how to reduce any language in NTIME(T), for $T(m) \le \exp(\text{poly}(m))$, to a generalized coloring problem referring de-Brujin graphs. (In the above overview, we have referred to this generalized coloring problem as to a Constraint Satisfaction Problem, where constraints are applied only to local neighborhoods consisting of a vertex and its neighbors.) In Section 6, following adequate algebraic complexity preliminaries presented in Section 5, we present arithmetizations of the generalized de-Brujin coloring problem. Specifically, Section 6.2 presents an Multivariate arithmetization, which is used in the proof of Theorem 2.6, whereas Section 6.1 presents an Univariate arithmetization, which is used in the proof of Theorem 2.5. The complexity bounds for the arithmetizations follow from facts that are proven in Section 5. As mentioned in the overview, proving

Theorem 2.6 requires a general proof composition technique that supports implicit specifications of verifiers.

4 A Universal Graph Coloring Problem

Our universal problem refers to a family of graphs that are related to deBruijn graphs. Let $\oplus$ denote the bitwise exclusive-or operation. We use commas to denote concatenation of binary strings.

Definition 4.1 *The* extended deBruijn graph $\mathcal{DB}_{k,l}$ *is a directed graph with l layers each containing 2^k nodes, which are represented by k-bit strings. The layers are numbered $0, 1, \ldots, l-1$. The node represented by $v = (b_0, \ldots, b_{i^*}, \ldots, b_{k-1})$ in layer i has edges pointing to the nodes represented by $\Gamma_{i,0}(v) = (b_0, \ldots, b_{i^*}, \ldots, b_{k-1})$ and $\Gamma_{i,1}(v) = (b_0, \ldots, b_{i^*} \oplus 1, \ldots, b_{k-1})$ in layer $i+1 \bmod l$, where $i^* \triangleq i \bmod k$.*

Let M be any fixed Turing machine (later we will fix M to be the universal Turing machine described in Lemma 7.5 . The bounded halting problem BH_M for machine M, is defined as below.

Definition 4.2 (Bounded Halting Problem) *The bounded halting problem for the Turing machine M, indicated by BH_M has instances of the form (y, t) where y is an instance of the language recognised by M and t is any positive integer. The instance $(y, t) \in \mathrm{BH}_M$ iff the machine M accepts the instance y within 2^t steps.*

We show how to reduce the bounded-halting problem for M to the following constraint satisfaction problem on (extended) deBruijn graphs. Following [VL], we actually prefer to present the local constraints as a generalized coloring problem.[2]

Definition 4.3 (Generalized deBruijn Graph Coloring) *The problem is defined with respect to the infinite family of extended deBruijn graphs, $\{\mathcal{G}_t = \mathcal{DB}_{t+3,(t+3)^2} = (V_t, E_t)\}_{t\in\mathbb{N}}$, from Definition 4.1, and is parameterized by five* (fixed) *finite objects:*

1. *a finite color-set $\mathcal{C}$,*
2. *an extraction function $f_{\mathsf{extract}} : \mathcal{C} \to \{0,1\} \cup \{\perp\}$,*
3. *a finite vertex-type set $\mathcal{V}$,*
4. *a type-coloring constraint $f_{\mathsf{color}} : \mathcal{V} \times \mathcal{C}^3 \to \{0,1\}$, and*
5. *a uniform $\mathbf{NC}_1$ family of type-assignments $f_{\mathsf{v-type}} = \{f_{\mathsf{v-type}}{}^{(t)} : V_t \to \mathcal{V}\}_{t\in\mathbb{N}}$; (that is, a logspace machine that on input 1^t, outputs a boolean formula computing $f_{\mathsf{v-type}}{}^{(t)}$),*

Given an input $y \in \{0,1\}^*$, the problem is to determine whether, for $t = \lceil \log_2 |y| \rceil$, there exists a coloring $C : V_t \to \mathcal{C}$ such that the following two conditions hold

1. *for all vertices v in $\mathcal{G}_t$, we have $f_{\mathsf{color}}(f_{\mathsf{v-type}}(v), C(v), C(\Gamma_{i,0}(v)), C(\Gamma_{i,1}(v))) = 0$.*

   In such a case, we say that C satisfies the coloring constraints*, which are induced by $f_{\mathsf{v-type}}$ and f_{color}.*

[2] We mention that the generalized coloring problem used in [VL] was defined over the grid, and its universality followed by a reduction from the standard tableau associated with the computation of a TM. On the other hand, in [VL] only one type of vertices is used (i.e., $|\mathcal{V}| = 1$).

2. *for all* $1 \le i \le K = |y|$*, we have* $y_i = f_{\textsf{extract}}(C(v_i))$*, where* v_i *is vertex number* $2 \cdot 2^t + i$ *in layer 0 of* $\mathcal{G}_t$*.*

   In such a case, we say that C is consistent with the input (or with y).

We show that for any Turing machine M, there exist a setting of the five finite parameters of the Generalized deBruijn Graph Coloring problem that makes it universal. That is:

Theorem 4.4 (Universality of Generalized deBruijn Graph Coloring) *For any Turing machine* M*, there exist finite parameters* $\mathcal{C}$*,* $f_{\textsf{extract}}$*,* $\mathcal{V}$*,* $f_{\textsf{color}}$ *and* $f_{\textsf{v-type}}$*, such that the bounded-halting problem for* M *is reducible via the identity mapping*[3] *to the corresponding Generalized deBruijn Graph Coloring problem. Furthermore, for every* $y \in \{0,1\}^K$*, where* $K < 2^t$*, machine* M *halts on* y *within* 2^t *steps if and only if there exists a coloring* $C : V_t \to \mathcal{C}$ *that satisfies the coloring constraints and is consistent with* y*.*

Note that the size of $\mathcal{G}_t$ is $(t+3)^2 \cdot 2^{t+3}$, which is $O(T \cdot \log^2 T)$, where $T = 2^t$ bounds the running time of M. Theorem 4.4 is stronger than the related result of Polishchuk and Spielman [PS] in the sense that the coloring problem uses a fixed set of coloring constraints, which can be generated very efficiently (i.e., it admits a succinct description by uniform $\mathbf{NC}_1$ circuits). In contrast, the reduction of [PS] uses constraints that are computed in time $\text{poly}(T)$ based on the original instance.

Our proof of Theorem 4.4 combines the ideas of [PS] with the oblivious TM simulation of Pippenger and Fischer [PF]. Actually, it is simpler to bypass oblivious Turing machines and rather just show how to describe valid computations of M in a recursive manner (which is indeed the basis for the oblivious simulation). For simplicity, we only show how to do this for a 1-tape TM, but the proof is easily extended to handle multi-tape TMs (which is needed, because we will eventually work with a 2-tape universal Turing machine).

Proof of Theorem 4.4: Unlike [PS], we reduce from the computation of a (universal) Turing Machine rather than from a computation of a non-uniform circuit (which is given as explicit input in [PS]). Indeed, we can emulate a TM computation by a circuit. Considering the straightfoward circuit that emulates the computation of the TM, we obtain a quadratic blow-up, which will propagate to the size of the graph on which we "route" this circuit (following [PS]). But, as is well-known (e.g., [PF]), there exists a more efficient emulation; that is, a T-step TM computation can be emulated by a $(T \log T)$-size circuit, which in turn can be routed on a graph of size $\tilde{O}(T)$. The problem is that we need to find the relevant routing (which will be encoded as the coloring constraints) very efficiently — it should be described by a uniform $\mathbf{NC}_1$ circuit of size $\text{poly}\log T$ (c.f., Definition 4.3). In a sense, manage to do so by capitalizing on the simple structure of the emulation in [PF]. however, it turns out to be more convenient to take a direct approach and work with TM computations, rather than reducing back and forth to the evaluation of some highly-uniform circuits.

Say M has state set Q, containing a start state q_{start} and an accept state q_{accept}, tape alphabet $\Gamma = \{0,1\} \cup \{\sqcup\}$, and transition function $\delta_M : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$. Assuming that $\perp \notin Q$, we represent configurations of machine M by sequences over $\Lambda = \Gamma \times (\{\perp\} \cup Q)$, where each symbol in Λ represents a tape symbol, and indicates whether or not the head of M is in that position, and if so the state of M. A (partial) configuration of length L is a function $\sigma : [0, L-1] \to \Lambda$, representing

[3]We say that a problem A is reducible to another problem B via the identity mapping if the following holds: For every instance x of the problem A, x is an YES-instance (similarly NO-instance) of A iff x is an YES-instance (NO-instance) of B.

a L-cell window of M's computation. Specifically, $\sigma(i) = (\sigma_s(i), \sigma_q(i))$ says that $\sigma_s(i) \in \Gamma$ is the tape symbol in the i^{th} cell of the window, $\sigma_q(i) = \perp$ indicates that the head of M is not in the i^{th} cell, whereas $\sigma_q(i) \neq \perp$ indicates that the head of M is in the i^{th} cell and that state of M is $\sigma_q(i)$. We say that σ is a valid configuration if there is exactly one cell i such that the second component of $\sigma(i)$ is in Q, and we denote this cell by $\texttt{head}(\sigma) = i$. For configurations σ and σ' both of length L, we call σ' the successor of σ if σ' is obtained from σ by one step of M and this step does not move the head outside the specified window. In particular, if $0 < \texttt{head}(\sigma) < L$, then σ has a (unique) successor. Similarly, σ' is the t^{th} successor of σ if σ' is obtained from σ by t steps of M, none of which leave the tape window of length L.

We say that a valid $8L$-symbol long configuration σ is safe if the head is in the "middle half" of σ; that is, $\texttt{head}(\sigma) \in [2L, 6L-1]$. A triple of $8L$-symbol long configurations $(\sigma_I, \sigma_M, \sigma_F)$ (where the subscripts stand for "initial," "middle," and "final") is defined to be good if σ_I is a safe configuration of length $8L$, configuration σ_M is the L^{th} successor of σ_I, and σ_F is the L^{th} successor of σ_M. (Note that the fact that σ_I's head is in the middle half implies that the head cannot move outside the window within $2L$ steps, so σ_M and σ_F are well-defined.)

The key observation underlying the Pippenger–Fischer oblivious TM simulation is that computations of $2L$ steps in a window of size $8L$ can be recursively simulated by two computations of L steps each in windows of size $4L$. The following lemma formulates this idea in a way convenient for our purposes.

Lemma 4.5 (implicit in [PF]) *Let $(\sigma_I, \sigma_M, \sigma_F)$ be a triple of $8L$-symbol long configurations, and suppose that σ_I is* safe*. The triple $(\sigma_I, \sigma_M, \sigma_F)$ is* good *if and only if there exist numbers $h, h' \in [0, 4]$ and two* good *triples of $4L$-symbol long configurations, $(\sigma'_I, \sigma'_M, \sigma'_F)$ and $(\sigma''_I, \sigma''_M, \sigma''_F)$, such that the following holds:*

1. *For every $i \in [0, 4L-1]$,*
   - $\sigma'_I(i) = \sigma_I(h \cdot L + i)$,
   - $\sigma'_F(i) = \sigma_M(h \cdot L + i)$,
   - $\sigma''_I(i) = \sigma_M(h' \cdot L + i)$, *and*
   - $\sigma''_F(i) = \sigma_F(h' \cdot L + i)$.
2. *$\sigma_M(i) = \sigma_I(i)$ for every $i \in [0, 8L-1] \setminus [h \cdot L, h \cdot L + 4L - 1]$, and*
3. *$\sigma_F(i) = \sigma_M(i)$ for every $i \in [0, 8L-1] \setminus [h' \cdot L, h' \cdot L + 4L - 1]$.*

The forward direction of the the lemma implies that during the first L steps succeeding σ_I the head remains in the interval $[hL, hL + 4L]$, and the during the next L steps it remains in the interval $[h'L, h'L + 4L]$.

Proof Sketch: For the forward direction we set $h = \lfloor \texttt{head}(\sigma_I)/L \rfloor - 2$, implying that $h \in [0, 3]$ (because $\texttt{head}(\sigma_I) \in [2L, 6L-1]$) and $\langle \sigma'_I(i) \rangle_{i=0}^{4L-1} = \langle \sigma_I(h \cdot L + i) \rangle_{i=0}^{4L-1}$ is safe (because $\texttt{head}(\sigma_I) \in [hL+2L, hL+3L-1]$ and so $\texttt{head}(\sigma'_I) \in [2L, 3L-1] \subset [L, 3L-1]$). Let $b' = \lfloor \texttt{head}(\sigma_M)/L \rfloor \in [1, 6]$ (because $\texttt{head}(\sigma_M) \in [L, 7L-1]$), and set $h' = b' - 1$ if $b' < 5$ and $h' = b' - 2$ otherwise. Thus, $h' \in [0, 4]$ and $\langle \sigma''_I(i) \rangle_{i=0}^{4L-1} = \langle \sigma_M(h' \cdot L + i) \rangle_{i=0}^{4L-1}$ is safe (because $\texttt{head}(\sigma_M) \in [b'L, b'L + L - 1] \subset [h'L+L, h'L+3L-1]$ and so $\texttt{head}(\sigma''_I) \in [L, 3L-1]$). It follows that $(\sigma'_I, \sigma'_M, \sigma'_F)$ and $(\sigma''_I, \sigma''_M, \sigma''_F)$ are good. The opposite direction is straightforward. ∎

We now use this to express M's acceptance criterion as a constraint satisfaction problem (using an instance of size $\tilde{O}(2^t)$ to encode 2^t steps). Our focus is on the simplicity of the rule determining the unknowns that are considered in each constraint.

In the following lemma, $\mathbb{T}_0$ represents the initial contents of M's tape as well as its contents after 2^t and $2 \cdot 2^t$ steps. The other $\mathbb{T}_i$'s represent numerous partial configurations that arise in the computation. Specifically, $\mathbb{T}_i(j, \cdot)$ represents some window of length 2^{t-i} after $j \cdot 2^{t-i}$ (as well as after $(j+1) \cdot 2^{t-i}$ and $(j+2) \cdot 2^{t-i}$) computation steps. The correspondances between the various $\mathbb{T}_i$'s are given by the functions $\mathbb{H}_i$'s that correspond to the h's used in Lemma 4.5. The fixed Boolean functions ψ_0, ψ_1, ψ_2 and ψ_3 will capture the straightforward conditions that should hold for the aforementioned variable functions to encode a possible computation of M.

Lemma 4.6 (Reduction to a CSP) *For every Turing machine M, there exist fixed functions $\psi_0, \psi_1 : \Lambda^{(5+1)\cdot 3} \times [0,4]^2 \to \{0,1\}$, and $\psi_2 : \Lambda^3 \times [0,4]^2 \to \{0,1\}$ and $\psi_3 : \Lambda^{8\cdot 3} \to \{0,1\}$ such that for every $t \in \mathbb{N}$ and $K < 2^t$, machine M accepts the input $y \in \{0,1\}^K$ within 2^t steps if and only if there exist functions $\mathbb{T}_0, \ldots, \mathbb{T}_t$ and $\mathbb{H}_0, \ldots, \mathbb{H}_t$ satisfying the following conditions:*

1. $\mathbb{T}_i : \{0, ..., 2^i - 1\} \times \{0, ..., 8 \cdot 2^{t-i} - 1\} \to \Lambda^3$,*for every* $i = 0, \ldots, t$.

   The 3 components will be indexed by I, M and F.

2. $\mathbb{H}_i : \{0, ..., 2^i - 1\} \to [0,4]^2$, *for every* $i = 0, \ldots, t$.

3. $\mathbb{T}_0(0, \cdot)_I$ *encodes the initial configuration with input y. That is,* $\mathbb{T}_0(0, 2 \cdot 2^t)_I = (\sqcup, q_{\text{start}})$, $\mathbb{T}_0(0, 2 \cdot 2^t + k)_I = (y_k, \perp)$ *for* $k = 1, \ldots, K$, *and* $\mathbb{T}_0(0,k)_I = (\sqcup, \perp)$ *for all other values of* k.

4. $\mathbb{T}_0(0, \cdot)_F$ *encodes an accepting configuration. That is,* $\mathbb{T}_0(0, 2 \cdot 2^t)_F = (\cdot, q_{\text{accept}})$.

5. *For every* $i = 0, \ldots, t-1$, $j = 0, \ldots, 2^i - 1$ *and* $k = 0, \ldots, 4 \cdot 2^{t-i} - 1$,

$$\psi_0\Big(\langle \mathbb{T}_i(j, k + h \cdot 2^{t-i}) \rangle_{h \in [0,4]}, \mathbb{T}_{i+1}(2j, k), \mathbb{H}_i(j) \Big) = 1,$$

   and

$$\psi_1\Big(\langle \mathbb{T}_i(j, k + h \cdot 2^{t-i}) \rangle_{h \in [0,4]}, \mathbb{T}_{i+1}(2j+1, k), \mathbb{H}_i(j) \Big) = 1.$$

   In other words, For every $i = 0, \ldots, t-1$ and $j = 0, \ldots, 2^i - 1$, the functions $\mathbb{T}_i(j, \cdot)_I$ and $\mathbb{T}_i(j, \cdot)_M$ fit $\mathbb{T}_{i+1}(2j, \cdot)_I$ and $\mathbb{T}_{i+1}(2j, \cdot)_F$, whereas $\mathbb{T}_i(j, \cdot)_M$ and $\mathbb{T}_i(j, \cdot)_F$ fit $\mathbb{T}_{i+1}(2j+1, \cdot)_I$ and $\mathbb{T}_{i+1}(2j, \cdot)_F$, where the fitting is with respect to adequate shifts in $\mathbb{T}_i$, which in turn are given by $\mathbb{H}_i$.

6. *For every $i = 0, \ldots, t-1$ and $j = 0, \ldots, 2^i - 1$, the* "unfitted" *portions of $\mathbb{T}_i(j, \cdot)$ remain unchanged. That is, for every $i = 0, \ldots, t-1$, $j = 0, \ldots, 2^i - 1$, and $k = 0, \ldots, 8 \cdot 2^{t-i} - 1$,*

$$\psi_2(\mathbb{T}_i(j,k), \mathbb{H}_i(j)) = 1.$$

7. $\mathbb{T}_t(j, \cdot)$ *encodes single computation steps of M. That is, for every $i = 0, 2^t - 1$, it holds that* $\psi_3(\{\mathbb{T}_t(x,k) : k \in [0,7]\}) = 1$.

Note that Lemma 4.6 asserts a reduction, via the identity transformation, from the Bounded Halting (BH_M) problem (of M) to a Constraint Satisfaction Problem (CSP). Indeed, the instance (y, t) of BH_M (representing the question of whether M accepts y within 2^t steps) is mapped to the instance (y, t) of CSP (representing the question of whether there exist functions that satisfy some set of conditions that depend on t and y).

Proof Sketch: For any $i \in \{0, ..., t-1\}$ and $j \in \{0, ..., 2^i - 1\}$, the triple $\mathbb{T}_i(x, \cdot)$ represents a triple of $8 \cdot 2^{t-i}$-long configurations, denoted $(\sigma_I, \sigma_M, \sigma_F)$, whereas $\mathbb{T}_{i+1}(2j, \cdot)$ and $\mathbb{T}_{i+1}(2j+1, \cdot)$ represent the two "children" configurations as in Lemma 4.5, with $\mathbb{H}_i(j)$ representing the pair (h, h') (asserted in Lemma 4.5). (Indeed, as indicated below, $\mathbb{T}_0(\cdot)_I$ is safe.)

Item 3 specifies that the initial configuration of length $8 \cdot 2^t$ has the TM head in the start state on tape position $2 \cdot 2^t$ followed by the input y, and the rest of the tape has blank cells. (Thus, $\mathbb{T}_0(\cdot)_I$ is safe.) Item 4 specifies that the final configuration has the TM head in the accept state (on tape position $2 \cdot 2^t$). Note that there is a $\mathbb{T}_0$ that satisfies these two constraints and constitutes a *good* configuration-triple if and only if M accepts y within 2^t steps.

Items 5 and 6 (when coupled with suitable functions ψ_0, ψ_1 and ψ_2) capture the recursive characterization of a good configuration-triple $\mathbb{T}_i(j, \cdot)$ in terms of its "children" $\mathbb{T}_{i+1}(2j, \cdot)$ and $\mathbb{T}_{i+1}(2j+1, \cdot)$, as given in Lemma 4.5 (using the pair $(h, h') = \mathbb{H}_i(x)$). Item 7 (when coupled with a suitable function ψ_3) verifies that each leaf (in the recursion tree) $\mathbb{T}_t(j, \cdot) = (\sigma_I, \sigma_M, \sigma_F)$ constitutes a good configuration-triple, which in turn means that σ_M (resp., σ_F) is the (direct) successor of σ_I (resp., σ_M). Indeed, ψ_3 depends on the transition function of the TM M. ■

We now show how the above CSP can be embedded in a extended deBruijn graph, establishing Theorem 4.4. Recall that the extended deBruijn graph has $(t+3)^2$ layers, each with $8 \cdot 2^t$ vertices represented by bit-strings in $\{0,1\}^{t+3}$. We will only use the first $(t+1) \cdot (t+3)$ layers, which we will view as being numbered $(0, 0), \ldots, (0, t+2), \ldots, (t, 0), \ldots, (t, t+2)$. We will interpret a coloring of vertex in layer $(i, 0)$ as giving the functions $\mathbb{T}_i$ and $\mathbb{H}_i$. Specifically, viewing a vertex $v \in \{0,1\}^{t+3}$ in layer $(i, 0)$ as a pair (j, k), where $j \in \{0, ..., 2^i - 1\}$ and $k \in \{0, ..., 8 \cdot 2^{t-i} - 1\}$, we interpret v's color as a pair $(\mathbb{T}_i(j, k), \mathbb{H}_i(j, k)) \in \Lambda^3 \times [0, 4]^2$. Note, however, that in Lemma 4.6, the function $\mathbb{H}_i$ only depends on j. Thus, in addition to the conditions listed in Lemma 4.6, we will need to enforce the condition $\mathbb{H}_i(j, k) = \mathbb{H}_i(j, k')$ for all k, k'. To enforce all of these conditions, we will use the intermediate layers between layer $(i, 0)$ and layer $(i+1, 0)$. Specifically, we will route information between layer $(i, 0)$ and layer $(i+1, 0)$, using easily constructible routes. We will use the coloring constraints to guarantee proper routing of information (through the intermediate vertices) as well as to enforce the conditions listed in Lemma 4.6 (at the end vertices, i.e., at layers $(\cdot, 0)$). Actually, to allow for this routing, we use a larger set of color such that each intermediate vertex is colored by a $O(1)$-long sequence of "basic colors" (i.e., of the type used for vertices at layers $(\cdot, 0)$). Indeed, the coloring constraints will depend on the vertex, and typically most conspiciously on the layer of the vertex.

Lemma 4.7 (Reducing CSP to deBrujin Coloring) *For every quadruple of functions $(\psi_0, \psi_1, \psi_2, \psi_3)$, there exist finite parameters $\mathcal{C}$, f_{extract}, $\mathcal{V}$, f_{color} and $f_{\mathsf{v-type}}$, such that the following holds: For every $t \in \mathbb{N}$, $K < 2^t$, and $y \in \{0,1\}^K$ there exist functions $\mathbb{T}_0, \ldots, \mathbb{T}_t$ and $\mathbb{H}_0, \ldots, \mathbb{H}_t$ satisfying the conditions of Lemma 4.6 if and only if there exists a coloring $C : V_t \to \mathcal{C}$ that satisfies the coloring constraints and is consistent with y.*

Lemma 4.7 reduces the CSP (of Lemma 4.6) to the Generalized deBruijn Graph Coloring. Again, the reduction is by the identity mapping (applied to the instance (y, t)).

Proof Sketch: Referring to the aforementioned correspondance between colors and the functions $\mathbb{T}_0, ..., \mathbb{T}_t$ and $\mathbb{H}_0, ..., \mathbb{H}_0$, we need to show how the parameters of the coloring problem can enforce the conditions of the CSP. Recall that we need to deal with conditions of three types:

1. The CSP conditions listed in Lemma 4.6, which typically refer to $\mathbb{T}_i(j, k + h \cdot 2^{t-i})$ and $\mathbb{T}_i(2j + h, k)$ for a constant number of values of h as well as to $\mathbb{H}_i(j, k)$.

2. The auxiliary conditions $\mathbb{H}_i(j, k) = \mathbb{H}_i(j, k')$, for all j, k, k'.

To enforce all of these conditions, we will use the intermediate layers between layer $(i, 0)$ and layer $(i+1, 0)$. Specifically, we will "route" the $(\mathbb{T}_i, \mathbb{H}_i)$ values assigned to vertices in layer $(i, 0)$ through these intermediate layers so that each of the constraints can be checked by a coloring condition that refers to the values that reach each vertex of layer $(i + 1, 0)$: For each vertex $v = (j, k)$ in layer $(i + 1, 0)$ we check (via a coloring condition) its value (i.e., color) against the values assigned to vertices $(\lfloor j/2 \rfloor, k), ..., (\lfloor j/2 \rfloor, k + 4 \cdot 2^{t-i})$ of layer $(i, 0)$. This is done by ensuring that only a constant number of values are routed through any intermediate vertex and that the routing is simple enough. It suffices to consider each of the above five required routings seperately. That is, for some $h \in [0, 4]$, we wish to determine a set of simple routes such that each vertex $v = (j, k)$ of layer $(i + 1, 0)$ is reached from the corresponding vertex $u = (\lfloor j/2 \rfloor, k + h \cdot 2^{t-i})$ of layer $(i, 0)$.

Let us take a closer look at these two (generic) vertices. Denoting by $\mathrm{bin}_\ell(q)$ the ℓ-bit long binary representation of the integer $q \in \{0, ..., 2^\ell - 1\}$, we observe that $v = \mathrm{bin}_{i+1}(j)\mathrm{bin}_{t-i+2}(k)$ whereas $u = \mathrm{bin}_i(\lfloor j/2 \rfloor)\mathrm{bin}_{t-i+3}(k + h \cdot 2^{t-i})$, where $j \in \{0, ..., 2^{i+1} - 1\}$ and $k \in \{0, ..., 2^{t-(i+1)} - 1\}$. Thus, letting $\alpha = \mathrm{bin}_i(\lfloor j/2 \rfloor)$ and $\tau = j \mod 2$, we have $v = \alpha\tau 0^3 \mathrm{bin}_{t-(i+1)}(k)$ and $u = \alpha \mathrm{bin}_3(h)\mathrm{bin}_{t-(i+1)}(k)$. We infer that v and u differ only in a constant number of positions, and furthermore these positions are easy to determine (because they depend merely on h and the $(i+1)^{\mathrm{st}}$ most significant bit of v, which is the least significant bit of j). Thus for any n, there exists a constant number of patterns $p \in \{0, 1\}^{t+3}$ (in our case two) such that routing each u to $u \oplus p$, for each pattern p, will serve all routes we need. We note that for each destination vertex, only one of the two incoming routes will be relevant. But we can ignore the information coming from the irrelevant route. Determining which route is relevant can be done by just looking at the $(i + 1)^{\mathrm{th}}$ most significant bit of v and thus, certainly by a uniform $\mathbf{NC}_1$ of the vertex name $(i + 1, v)$. The destination vertex (of layer $(i + 1, 0)$) also needs to apply the relevant ψ_q's (from Lemma 4.6) to the values contained in the colors available to it. This can be done by a fixed coloring constraint (which will be applied at vertices of layer $(i + 1, 0)$).

For any fixed p, routing u of layer $(i, 0)$ to $u \oplus p$ of layer $(i + 1, 0)$ is done in a straightforward manner. That is, each intermediate vertex $w = w_1 \cdots w_{t+3}$ in layer (i, i') routes to vertex $w' = w'_1 \cdots w'_{t+3}$ such that $w'_{i'} = w_{i'} \oplus p_{i'}$ and $w'_t = w_t$ for all other t, where $p = p_1 \cdots p_{t+3}$. This routing can be enforced by a fixed coloring constraint (which will be applied at vertices of intermediate layers).

We still need to impose the (auxiliary) constraint $\mathbb{H}_i(j, k) = \mathbb{H}_i(j, k')$ for all k, k'. It suffices to impose this constraint for all k, k' that differ in one bit position; that is, it suffices to verify that, for every string $\alpha \in \{0, 1\}^{t-i+3}$ of Hamming weight one, $\mathbb{H}_i(j, k) = \mathbb{H}_i(j, \mathrm{int}(\mathrm{bin}_{t-i+3}(k) \oplus \alpha))$, for every (j, k), where $\mathrm{int}(\beta)$ is the integer represented by β. This can be achieved by routing $\mathbb{H}_i(j, k)$ (from vertex (j, k) of layer $(i, 0)$) to vertex (j, k) of layer $(i + 1, 0)$, via the "identity route" (i.e., without flipping bits), and letting vertex (j, k) of layer $(i + 1, i + i')$ compare $\mathbb{H}_i(j, k)$ (which is routed through it) to $\mathbb{H}_i(j, \mathrm{int}(\mathrm{bin}_{t-i+3}(k) \oplus 0^{i'-1}10^{t+3-i-i'}))$, which can be obtained from one of its neighbors. Again, this only blows up our sets of colors and vertex types by a constant, and the vertex types can be specified by a uniform $\mathbf{NC}_1$ circuit.

The above refers to the Items 5, 6 and 7 of Lemma 4.6. Dealing with Items 3 and 4, is much easier. In particular, for Item 3, we should require that the color of vertex $(0, 2\cdot 2^t + k)$ of layer $(0,0)$ corresponds to the k^{th} bit of y. Recall that the aforementioned color encodes $\mathbb{T}_0(0, 2\cdot 2^t + k)_I$, which is required to equal $(y_k, \perp)$. This correspondence is enforced by an adequate choice of the function f_{extract}. ∎

Combining Lemmas 4.6 and 4.7, Theorem 4.4 follows. ∎

5 Linear Maps and Efficient Algebraic Computation

In this section, we show that linear maps over $\mathbb{F}_2$-vector spaces can be expressed as sparse polynomials over some extension field of $\mathbb{F}_2$. We then show how the sparsity of these polynomials can be used to construct small-sized algebraic circuits from small-sized boolean circuits (for the corresponding functions).

Motivation: As hinted above, we do not need sparse polynomial representation per se, but rather use it to provide small circuits for computations that arise in our PCPP constructions. Specifically, the work of [BSb] refers to univariate polynomails of degree K (and [GS, BGHSVb] refers to multivariate polynomials of degree $2^{\sqrt{\log K}}$ or so). Some of the PCPP verification relies on the ability to evaluate such polynomial at a given input, and in our context we wish to perform this task in time $\operatorname{poly}\log K$. For a large degree d (e.g., $d > 2^{\sqrt{\log K}}$), this is possible in case the polynomial is sparse. Specifically, if a polynomial of degree d has t terms (with all coefficients being known), then it can be evaluated in time $t \cdot \operatorname{poly}\log d$.

5.1 Linear Maps are Sparse Polynomials

We first prove some general results about the sparse polynomial representation of linear maps over vector spaces over any finite field (not necessarily $\mathbb{F}_2$). These will be specialized to $\mathbb{F}_2$ and used in Section 5.2 (to obtain efficient algebraic circuits).

For an elaborate discussion of linear maps over vector spaces over finite fields, refer the excellent book on finite fields by Lidl and Niederreiter [LN, Chapter 3.4]. The results presented in this section can be proven using the techniques mentioned in [LN]. For the sake of completeness, we (re)state and prove the results in a form that is convenient for us.

Let $\mathbb{B} \subset \mathbb{F}$ be two fields of sizes $|\mathbb{B}| = q$ and $|\mathbb{F}| = q^f$ respectively. Let $H \subseteq \mathbb{F}$ be a vector space of dimension h over the (smaller) field $\mathbb{B}$; that is, H is a vector space $\mathbb{B}$-spanned by h elements of $\mathbb{F}$, i.e., there exists a basis $\{e_1, \ldots, e_h\}$ of h elements in $\mathbb{F}$ such that every element of H can be expressed as $\sum_{i=1}^{h} c_i e_i$ with $c_1, \ldots, c_h \in \mathbb{B}$. (For example, we may take $H = \mathbb{F}$ (in which case $h = f$).) A $\mathbb{B}$-linear map of H to $\mathbb{F}$ is a function $f : H \to \mathbb{F}$ that satisfies $f(ax + by) = af(x) + bf(y)$ for every $x, y \in H$ and $a, b \in \mathbb{B}$.

The following result shows that any linear map has a sparse polynomial representation.

Proposition 5.1 *Let $H \subseteq \mathbb{F}$ be a vector space of dimension h over the (smaller) field $\mathbb{B}$, and $f : H \to \mathbb{F}$ be $\mathbb{B}$-linear map. Then there exists a unique polynomial $\widehat{f} : \mathbb{F} \to \mathbb{F}$ of the form*

$$\widehat{f}(x) = \sum_{i=0}^{h-1} c_i x^{q^i}, \text{ where } c_0, \ldots, c_{h-1} \in \mathbb{F}$$

such that $\widehat{f}$ agrees with f on all of H. Moreover, given the evaluations of f on any basis for H, the coefficients $c_0, \ldots, c_{h-1}$ can be found with $\mathrm{poly}(h, \log q)$ *arithmetic operations over $\mathbb{F}$.*

Since $\widehat{f}$ is of degree at most $|H|/q$, we call $\widehat{f}$ the low-degree extension (LDE) of the linear map f.
Proof: Let $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ denote the set of $\mathbb{B}$-linear maps from H to $\mathbb{F}$. For $a, a' \in \mathbb{F}$ and L, L', defining $(aL + a'L')(z) = aL(z) + a'L'(z)$, establishes $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ as a vector space over the field $\mathbb{F}$. Observe that any $L \in \mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ is uniquely determined by its values on any basis $\{e_1, \ldots, e_h\}$ of H, because for any $\lambda_1, ..., \lambda_h$ it holds that $L(\sum_{i=1}^{h} \lambda_i e_i) = \sum_{i=1}^{h} \lambda_i L(e_i)$. On the other hand, any choice for values $v_1, ..., v_h$ for the points $e_1, ..., e_h$ can be extended to a linear map (i.e., $L(\sum_{i=1}^{h} \lambda_i e_i) = \sum_{i=1}^{h} \lambda_i v_i$). Thus, the number of linear maps in $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ is exactly $|\mathbb{F}|^h$, and so $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ is an h-dimensional vector space over $\mathbb{F}$.

We observe that, for any $i \geq 1$, the mapping $x \mapsto x^{q^i}$, denoted π_i, is a $\mathbb{B}$-linear mapping (of H to $\mathbb{F}$). Indeed, note that for $x, y \in H$ and $a, b \in \mathbb{B}$ it holds that $(ax + by)^q = (ax)^q + (by)^q = ax^q + by^q$, and so $\pi_i(ax + by) = (ax + by)^{q^i} = ax^{q^i} + by^{q^i} = a\pi_i(x) + b\pi_i(y)$. We claim that $\pi_0, ..., \pi_{h-1}$ constitutes a basis for $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$. This is shown by noting that the π_i's are linearly independent (over $\mathbb{F}$); that is $\sum_{i=0}^{h-1} c_i \pi_i(x) = \sum_{i=0}^{h-1} c_i x^{q^i}$ is the all-zero mapping if and only if all c_i's are zero. (The latter holds because $\sum_{i=0}^{h-1} c_i x^{q^i}$ has degree at most $q^{h-1} < |H|$ and therefore cannot vanish on H unless it is the all-zero polynomial.) Since the dimension of $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ over $\mathbb{B}$ is h, the maps $\pi_i, i = 0, \ldots, h-1$ constitute a basis for $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$ over the field $\mathbb{B}$. It follows that every linear map f can be represented as a unique linear combination of the π_i's.

The corresponding coefficients, denoted $c_0, ..., c_{h-1}$, can be found by solving (for the c_i's) the linear system $\{\sum_{i=0}^{h-1} c_i \pi_i(e_j) = f(e_j) : j = 1, \ldots, h\}$, where $e_1, ..., e_h \in \mathbb{F}$ is a basis for H. This involves computing the elements $\pi_i(e_j) = e_j^{q^i}, i, j \in \{0, 1 \ldots, h-1\}$ and then solving the linear system. The former can be done by repeated squaring in $\mathrm{poly}(h \log q)$ arithmetic operations over $\mathbb{F}$ while the latter requires $O(h^3)$ arithmetic operations over $\mathbb{F}$. The uniqueness of the solution follows, because otherwise there exist non-all-zero coefficients $c_0, ..., c_{h-1}$ such that $\sum_{i=0}^{h-1} c_i \pi_i$ is the all-zero mapping (from H to $\mathbb{F}$), which contradicts the (established) fact that $\pi_0, ..., \pi_{h-1}$ is a basis for $\mathcal{L}_{\mathbb{B}}^{H \to \mathbb{F}}$. ∎

We will be particularly interested in the following "S-vanishing polynomial", which can be defined for any $S \subseteq \mathbb{F} = \mathbb{F}_{q^f}$, but we will focus on the case that S is a vector space over the base field $\mathbb{B}$.

Definition 5.2 (The S-Vanishing Polynomial) *Consider an arbitrary subset of $\mathbb{F}$, denoted S. The S-vanishing polynomial is defined to be the polynomial whose zeros are precisely the elements of S. That is:*

$$Z_S(x) = \prod_{s \in S} (x - s).$$

Proposition 5.3 *If S is a vector space over the base field $\mathbb{B}$ then $Z_S : \mathbb{F} \to \mathbb{F}$ is a $\mathbb{B}$-linear map; that is:*

(a) For all $u, v \in \mathbb{F}$, $Z_S(u+v) = Z_S(u) + Z_S(v)$.

(b) For all $a \in \mathbb{B}, v \in \mathbb{F}$, $Z_S(av) = a \cdot Z_S(v)$.

Proof:

(a) Consider the bivarite polynomial $p : \mathbb{F}^2 \to \mathbb{F}$ defined as $p(x, y) \triangleq Z_S(x + y) - Z_S(x) - Z_S(y)$. Clearly, the total degree of p is strictly less than $|S|$ (note that the highest degree terms in x and y get cancelled) . Furthermore, for any two elements $u \in S$ and $v \in \mathbb{F}$, we have that $Z_S(v) = Z_{\{s-u:s\in S\}}(v) = Z_S(u + v)$, because S is a linear subspace of $\mathbb{F}$. Hence, if either $u \in S$ or $v \in S$, we have $p(u, v) = 0$. Thus, the fraction of zeros of p in $\mathbb{F}^2$ is at least $(2|S| \cdot |\mathbb{F}| - |S|^2)/|\mathbb{F}|^2 \geq |S|/|\mathbb{F}|$, because $|S| \leq |\mathbb{F}|$. This quantity is strictly greater than $\deg(p)/|\mathbb{F}|$. Hence, it follows from the Schwartz-Zippel Lemma that the polynomial p is the zero polynomial.

(b) If $a = 0$, then $Z_S(av) = Z_S(0) = 0 = a \cdot Z_S(v)$, because S (as a vector space over $\mathbb{B}$) contains 0. Otherwise (i.e., $a \neq 0$),

$$\begin{aligned} Z_S(av) &= \prod_{s\in S}(av - s) = a^{|S|} \cdot \prod_{s\in S}(v - a^{-1}s) = a^{|S|} \cdot \prod_{s\in S}(v - s) \\ &= a^{|S|} \cdot Z_S(v) = a \cdot Z_S(v) \end{aligned}$$

■

Proposition 5.4 *If S is a d-dimensional vector space over the base field $\mathbb{B}$ then there exist $c_0, \ldots, c_{d-1} in \mathbb{F}$ such that*

$$Z_S(x) = x^{q^d} + \sum_{i=0}^{d-1} c_i x^{q^i}$$

Moreover, the coefficients $c_0, \ldots, c_{d-1}$ can be computed with $\mathrm{poly}(d, \log q)$ arithmetic operations over $\mathbb{F}$, when given as input a basis for S.

Proof: If $S = \mathbb{F}$, then $Z_S(x) = x^{|\mathbb{F}|} - x = x^{|S|} - x = x^{q^d} - x$ and is of the required form. Suppose $S \subsetneq \mathbb{F}$. By Proposition 5.3, we have that $Z_S : \mathbb{F} \to \mathbb{F}$ is a $\mathbb{B}$-linear map. Hence, by Proposition 5.1 (as applied to $H = \mathbb{F}$), there exists a polynomial $\widehat{Z_S} : \mathbb{F} \to \mathbb{F}$ of the form $\widehat{Z_S}(x) = \sum_{i=0}^{f-1} c_i x^{q^i}$ that agrees with Z_S on all of $\mathbb{F}$. Since $S \subsetneq \mathbb{F}$, the degree of Z_S is $|S| < |\mathbb{F}|$, whereas the degree of $\widehat{Z_S}$ is $q^{(f-1)} < |\mathbb{F}|$. Thus, both Z_S and $\widehat{Z_S}$ are polynomials of degree less than $|\mathbb{F}|$, and since they agree on all of the field $\mathbb{F}$, they must be the same polynomial. Since Z_S is a monic polynomial of degree $|S| = q^d$, it follows that $Z_S(x) = \sum_{i=0}^{f-1} c_i x^{q^i}$, with $c_d = 1$ and $c_j = 0$ for every $j > d$, and the first part of the proposition follows.

The second part follows since the coefficients of $\widehat{Z_S}$ (and hence that of Z_S) can be computed with $\mathrm{poly}(d, \log q)$ arithmetic operations over $\mathbb{F}$ (c.f., Proposition 5.1).

■

5.2 Efficient Algebraic Computation

For the purpose of this section, we will interpret the results of the earlier section (Section 5.1) for the case when the smaller field is $\mathbb{B} = \mathbb{F}_2$. Hence, in the notation of the earlier section, $q = 2$. Recall that the larger field $\mathbb{F}$, which is an extension field of $\mathbb{F}_2$, can be viewed as a vector space over $\mathbb{F}_2$. Let H be an h-dimensional subspace of $\mathbb{F}$, spanned by the vectors $\{e_1, \ldots, e_h\}$.

Define $\texttt{bin} : H \to \{0,1\}^h$ to be the function that provides the representation of elements in H in terms of the aforementioned basis; that is, for any $x = \sum_{i=1}^{h} \lambda_i e_i \in H$, it holds that $\texttt{bin}(x) = (\lambda_1, \ldots, \lambda_h)$. Note that $\texttt{bin}$ is a one-to-one function, as so referring to it as a representation of elements in H is indeed justified. The function $\texttt{bin}$ can be naturally generalized to multiple inputs; that is, $\texttt{bin} : H^m \to \{0,1\}^{mh}$ satisfies $\texttt{bin}(x_1, \ldots, x_m) = \texttt{bin}(x_1) \circ \texttt{bin}(x_2) \circ \cdots \circ \texttt{bin}(x_m)$, where $\circ$ is the concatenation operator.

It is natural to call $\texttt{bin}(x)$ a *binary representation* of $x \in H$. Our main theorem shows that any small-depth, small-size Boolean circuit operating on the binary representation of H^m can be converted into an equivalent arithmetic circuit of small size and moderate degree (exponential in the depth) over $\mathbb{F}$. In particular it says that any bit of the binary representation of an element of H^m can be computed efficiently.

Theorem 5.5 *Let $\mathbb{B}, \mathbb{F}, H$ and $\texttt{bin}$ be as above. For any Boolean function $f : \{0,1\}^{mh} \to \{0,1\}$ computed by a circuit C of size s and depth d, there exists a polynomial $\widehat{f} : \mathbb{F}^m \to \mathbb{F}$ of degree at most $|H| \cdot 2^d$ computable by an $\mathbb{F}$-algebraic circuit C' of size $O(s + mh^2)$ such that*

$$\forall (x_1, \ldots, x_m) \in H^m, \quad \texttt{bit}(\widehat{f}(x_1, \ldots, x_m)) = f(\texttt{bin}(x_1, \ldots, x_m)),$$

where $\texttt{bit} : \mathbb{F} \to \{0, 1, \perp\}$ is defined such that $\texttt{bit}(0) = 0$, $\texttt{bit}(1) = 1$ and $\texttt{bit}(x) = \perp$ for every $x \in \mathbb{F} \setminus \mathbb{B}$. Moreover, C' can be constructed in polynomial-time, when given C, m, and a basis $\{e_1, \ldots, e_h\}$ for H.

Recall that $|H| = 2^h$. Thus, although the degree of $\widehat{f}$ may blow-up by a factor of 2^{h+d}, the size of the algebraic circuit remains almost unchanged if $s > mh^2$ (which will be the case in our applications).

Proof: For simplicity, we will abuse notation and omit the application of $\texttt{bit}$ to the output of $\widehat{f}$. We start by considering the special case when the function f is a projection function to a bit. In this case, we will show that there exists a polynomial $\widehat{f}$ of degree $|H|/2$ that agrees with f on H^m, and is computed by an $\mathbb{F}$-algebraic circuit of size $O(h)$.

We first prove this special case for $m = 1$. Without loss of generality, suppose $f : \{0,1\}^h \to \{0,1\}$ is the projection to the first bit function $x \mapsto x_1$. Consider the function $\tilde{f} : H \to \mathbb{F}$ defined by $\tilde{f}(x) = f(\texttt{bin}(x))$ (or rather satisfying $\texttt{bit}(\tilde{f}(x)) = f(\texttt{bin}(x))$). Thus, actually $\tilde{f} : H \to \mathbb{B}$. Furthermore, $\tilde{f}$ is a $\mathbb{F}_2$-linear map, because for any $x = \sum_{i=1}^{h} \lambda_i e_i$ and $x' = \sum_{i=1}^{h} \lambda'_i e_i$ it holds that $f(\texttt{bin}(x+x')) = \lambda_1 + \lambda'_1 = f(\texttt{bin}(x)) + f(\texttt{bin}(x'))$. Hence, by Proposition 5.1, there exists a unique low-degree extension $\widehat{f} : \mathbb{F} \to \mathbb{F}$ of $\tilde{f}$ that agrees with $\tilde{f}$ on all of H and has the following form

$$\widehat{f}(x) = \sum_{i=0}^{h-1} c_i x^{2^i}$$

for some $c_0, ..., c_{h-1} \in \mathbb{F}$. Observe that $\widehat{f}$ can be computed by an $\mathbb{F}$-algebraic circuit of size $O(h)$: the circuit first computes the powers $x, x^2, \ldots, x^{2^{h-1}}$, by repeated squaring, and then computes the appropriate linear combination. Also note that the degree of $\widehat{f}$ is at most $2^{h-1} = |H|/2$. This proves the result for projections when $m = 1$ (because, for all $x \in H$, it holds that $\widehat{f}(x) = \tilde{f}(x) = f(\mathtt{bin}(x))$).

The case for larger m is identical. The algebraic circuit only works with that component of $\mathbb{F}^m$ in which the projected bit is present, and ignores the remaining components of $\mathbb{F}^m$.

The above special case shows that any individual bit can be extracted by a polynomial of degree at most $|H|/2$ that is computable by a $\mathbb{F}$-algebraic circuit of size at most $O(h)$. The general claim (of the theorem) is then obtained by constructing an arithmetic circuit that first extracts all individual bits, and then applies (to them) a straightforward arithmetization of the original circuit C. For example, the arithmetization of the AND and NOT gates is performed as follows:

$$\begin{aligned} \textsc{not}(x) &= 1 - x \\ \textsc{and}(x, y) &= x \cdot y \end{aligned}$$

where the result maintains the intended value for $x, y \in \mathbb{B}$. (More generally, any binary gate $G(x, y)$ is replaced by an appropriate multilinear polynomial x and y, which extends the corresponding mapping $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$.) Thus, the size of the resulting $\mathbb{F}$-algebraic circuit is at most a constant factor larger than the size of original circuit C plus the size of the algebraic circuits extracting the individual bits, resulting in a total size of $O(s) + mh \cdot O(h)$. The degree of the polynomial computed by this circuit is at most 2^d times the degree of the algebraic circuit extracting the individual bits. The theorem follows. ∎

6 Algebraic Constraint Satisfaction Problems

In this section, we arithmetize the universal graph coloring problem (Theorem 4.4) to obtain an algebraic constraint satisfaction problem that is easily amenable to PCP constructions. Recall that we desire to construct efficient (wrt to running time) versions of the short PCPs of [BSb, BGHSVb]. The PCP constructions of [BSb] require a univariate algebraic CSP with just one constraint polynomial while that of [BGHSVb] require a multivariate algebraic CSP involving a logarithmic number of constraint polynomials. For this purpose, we construct two different (univariate and multivariate) algebraic constraint satisfaction problems for which PCPs can be constructed along the lines of [BSb] and [BGHSVb]. The key difference between the algebraic CSPs constructed in this paper and the ones in [BGHSVb, BSb] (as well as those of [PS, HS, BSVW]) is that the constraint polynomials in the CSPs constructed here can be obtained very efficiently. More specifically, the verifier can evaluate the constraint polynomial (at any point it wishes) in time polylogarithmic in the proof size. Verifiers in earlier constructions of nearly linear-sized PCPs required polynomial time for the same task.

We first arithmetize the graph coloring problem to a univariate algebraic CSP (Section 6.1) and then perform another arithmetization on the graph coloring problem to obtain a multivariate algebraic CSP (Section 6.2).

6.1 Univariate Algebraic CSP

The arithmetization to a univariate algebraic CSP is performed along the lines of [BSb]. Specifically, we can reduce via the identity mapping the Generalized deBruijn Graph Coloring to the following *univariate algebraic CSP*.

Definition 6.1 (Univariate Algebraic CSP) *The* Univariate Algebraic CSP (UNIALGCSP_t) *of size $t > m$ is parametized by two natural numbers α, d and seven* (fixed) *objects that are constructible in uniform* $\text{poly}(t)$*-time. The parameters are*

1. *A family of fields $\mathbb{F}_t = \mathbb{F}_{2^f}$, where $f = t + O(\log t)$, specified by an irreducible polynomial of degree f.*

2. *A family of linear spaces $H_t \subset \mathbb{F}_t$ of dimension $h = t + O(1)$, each specified by a basis that spans it.*

3. *A family of affine* (neighborhood) *maps $L_n = \langle \ell_t^i : \mathbb{F}_t \to \mathbb{F}_t \rangle_{i=1,\ldots,d}$, each specified in the straightforward manner* (i.e., $\ell_t^i(x) = a_i x + b_i$ for some $a_i, b_i \in \mathbb{F}_t$).

4. *A family of* (constraint) *polynomials $C_t : \mathbb{F}_t^{d+1} \to \mathbb{F}_t$ that have degree at most 2^{f-10} in the first variable and constant degree in the remaining variables, each specified by an algebraic circuit* (of size $\text{poly}(t)$).

5. *A family of affine subspace of $S_t \subseteq H_t$ of dimension t, each specified by a basis and an offset.*

6. *A family of bijections $I_t : [2^t] \to S_t$, specified by polynomial-time procedures for computing and inverting them.*

7. *A family of* (extraction) *functions $f^t_{\mathsf{extract}} : \mathbb{F}_t \to \{0,1\} \cup \{\perp\}$, specified by a polynomial-time procedure for computing them.*

Given an input $y \in \{0,1\}^K$, for $K < 2^t$, the problem is to determine whether there exists a assignment polynomial $A : \mathbb{F}_t \to \mathbb{F}_t$ of degree at most 2^h such that the following two conditions hold

1. *For all $x \in H_t$, it holds that $C_t(x, A(x), A(\ell_1(x)), \ldots, A(\ell_d(x))) = 0$. In this case, we say that the assignment polynomial A* satisfies *the constraint polynomial C_t.*

2. *For all $1 \le k \le K$ we have $y_k = f^t_{\mathsf{extract}}(A(I_t(k)))$. In this case, we say that the assignment polynomial A is* consistent *with the input y.*

Theorem 6.2 (Universality of Univariate Algebraic CSP) *For every Turing machine M, there exist a setting of the parameters for the Univariate Algebraic CSP* (of Definition 6.1) *such that the bounded-halting problem for M is reducible via the identity mapping to the corresponding Univariate Algebraic CSP* (i.e., the reduction is by the identity mapping). *Furthermore, for every $m < t$ and $y \in \{0,1\}^K$, where $K < 2^t$, machine M halts on y within 2^t steps if and only if there exists an assignment polynomial A, of degree at most 2^h, that satisfies the constraint polynomials and is consistent with y.*

The above arithmetization is performed along the lines of [BSb]. This arithmetization is performed in 2 steps: First, we embed the extended de Bruijn graph in an affine graph, and then we arithmetize the coloring constraints to efficiently-computable low-degree polynomials.

6.2 Multivariate Algebraic CSP

The arithmetization to a multivariate algebraic CSP is similar to the one performed in [BGHSVa]. Hence, we omit the details of the proof and mention only the differences between the original arithmetization and the one done here. We observe that the Generalized deBruijn Graph Coloring specified in Definition 4.1 is identical to the STRUCTURED-CKTSAT problem described in [BGHSVa, Definition 6.4], except for the following two differences:

1. An instance of STRUCTURED-CKTSAT is specified by a table of vertex types, which is given as an explicit input, and the verifier starts by constructing a low-degree extension of this table,[4] where the construction is done in polynomial-time (i.e., polynomial in the size of the table). In contrast, the vertex types in the universal graph coloring problem are specified by a fixed uniform $\mathbf{NC}_1$ circuit $f_{\mathsf{v-type}}$, and we need to compute a low-degree extension of it in polylogarithmic time (i.e., polylogarithmic in the length of the actual input, denoted y).

   Using the fact that the vertex type is specified by a $\mathbf{NC}_1$ circuit (applied to the vertex name) and using Theorem 5.5, we obtain the desired low-degree extension. Specifically, Theorem 5.5 asserts that the low-degree extension of any function specified by a circuit of size s (and depth d) can be computed by an arithmetic circuit of size polynomial in s (and logarithmic in the field size), and that the circuit transformation can be effecting in polynomial-time. Thus, the size of the arithmetic circuit (as well as the time needed to construct it) is polylogarithmic in the length of the aforementioned input y.

   We will also use the fact that this low-degree extension has degree at most $2^d \cdot |H|$, where the vertex set is associated with $H^m \subset \mathbb{F}^m$ (for some adequate m). Note that that $2^d \cdot |H| = \mathrm{poly}(m \log |\mathbb{F}|) \cdot |H|$, because $d = O(\log \log |\mathbb{F}|^m)$, whereas in [BGHSVb] the degree bound is $m|H|$. This means that we have to increase the field size by a factor of $\mathrm{poly}(m \log |\mathbb{F}|)$, which yields an additive increase of $m \cdot \log \mathrm{poly}(m \log |\mathbb{F}|) = O(m \log m) + O(m \log \log |\mathbb{F}|)$ in the randomness complexity of the verifier specification, which is dominated by the terms $O(m \log m) + O(m \log \log |\mathbb{F}|)$ that appear in it anyhow.

2. The description of graph coloring problems involves an extraction function f_{extract}, whereas that of STRUCTURED-CKTSAT does not have one. The extraction function f_{extract} allows for checking the consistency of the input y against the coloring function C. A function similar to this extraction function is added later in the PCP constructions of [BGHSVb], when building PCPs of proximity [BGHSVa, Section 7].

   We can arithmetize the extraction function f_{extract} as follows: We first observe that the extraction function is a constant sized sized circuit and hence a low-degree extension f^t_{extract} of it can be easily obtained in time polynomial in the logarithm of the field size (again using Theorem 5.5). Then, we need to know which portion of the space H^m corresponds to the input y. For this we note that the k^{th} bit of the input is supposed to be encoded in the (color of the) node numbered $2 \cdot 2^t + k$ in layer $0 = (0,0)$ of the $\mathcal{DB}_{k,l}$ graph. The arithmetization suggested in [BGHSVa] maps these vertices to a subset S_t of H^m whose membership can be easily checked. Furthermore, there exist efficiently computable an invertible maps $I_t : \{1, ..., |y|\} \to S_t$.

[4] Actually, the table consists of a sequence of tables $T_0, \ldots, T_{l-1}$, not to be confused with the unknown functions $T_0, ..., T_t$ appearing in Lemma 4.6. Thus, the BGHSV-verifier constructs low-degree extensions of the l functions.

Performing the arithmetization along the lines of [BGHSVa] with the above modifications, we can reduce via the identity mapping the Generalized deBruijn Graph Coloring to the following multivariate *algebraic CSP*.

Definition 6.3 (Multivariate Algebraic CSP) *The* Multivariate Algebraic CSP ($\textsc{MultiAlgCSP}_{t,m}$) *of dimension m and size $t > m$ is parametized by a constant number α and seven* (fixed) *objects that are constructible in uniform* $\mathrm{poly}(t)$*-time.*[5] *The parameters are*

1. *A family of fields, $\mathbb{F}_t = \mathbb{F}_{2^f}$, where $f = \lceil (t+3)/m \rceil + \alpha \log_2 t$, each specified by an irreducible polynomial of degree f.*
2. *A family of $\mathbb{F}_2$-linear spaces $H_t \subset \mathbb{F}_t$ of dimension $h \triangleq \lceil (t+3)/m \rceil$, each specified by a basis that spans it.*
3. *A family of affine* (neighborhood) *maps $L_t = \langle \Gamma_{i,b} : \mathbb{F}_t^m \to \mathbb{F}_t^m \rangle_{i=0,\ldots,(t+3)^2-1\,,\,b\in\{0,1\}}$ such that $\Gamma_{i,0}$ is the identity function, for all i, whereas $\Gamma_{i,1}$ flips the i^{th} bit in the binary representation of its input.*[6]
4. *A family of* (type-assignment) *polynomials $\mathcal{T}_t = \langle T_i : \mathbb{F}^m \to \mathbb{F} \rangle_{i=0,\ldots,(t+3)^2-1}$, each of degree at most $t^{\alpha-1} \cdot 2^h$, specified by algebraic circuits* (of size $\mathrm{poly}(t)$)*.*
5. *A family of* (constraint) *polynomials $\psi_t : \mathbb{F}_t^4 \to \mathbb{F}_t^2$ of constant degree κ. The polynomials ψ_t are specified by an algebraic circuit.*
6. *A family of* (extraction) *functions $f^t_{\mathsf{extract}} : \mathbb{F}_t \to \{0,1\} \cup \{\perp\}$ specified by a polynomial-time evaluation procedure.*
7. *A family of bijections $I_t : [2^t] \to H_t$ specified by polynomial-time procedures for evaluating I_t and its inverse.*

Given an input $y \in \{0,1\}^K$, for $K < 2^t$, the problem is to determine whether there exists a sequence of assignment polynomials $A_i : \mathbb{F}_t^m \to \mathbb{F}_t$, for $i = 0, \ldots, (t+3)^2 - 1$, each of degree at most $m \cdot 2^h$ such that the following two conditions hold

1. *For all $i \in \{0, \ldots, (t+3)^2 - 1\}$ and $x \in H_t^m$,*

$$\psi_t\Big(T_i(x), A_i(x), A_{i+1}(\Gamma_{i,0}(x)), A_{i+1}(\Gamma_{i,1}(x))\Big) = (0,0).$$

   In this case, we say that the assignment polynomials $A = \{A_i\}$ satisfy *the constraint polynomials ψ_t.*

2. *For all $1 \le k \le K$ we have $y_k = f^t_{\mathsf{extract}}(A_0(I_t(k)))$. In this case, we say that the assignment polynomials A are* consistent *with the input y.*

[5] In each case, there exists a uniform $\mathrm{poly}(t)$-time that given (m,t) produces the corresponding output.

[6] Since $\mathbb{F}_t = \mathbb{F}_2{}^f$, we can view $\mathbb{F}_t^m$ as mf-dimensional space over $\mathbb{F}_2$. Hence, any vector $(z_0, \ldots, z_{m-1}) \in \mathbb{F}_t^m$ can be written as $b = (b_{0,0}, ..., b_{0,f-1}, ..., b_{m-1,0}, ..., b_{m-1,f-1})$. Then $\Gamma_{i,1}(b) = (b'_{0,0}, ..., b'_{m-1,f-1})$, where $b'_{j,k} = b_{j,k} + 1$ if $j = \lfloor i/h \rfloor$ and $k = i \bmod m$ and $b'_{j,k} = b_{j,k}$ otherwise.

Note that the above definition of MultiAlgCSP is identical to that of AS-CktSAT in [BGHSVa] except for the following: (a) The constraint polynomials in MultiAlgCSP can be efficiently specified as a small sized algebraic circuit unlike a list of coefficients as in AS-CktSAT and (b) the MultiAlgCSP incorporates a proximity test while the AS-CktSAT did not.

Theorem 6.4 (Universality of Multivariate Algebraic CSP) *For every Turing machine M, there exist a setting of the parameters for the Multivariate Algebraic CSP* (of Definition 6.3) *such that the bounded-halting problem for M is reducible via the identity mapping to the corresponding Multivariate Algebraic CSP* (i.e., the reduction is by the identity mapping). *Furthermore, for every $m < t$ and $y \in \{0,1\}^K$, where $K < 2^t$, machine M halts on y within 2^t steps if and only if there exists a set of assignment polynomials $A = \{A_i\}$, each of degree at most $m \cdot 2^{\lceil (t+3)/m \rceil}$, that satisfies the constraint polynomials and is consistent with y.*

Proof Sketch: Using Theorem 4.4, it suffices to reduce the Generalized deBruijn Graph Coloring problem to the Multivariate Algebraic CSP. The constant α is determined by the depth of the $\mathbf{NC}_1$ circuit computing $f_{\mathsf{v-type}}$ (used in Theorem 4.4): specifically, $\alpha = c + 4$, where the aforemention circuit for inputs of length ℓ has depth $c \log_2 \ell$.

The constructions of the first three objects is straightforward, using the fact that an irreducible polynomial of degree f over $\mathbb{F}_2$ can be found in time polynomial in f [LN]. Specifically, we set $h = \lceil (t+3)/m \rceil$, which guarantees that $|H_t|^m = 2^{hm} \geq 8 \cdot 2^t$.

The other objects are obtained by low-degree extension of the corresponding objects in the Generalized deBruijn Graph Coloring problem. The bounds on the size of circuits and the degree of the polynomials computed by them (especially, the polynomials $\mathcal{T}_t$) are obtained using Theorem 5.5. ■

7 Verifier Specifications, Composition, and Simplifying Conventions

The standard approach for reducing the query complexity of PCP verifiers is the paradigm of "proof composition" [AS]. Instead of actually making the queries generated by the initial, "outer" verifier, an "inner" verifier is used to check (with the help of some auxiliary proofs) that the outer verifier would have accepted had it made its queries. The benefit is that the query complexity of the composed verifier equals that of the inner verifier (which is typically verifying a much shorter statement than the outer verifier). Note, however, that the running time of the composed verifier is the sum of the running times of the two verifiers.

According to the traditional formalism for verifiers, the running time of a verifier is at least its query complexity (in order to specify the queries). Most of the constructions of nearly-linear sized PCPs, such as [PS, HS, GS, BSVW, BGHSVb, BSb], obtain their savings in proof length by constructing initial verifiers of fairly large query complexity, such as $n^{\Omega(1)}$, and reducing the query complexity via composition. But, as described above, the *running time* of the composed verifier will remain large.

To overcome this difficulty, in this section, we present a formalism that allows a verifier's queries (and decision predicate) to be specified *implicitly*, e.g. by a small circuit. Producing this specification can then potentially be done in time that is polylogarithmic in the query complexity of the verifier. We show how such verifier specifications can be composed to reduce the query complexity, ultimately allowing us to construct standard verifiers with polylogarithmic running time (and short proof length, and small query complexity).

Verifier Specifications. Let M be a (deterministic) two-tape Turing Machine whose computations we wish to verify. (We will eventually apply the results to a universal Turing machine $M = M_U$ which we define later.) We will be constructing PCPP verifiers that aim to verify that a pair (x, y) is accepted by M within T steps. Such a verifier gets as explicit input (x, K, T), and expects oracle access to (y, π) for some (implicit input) $y \in \Sigma^K$ and proof (oracle) π. A standard verifier would toss some coins R, generate a sequence $Q = (i_1, \dots, i_k)$ of queries into the oracles (y, π) and then apply a decision predicate $D : \{0,1\}^k \to \{0,1\}$ to the received answers. We will allow both the queries and decision predicate to be specified implicitly. The queries will simply be specified by a circuit $Q : [k] \to \mathbb{Z}^+$. We cannot use a circuit for D itself, because such a circuit would have to be at least as large as its input length, which is k. Thus, instead we would like to specify D by an input x' (and a time bound T') to some fixed Turing machine M' (which should halt within T' steps).

Later, for composability we will take $M' = M$, but the more general formulation below is useful en route to our results. Also to facilitate composition, following [BGHSVb] we insist on *robust* soundness; that is, instead of merely requiring that (with high probability) the verifier rejects false assertions, we require that (with high probability) the answers (obtained from the queried locations) are *far* from satisfying the decision predicate.

Define $L_{M,x}(T) = \{y{:}M \text{ accepts } (x, y) \text{ in } T \text{ steps}\}$. We view strings as functions/oracles. So $y \in \Sigma^n$ may be viewed as a function/oracle $y : [n] \to \Sigma$. We use (y, z) to denote the concatenation of strings y and z, and $y|_Q$ to denote the restriction of y to $Q : [k] \to [n]$, i.e., $y|_Q[j] = y[Q[j]]$.

Actually, it will be convenient for us to allow our query circuit Q to sometimes output alphabet symbols from Σ instead of an index into the proof. Intuitively, $Q(i) = \sigma \in \Sigma$ means that the answer to the i'th query is always treated as σ (and thus no actual query needs to be made to the oracle). Thus for $Q : [k] \to [n] \cup \Sigma$ and $y \in \Sigma^n$, we define $y|_Q \in \Sigma^k$ by

$$y|_Q[j] = \begin{cases} y[Q[j]] & \text{if } Q[j] \in [n] \\ Q[j] & \text{if } Q[j] \in \Sigma. \end{cases}$$

Intuitively, a verifier specification for M (w.r.t M'), takes as input a triple (x, T, K), and randomly produces a quadruple (x', T', k, Q) such that $Q : [k] \to \mathbb{Z}^+ \cup \Sigma$ and the following condition holds: If $y \in L_{M,x}(T) \cap \Sigma^K$ (i.e., M accepts (x, y) in T steps) then there exists (an auxilary proof) π such that M' accepts $(x', (y, \pi)|_Q)$ in T' steps (i.e., $(y, \pi)|_Q \in L_{M',x'}(T')$). That is, $M'(x', \cdot)$ represents the decision predicate to be applied to answers obtained from the queries $Q[1], ..., Q[k]$ addressed to the oracle (y, π).

Definition 7.1 (Verifier Specification) *A* verifier specification *(or* verifier *or* specification*)* V *for* M *with respect to* M' *takes as input a triple* (x, T, K)*, where* $x \in \Sigma^*$ *and* T, K *are nonnegative integers given in binary such that* $T \geq |x| + K$*, and picks a random string* $R \in \{0,1\}^*$ *and outputs a quadruple* $(x', T', k, Q) = V(x, T, K; R)$ *where* $x' \in \Sigma^*$*,* T', k *are non-negative integers given in binary and* Q *is a circuit specifying a function from* $[k] \to \mathbb{Z}^+ \cup \Sigma$*. (If* $M' = M$*, then we simply call* V *a verifier specification for* M*.)*

Complexity (limitations): *For functions* $r, q, d, \ell : \mathbb{Z}^+ \to \mathbb{Z}^+$ *and* $\tau : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$ *we say that a verifier* V *is said to be* (r, q, ℓ, d, τ)-restricted *if* $V(x, T, K)$ *runs in time* $\tau(T, |x|)$*, tosses* $r(T)$ *random coins, and outputs a quadruple* (x', T', k, Q) *such that* $T' \leq d(T)$*,* $k \leq q(T)$ *and the range of* Q *is* $[K + \ell(T)]$*.*

We call r *the* randomness complexity*,* q *the* query complexity*,* d *the* decision complexity*,* ℓ *the* proof-size*, and* τ *the* running time *of the specification* V*.*

Guarantees (as a proof system): *For* $\rho, s, \delta : \mathbb{Z}^+ \to [0, 1]$*, the specification* V *is said to be* (δ, s, ρ)-sound *if the following two conditions hold:*

Completeness: *For every* $y \in L_{M,x}(T) \cap \Sigma^K$*, there exists a* $\pi \in \Sigma^{\ell(T)}$ *such that for every random string* R*, it holds that* M' *accepts* $(x', (y, \pi)|_Q)$ *in* T' *steps. where* $(x', T', k, Q) = V(x, T, K; R)$*.*

Robust Soundness: *If* $y \in \Sigma^K$ *is* $\delta(T)$*-far from* $L_{M,x}(T) \cap \Sigma^K$ *then, with probability at least* $1 - s(T)$*, it holds that* $(y, \pi)|_Q$ *is* $\rho(T)$*-far from* $L_{M',x'}(T') \cap \Sigma^q$*, where* $(x', T', k, Q) \leftarrow V(x, T, K)$*.*

We call δ *the* proximity parameter*,* s *the* soundness error*, and* ρ *the* robustness parameter *of* V*.*

The universal machine and its corresponding pair language for which we will construct a verifier specification is the following: Let M_U be a (deterministic two-tape) universal TM that on input $((M, w), z)$ simulates M on (w, z), with T steps of M being simulated using $\text{poly}(|M|) \cdot T$ steps of M_U. The following proposition shows that verifier specifications for M_U imply robust PCPP verifiers.

Proposition 7.2 *For any monotonically non-decreasing functions* r, q, ℓ, d, τ *and monotonically non-increasing functions* δ, s, ρ*, if the universal Turing Machine* M_U *has a verifier specification* V *that is* (r, q, ℓ, d, τ)*-restricted and* (δ, s, ρ)*-sound for some* $\rho < 1$*, then, for every pair language* L *in* $\text{NTIME}(T)$*, there is a constant* c *such that* $L \in \text{PCPP}_{s',\delta'}[r', q', t']$*, where* $s'(m) = s(c \cdot T(m))$*,* $\delta'(m) = 2\delta(c \cdot T(m))$*,* $r'(m) = r(c \cdot T(m))$*,* $q'(m) = q(c \cdot T(m))$*, and*

$$t'(n, K) = q\left(c \cdot T(n + K)\right) \cdot \tilde{O}\left(\tau\left(c \cdot T(n + K), n + c\right)\right) + d\left(c \cdot T(n + K)\right) .$$

Furthermore, this verifier utilizes proofs of length $\ell'(m) = \ell(2T(m) + m) + T(m)$*.*

Proof: The actual contents of the proof is that we can transform a verifier specification into a real verifier: The real verifier starts by computing the description of the queries to be made and the predicate to be evaluated, then makes the actual queries and applies the predicate to the answers obtained. But before doing this, we must address two techinal problems. Firstly, we are given a verifier specification for the computation of a single machine, whereas we need to provide real verifiers for a set of machines corresponding to a complexity class. This gap is minor, because the former machine is universal. However, a more serious problem is that the universal machine is deterministic (hence able to emulate deterministic machines) whereas we need to emulate a non-deterministic class. The obvious solution is to incorporate the non-deterministic choices as part of the implicit input, and (implicitly) obtain this part from the proof oracle. But when we do so, we should make sure that the length of the original implicit input dominates the length of the new implicit input (so that assertions regarding the distance of the new implicit input from some set will yield bounds on the distance of the original implicit input from the relevant set). Details follow.

We start with a nondeterministic TM, denoted M_0, that recognizes the pair language L within time complexity T. Let M_1 be the corresponding deterministic machine (i.e., M_1 accepts the pair $((x,y),z)$ if and only if M_0 accepts the input $(x,y) \in L$ when taking the nondeterministic choices described in z). Aiming to incorporate z and y in a new (implicit) input such that y dominates the length of the latter, we conside a machine M_2 such that $M(x,(y_1,\ldots,y_t),z) = 1$ if and only if $y_1 = y_2 = \cdots = y_t$ and $M_1((x,y_1),z) = 1$.[7] Note that $y \in L_{M_0,x}(T(|x|+K)) \cap \Sigma^K$ if and only if $((y_1,\ldots,y_t),z) \in L_{M,x}(2T(|x|+K)) \cap \Sigma^{tK+T(|x|+K)}$, which in turn holds if and only if $((y_1,\ldots,y_t),z) \in L_{M_U,(M,x)}(c \cdot T(|x|+K)) \cap \Sigma^{tK+T(|x|+K)}$, where $c = \mathrm{poly}(|M_0|)$. Thus, verifying that $(x,y) \in L$ (or that y is close to $\tilde{y}$ such that $(x,\tilde{y}) \in L$) reduces to verifying that $(y^t,z) \in L_{M_U,(M,x)}(c \cdot T(|x|+K)) \cap \Sigma^{tK+T(|x|+K)}$, where $K = |y|$ and $t = \lceil |z|/|y| \rceil$. Such a verifier may utilize a proof, denoted π.

Recall that we are given a verifier specification V for M_U, and we are going to construct a real verifier, denoted V', for L. The verifier V' proceeds as follows, when given explicit input (x,K), an implicit input $y \in \Sigma^K$, and a proof oracle $\pi' \in \Sigma^{\ell'(|x|+K)}$:

Let $m = |x|+K$, $T_U = cT(m)$, and $t = \lceil T(m)/K \rceil$.

1. The verifier V' invokes $V((M,x),T_U,2T(m))$ to obtain (x',T',k,Q).

   Note that $T' \leq d(T_U))$, $k \leq q(T_U)$ and $Q : [k] \to [2T(m) + \ell(T_U)]$. Futhermore, the size of the description of Q is upperbounded by $\tau(T_U, c+|x|)$.

   V' parses π' as (z,π), where $z \in \Sigma^{T(m)}$ and $\pi \in \Sigma^{\ell(c+|x|+2T(m))}$

2. V' computes $y' = ((y^t,z),\pi)|_Q$, via q evaluations of the circuit Q, and making queries to the oracle $(y,\pi') = (y,(z,\pi))$.

3. V' emulates M_U on (x',y') for T' steps, and accept if M_U accepts.

Note that the verifier's computation consists of one evaluation of V, taking time $T_V = \tau(T_U, c+|x|)$, followed by $k \leq q(T_U)$ evaluations of the circuit Q, which is of size at most T_V, and simulating M_U for $T' \leq d(T_U)$ steps. Since a circuit of size S can be evaluated in time $\tilde{O}(S)$, we obtain a total running-time of at most $T_V + k \cdot \tilde{O}(T_V) + T'$, which is upper-bounded by $q(T_U) \cdot \tilde{O}(\tau(T_U, c+|x|) + d(T_U)$, as required.

Most of the other parameters can be verified by inspection. For soundness, note that if (y^t,z) is δ-close to some $(\widehat{y}^t,\widehat{z})$ in $L_{M_U,(M,x)}(T_U)$, then y is 2δ-close to $\widehat{y}$ and M accepts $(x,\widehat{y}^t,\widehat{z})$ in $2T(|x|+|y|)$ steps. This implies that M_0 (non-deterministically) accepts $(x,\widehat{y})$ in $T(|x|+K)$ steps, where $K = |y|$. Thus, y is 2δ-close to L_x, as desired. ■

Composition. The next lemma shows how verifier specifications compose. This composition is similar to proof compositions as presented in prior works and in particular to that of [BGHSVb], except that the time complexity of the composed *verifier specification* may be much smaller than the query complexity of the associated (real) verifiers.

[7] The above description assumes some fixed parsing rule used by M to parse its input into the form $(x,(y_1,\ldots,y_t),z)$, for some t.

Lemma 7.3 *Suppose M has a verifier specification V_1 that is $(r_1, q_1, \ell_1, d_1, \tau_1)$-restricted and $(\delta_1, 1-\epsilon_1, \rho_1)$-sound, and a verifier specification V_2 that is $(r_2, q_2, \ell_2, d_2, \tau_2)$-restricted and $(\delta_2, 1-\epsilon_2, \rho_2)$-sound. Further, suppose that for every T it holds that $\delta_2(d_2(T)) \leq \rho_1(T)$. Then M has a verifier specfdication $V = V_1 \circ V_2$ that is $(r', q', \ell', d', \tau')$-restricted and (δ', s', ρ')-sound, for*

- *randomness complexity $r'(T) = r_1(T) + r_2(d_1(T))$,*
- *query complexity $q'(T) = q_1(d_1(T))$,*
- *proof length $\ell'(T) = \ell_1(T) + 2^{r_1(T)} \cdot \ell_2(d_1(T))$,*
- *decision complexity $d'(T) = d_2(d_1(T))$,*
- *specification running time $\tau'(T, n) = O(\tau_1(T, n) + \tau_2(d_1(T), \tau_1(T, n))) + \operatorname{poly}\log \ell'(T)$,*
- *robustness parameter $\rho'(T) = \rho_2(d_1(T))$,*
- *proximity parameter $\delta'(T) = \delta_1(T)$,*
- *soundness error $s'(T) = 1 - \epsilon_1(T) \cdot \epsilon_2(d_1(T))$, and*

Proof: The composed verifier $V_1 \circ V_2$ does the following on input (x, T, K): choose random coin tosses $R_1 \in \{0,1\}^{r_1(T)}$ and runs $V_1(x, T, K; R_1)$ to obtain (x_1, T_1, k_1, Q_1), and then runs $V_2(x_1, T_1, k_1)$ to obtain (x_2, T_2, k_2, Q_2). It constructs a circuit $Q' : [k_2] \to [K + \ell'(T)]$ that operates as follows.

$Q'(i)$**:** Compute $j = Q_2(i)$. If $j \leq k_1$, then output $Q_1(j)$. Otherwise (i.e., if $j > k_1$), output $K + \ell_1(T) + R_1 \cdot \ell_2(d_1(T)) + (j - k_1)$, where here R_1 is viewed as the binary representation of an integer in $[0, 2^{r_1(T)} - 1]$.

That is, the circuit Q' consists of the the circuits Q_1 and Q_2, as well as small circuits for doing the additional arithmetics.

The output of the verifier $V_1 \circ V_2$ is (x_2, T_2, k_2, Q').

Note that the computation of the composed verifier specification consists of one execution of V_1 on time bound T and an explicit input of length $n = |x|$, one execution of V_2 on time bound $T_1 \leq d_1(T)$ and an explicit input of length $|x_1| \leq \tau_1(T, n)$, and the construction of the circuit Q', which involves simple manipulations of the two circuits Q_1 and Q_2 and some arithmetic on numbers of length at most $\log \ell'(T)$. Thus, obtaining (x_1, T_1, k_1, Q_1) and (x_2, T_2, k_2, Q_2) is done in upto $\tau \triangleq \tau_1(T, n) + \tau_2(d_1(T), \tau_1(T, n))$ steps, and constricting Q' can be done within $O(\tau) + \operatorname{poly}\log \ell'(T)$ additional steps.

The rest of the analysis of the verifier is almost identical to the Composition Theorem of [BGHSVb]. ■

In the analysis above, we pay a constant factor in the running time of the composed verifier specification (compared to the sums of the running times of the two specifications), for the overhead in combining the two query circuits Q_1 and Q_2. If we compose k times, this gives a factor of $\exp(O(k))$. We are able to afford this (because we only compose for a double-logarithmic number of times), but we note that a more efficient way of doing multiple compositions exists: Specifically, rather than constructing the itermediate (query) circuits, we may just keep record of them and combine all of them together once we are done. Indeed, we rely on the fact that the (inner) verifier specification does not need to know the query circuit of the (outer) verifier specification with whom it is composed. Thus, the running time of the final composed verifier specification can be linear in the sum of the running times of the verifier specifications being composed, regardless of the number of compositions, plus a term that is polylogarithmic in the proof length (and hence also accounts for the number of compositions).

Simplifying Conventions. Finally, it will be convenient for us to not deal with any explicit input in our basic constructions, and in addition assume that that the implicit input is of length $K = \Omega(T)$, where T is the running time bound on the machine whose computations we wish to verify. That is, we will initially construct verifier specifications of the following form.

Definition 7.4 *A* simplified verifier specification V *for* M *(with respect to some TM* M'*) is defined in the same way as a verifier specification, except it is not given any explicit input* x *and the length* K *of the implicit input is set to* $K = T/100$.[8] *That is,* V *takes an input consisting of a nonnegative integer* T *in binary and outputs a quadruple* (x', T', k, Q). *The running time* τ *is measured as a function of only* T*, and the completeness and soundness now refer to the distance of strings* $y \in \Sigma^{T/100}$ *from* $L_M(T) = \{y : M \text{ accepts } y \text{ in } T \text{ steps}\}$.

We now argue that such a simplified verifier specification can be converted into a standard verifier specification with similar parameters.

Lemma 7.5 *For every TM* M*, there is a TM* $\widehat{M}$ *such that if* $\widehat{M}$ *has a simplified verifier specification (with respect to some TM* M'*) that is* $(\widehat{r}, \widehat{q}, \widehat{\ell}, \widehat{d}, \widehat{\tau})$*-restricted and* $(\widehat{\delta}, \widehat{s}, \widehat{\rho})$*-sound, then* M *has a (standard) verifier specification (with respect to* M'*) that is* (r, q, ℓ, d, τ)*-restricted and* (δ, s, ρ)*-sound, where* $r(T) = \widehat{r}(\tilde{O}(T))$, $q(T) = \widehat{q}(\tilde{O}(T))$, $d(T) = \widehat{d}(\tilde{O}(T))$, $s(T) = \widehat{s}(\tilde{O}(T))$, $\rho(T) = \widehat{\rho}(\tilde{O}(T))$, $\delta(T) = 12\widehat{\delta}(\tilde{O}(T))$*, and* $\tau(T, n) = \widehat{\tau}(\tilde{O}(T)) + \tilde{O}(n + \log T)$.

Proof: We begin with some intuition. In doing so, we will blur the distinction between PCPP verifiers and verifier specifications, and also focus on how to eliminate the explicit input x (and not how to make $K = \Omega(T)$). We want to construct a verifier V for M that is given explicit inputs (x, T, K) and oracles (y, π) and is able to verify that M accepts (x, y) in T steps. But we only have a verifier $\widehat{V}$ that can verify such statements (for a TM $\widehat{M}$ of our choosing) when there is no explicit input x given. A natural idea is to take $\widehat{M} = M$ and have $V^{(y,\pi)}(x, T, K)$ simulate $\widehat{V}^{((x,y),\pi)}(T, K + |x|)$. Note that if M accepts (x, y) in T steps, then by the completeness of $\widehat{V}$, there is π that makes the verifier accept. However, we run into a problem with soundness. If $\widehat{V}$ accepts with high probability, we only know that (x, y) is *close* to some string (x', y') such that M accepts (x', y') in T steps. But we need to obtain the stronger conclusion that y alone is close to some y' such that M accepts (x, y'). That is, we need to ensure that $x' = x$. The solution is to encode x in an error-correcting code ECC, and define $\widehat{M}$ so that on input (w, y), it first checks that w is a valid codeword, then computes $x = \mathrm{ECC}^{-1}(w)$, and finally simulates M on (x, y). This ensures that if $(\mathrm{ECC}(x), y)$ is close to some (w, y') that makes $\widehat{M}$ accept, then by the distance property of the code we have $w = \mathrm{ECC}(x)$, and also M accepts (x, y), as desired.

Several technical issues come up in implementing this idea. First, we need the error-correcting code to be nearly linear time in order to closely maintain the efficiency parameters of our verifier.

[8] We have not attempted to optimize constants.

Second, since we are working with verifier specifications rather than PCPP verifiers, the transformation for the oracle (y, π) into the oracle $((\mathrm{ECC}(x), y), \pi)$ will have to be carried out entirely by manipulation of the query circuits for the specifications. Third, in order for closeness of pairs $(\mathrm{ECC}(x), y)$ and (w, y') to translate into closeness in each of the two components, the components have to be of nearly equal length. We achieve this by repetition, i.e. replacing $(\mathrm{ECC}(x), y)$ with $(\mathrm{ECC}(x)^{\ell}, y^{\ell'})$ for an appropriate k, and having $\widehat{M}$ check that the input is indeed of this form. Thus we need to ensure that the error-correcting code's distance and decoding properties are maintained under repetition (even when the length of the message and number of repetitions are not known). This repetition technique is also what allows us to ensure that $K = \Omega(T)$.

Now we proceed with the actual proof. We assume without loss of generality that $\{0,1\} \subseteq \Sigma$. We will need an error-correcting code $\mathrm{ECC} : \Sigma^* \to \Sigma^*$ with the following properties:

1. For every u, $|\mathrm{ECC}(u)| = c \cdot |u|$, for a constant $c \in \mathbb{N}$.

2. On inputs u of length m, $\mathrm{ECC}(u)$ can be evaluated in time $\tilde{O}(m)$.

3. For every two strings $u \in \Sigma^m$, $u' \in \Sigma^{m'}$, and every $t \geq \max\{cm, cm'\}$, the strings

$$\mathrm{ECC}(u)^{\lceil t/cm \rceil} 0^{2t - \lceil t/cm \rceil \cdot cm} \text{ and } \mathrm{ECC}(u')^{\lceil t/cm' \rceil} 0^{2t - \lceil t/cm' \rceil \cdot cm'}$$

   are (1/5)-far.

4. There is an algorithm that given a string $v \in \Sigma^{2t}$ for some $t \in \mathbb{N}$, runs in time $100t$ and finds an u such that $v = \mathrm{ECC}(u)^{\lceil t/cm \rceil} 0^{2t - \lceil t/cm \rceil \cdot cm}$ where $m = |u| \leq t/\log^c t$, if such a u exists. If no such u exists, then the algorithm rejects.

Such codes can be obtained by modifications to the nearly linear-time codes of [**?**]; we defer the tedious details to the appendix.

Now, given a TM M, we define the following TM $\widehat{M}$.

Turing machine $\widehat{M}$(Input):

1. Check that Input is of length $4t$ for some $t \in \mathbb{N}$ (reject if not), and let $v \in \Sigma^{2t}$ be the first half of the input.

2. Using the algorithm from Property 4 of ECC, find a u such that $v = \mathrm{ECC}(u)^{\lceil t/cm \rceil} 0^{2t - \lceil t/cm \rceil \cdot cm}$ and $m = |u| \leq t/\log^c t$, if such a u exists. If no such u exists, then immediately reject.

3. Parses u as a pair $u = (x, K, T)$.

4. Check that the second half of Input is of the form $y^{\lceil t/K \rceil} 0^{2t - \lceil t/K \rceil \cdot K}$ for some $y \in \Sigma^K$, rejecting immediately if not.

5. Run M on (x, y) for T steps and accept or reject as M does.

Now, suppose we have a simplified verifier specification $\widehat{V}$ for $\widehat{M}$. We construct a verifier specification V' for M as follows.

Verifier specification $V(x, T, K; R)$**:**

1. Let $t = \lceil 2T \cdot \log^c T \rceil$, $\widehat{K} = 4t$, and $\widehat{T} = 400t$.
2. Let $u = (x, T, K)$, and compute $w = \mathrm{ECC}(u)$.
3. Let $m = |u|$, $\ell = \lceil t/m \rceil$, $\ell' = \lceil t/K \rceil$.
4. Let $(x', T', k, \widehat{Q}) = \widehat{V}(\widehat{T}, \widehat{K}; R)$.
5. Let $Q : [k] \to \mathbb{Z}^+ \cup \Sigma$ be a circuit defined as follows:

$$Q[i] = \begin{cases} w[\widehat{Q}[i] \bmod cm] & \text{if } \widehat{Q}[i] \leq cm \cdot \ell \\ 0 & \text{if } cm \cdot \ell < \widehat{Q}[i] \leq 2t \\ (\widehat{Q}[i] - 2t) \bmod K & \text{if } 2t < \widehat{Q}[i] \leq 2t + K \cdot \ell' \\ 0 & \text{if } 2t + K \cdot \ell' < \widehat{Q}[i] \leq 4t = \widehat{K} \\ \widehat{Q}[i] - \widehat{K} + K & \text{if } \widehat{K} < \widehat{Q}[i]. \end{cases}$$

6. Output (x', T', k, Q).

The complexity parameters can be verified by inspection, so we focus on completeness and soundness.

Completeness. Suppose $y \in L_{M,x}(T) \cap \Sigma^K$. We claim that if u, w, m, ℓ, ℓ' are computed as in the definition of V above, then $\widehat{M}$ accepts $w^\ell 0^{2t-\ell \cdot cm} y^{\ell'} 0^{2t-\ell' \cdot K}$ in time $\widehat{T} = 400t$. This follows from the definitions of V and $\widehat{M}$, provided we verify that $m \leq t/\log^c t$ and that $\widehat{T}$ is enough time for $\widehat{M}$ to complete all of its steps. We bound on m as follows:

$$m = |u| \leq n + O(\log T) \leq T + O(\log T) \leq \frac{2T \cdot \log^c T}{(\log T + c \log\log T + 1)^c} = \frac{t}{\log^c t}.$$

For the running time of $\widehat{M}$, we note that Step 2 takes at most $100t$ steps by Property 4 of ECC. Steps 1, 3, and 4 are all simple linear-time parsing steps and can certainly be done in a total of $200t$ steps. Step 5 takes $T \leq t$ steps. The total number of steps is at most $301t < \widehat{T}$.

Therefore there exists a π such that with probability 1 over the random string R, it holds that $(w^\ell 0^{2t-\ell \cdot cm} y^{\ell'} 0^{2t-\ell' \cdot K}, \pi)|_{\widehat{Q}} \in L_{M',x'}(T')$. By the construction of Q, this is equivalent to $(y, \pi)|_{\widehat{Q}} \in L_{M',x'}(T')$, as desired.

Robust Soundness. We prove this by the contrapositive. Let $y \in \Sigma^K$, and suppose that with probability greater than $s(T)$ over the random string R, it holds that $(y, \pi)|_Q$ is $\rho(T)$-close to $L_{M',x'}(T')$. By the construction of Q, this means that with probability greater than $s(T)$, it holds that (Input, π) is $\rho(T)$-close to $L_{M',x'}(T')$, for $\mathsf{Input} = w^\ell 0^{2t-\ell \cdot cm} y^{\ell'} 0^{2t-\ell' \cdot K}$. By setting $s(T) = \widehat{s}(\widehat{T})$ and $\rho(T) = \widehat{\rho}(\widehat{T})$ and applying the robust soundness of $\widehat{V}$, we deduce that Input is $\widehat{\delta}(\widehat{T})$-close to some string $\overline{\mathsf{Input}}$ in $L_{\widehat{M}}(\widehat{T}) \cap \Sigma^{\widehat{K}}$.

By the definition of $\widehat{M}$, we have $\overline{\mathsf{Input}} = \overline{w}^{\overline{\ell}} 0^{2t-\overline{\ell} \cdot c\overline{m}} \overline{y}^{\overline{\ell'}} 0^{2t-\overline{\ell'} \cdot \overline{K}}$, where $\overline{w} = \mathrm{ECC}(\overline{u})$, $\overline{u} = (\overline{x}, \overline{K}, \overline{T})$, $\overline{m} = |(\overline{x}, \overline{K}, \overline{T})|$, $\overline{\ell} = \lceil t/c\overline{m} \rceil$, $\overline{\ell'} = \lceil t/\overline{K} \rceil$, and M accepts $(\overline{x}, \overline{y})$ within $\overline{T}$ steps.

Since Input and $\overline{\mathsf{Input}}$ are $\widehat{\delta}$-close (for $\widehat{\delta} = \widehat{\delta}(\widehat{T})$), it follows that $w^{\ell}0^{2t-\ell \cdot cm}$ and $\overline{w}^{\overline{\ell}}0^{2t-\overline{\ell} \cdot \overline{cm}}$ are γ-close for $\gamma = (\widehat{K}/(2t)) \cdot \widehat{\delta} = 2\widehat{\delta} < 1/5$. By Property 3 of ECC, it follows that $u = \overline{u}$; that is, $x = \overline{x}$, $K = \overline{K}$, and $T = \overline{T}$. Similarly, y and $\overline{y}$ are ϵ-close for $\epsilon = (\widehat{K}/(\ell' \cdot K)) \cdot \widehat{\delta} \leq 4\widehat{\delta} \leq \delta$. We conclude that y is δ-close to $L_{M,x}(T)$, as desired. ■

References

[AGHP] Alon, N., Goldreich, O., Håstad, J., and Peralta, R. Simple constructions of almost $k-$wise independent random variables. *Journal of Random Structures and Algorithms 3*, 3 (Fall 1992), 289–304.

[ALMSS] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. Proof verification and the hardness of approximation problems. *Journal of the ACM 45*, 3 (May 1998), 501–555. (Preliminary Version in *33rd FOCS*, 1992).

[AS] Arora, S., and Safra, S. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM 45*, 1 (Jan. 1998), 70–122. (Preliminary Version in *33rd FOCS*, 1992).

[BFLS] Babai, L., Fortnow, L., Levin, L. A., and Szegedy, M. Checking computations in polylogarithmic time. In *Proc.* 23*rd ACM Symp. on Theory of Computing* (New Orleans, Louisiana, 6–8 May 1991), pp. 21–31.

[BG] Barak, B. and Goldreich, O. Universal Arguments and their Applications. In *Proc.* 17*th IEEE Conference on Computational Complexity* (Montréal, Québec, Canada, 21–24 May 2002), pp. 75–81.

[BGHSVa] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., and Vadhan, S. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. ECCC Technical Report TR04-021, March 2004. (This is the full version of [BGHSVb]).

[BGHSVb] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., and Vadhan, S. Robust PCPs of proximity, shorter PCPs and applications to coding. In *Proc.* 36*th ACM Symp. on Theory of Computing* (Chicago, Illinois, 13-15 June 2004), pp. 1–10. (See full version [BGHSVa]).

[BGHSVc] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., and Vadhan, S. Short PCPs verifiable in polylogarithmic time. In *Proc.* 20*th IEEE Conference on Computational Complexity* (San Jose, California, 12–15 June 2005), pp. ??–??.

[BSa] Ben-Sasson, E., and Sudan, M. Simple PCPs with polylog rate and query complexity. ECCC Technical Report TR04-060, 2004. (This is the full version of [BSb]).

[BSb] Ben-Sasson, E., and Sudan, M. Simple PCPs with polylog rate and query complexity. To appear in *Proc.* 37*th ACM Symp. on Theory of Computing*, Baltimore, Maryland, 21–24 May 2005. (See full version [BSa]).

[BSVW] Ben-Sasson, E., Sudan, M., Vadhan, S., and Wigderson, A. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proc. 35th ACM Symp. on Theory of Computing* (San Diego, California, 9–11 June 2003), pp. 612–621.

[DR] Dinur, I., and Reingold, O. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. In *Proc. 45rd IEEE Symp. on Foundations of Comp. Science* (Rome, Italy, 17–19 Oct. 2004), pp. 155-164.

[EKR] Ergün, F., Kumar, R., and Rubinfeld, R. Fast approximate PCPs. In *Proc. 31st ACM Symp. on Theory of Computing* (Atlanta, Georgia, 1–4 May 1999), pp. 41–50.

[FGLSS] Feige, U., Goldwasser, S., Lovász, L., Safra, S., and Szegedy, M. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM 43*, 2 (Mar. 1996), 268–292. (Preliminary version in *32nd FOCS*, 1991).

[Gol] Goldreich, O. A sample of samplers – a computational perspective on sampling. Tech. Rep. TR97-020, Electronic Colloquium on Computational Complexity, 1997.

[GGR] Goldreich, O., Goldwasser, S., and Ron, D. Property testing and its connection to learning and approximation. *Journal of the ACM 45*, 4 (July 1998), 653–750. (Preliminary Version in *37th FOCS*, 1996).

[GS] Goldreich, O., and Sudan, M. Locally testable codes and PCPs of almost linear length. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science* (Vancouver, Canada, 16–19 Nov. 2002), pp. 13–22. (See ECCC Report TR02-050, 2002).

[HS] Harsha, P., and Sudan, M. Small PCPs with low query complexity. *Computational Complexity 9*, 3–4 (Dec. 2000), 157–201. (Preliminary Version in *18th STACS*, 2001).

[Kil] Kilian, J. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th ACM Symp. on Theory of Computing* (Victoria, British Columbia, Canada, 4–6 May 1992), pp. 723–732.

[LN] Lidl, R., and Niederreiter, H. *Introduction to Finite Fields and their applications.* Cambridge University Press, Cambdridge, United Kingdom, 1994.

[Mic] Micali, S. Computationally sound proofs. *SIAM Journal of Computing 30*, 4 (2000), 1253–1298. (Preliminary Version in *35th FOCS*, 1994).

[NN] Naor, J., and Naor, M. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd ACM Symp. on Theory of Computing* (Baltimore, Maryland, 4–16 May 1990), pp. 213–223.

[PF] Pippenger, N., and Fischer, M. J. Relations among complexity measures. *Journal of the ACM 26*, 2 (Apr. 1979), 361–381.

[PS] Polishchuk, A., and Spielman, D. A. Nearly-linear size holographic proofs. In *Proc. 26th ACM Symp. on Theory of Computing* (Montréal, Québec, Canada, 23–25 May 1994), pp. 194–203.

[RS] Rubinfeld, R., and Sudan, M. Robust characterizations of polynomials with applications to program testing. *SIAM Journal of Computing 25*, 2 (Apr. 1996), 252–271. (Preliminary Version in *23rd STOC*, 1991 and *3rd SODA*, 1992).

[VL] Venkatesan, R., and Levin L.A. Random Instances of a Graph Coloring Problem are Hard. In *Proc. 20th ACM Symp. on Theory of Computing*, (White Plains, New York, 24–26 Oct 1988), pp 217–222.