Shorter Arithmetization of Nondeterministic Computations*

Alessandro Chiesa
alexch@csail.mit.edu
MIT

Zeyuan Allen Zhu
zeyuan@csail.mit.edu
MIT

July 15, 2015

Abstract

Arithmetizing computation is a crucial component of many fundamental results in complexity theory, including results that gave insight into the power of interactive proofs, multi-prover interactive proofs, and probabilistically-checkable proofs. Informally, an arithmetization is a way to encode a machine's computation so that its correctness can be easily verified via few probabilistic algebraic checks.

We study the problem of arithmetizing nondeterministic computations for the purpose of constructing short probabilistically-checkable proofs (PCPs) with polylogarithmic query complexity. In such a setting, a PCP's proof length depends (at least!) linearly on the length, in bits, of the encoded computation. Thus, minimizing the number of bits in the encoding is crucial for minimizing PCP proof length.

In this paper we show how to arithmetize any T-step computation on a nondeterministic Turing machine by using a polynomial encoding of length

$$O\big(T \cdot (\log T)^2\big) \ .$$

Previously, the best known length was $\Omega(T \cdot (\log T)^4)$. For nondeterministic random-access machines, our length is $O(T \cdot (\log T)^{2+o(1)})$, while prior work only achieved $\Omega(T \cdot (\log T)^5)$.

The polynomial encoding that we use is the Reed–Solomon code. When combined with the best PCPs of proximity for this code, our result yields quasilinear-size PCPs with polylogarithmic query complexity that are shorter, by at least two logarithmic factors, than in all prior work.

Our arithmetization also enjoys additional properties. First, it is *succinct*, i.e., the encoding of the computation can be probabilistically checked in $(\log T)^{O(1)}$ time; this property is necessary for constructing short PCPs with a polylogarithmic-time verifier. Furthermore, our techniques extend, in a certain well-defined sense, to the arithmetization of yet other NEXP-complete languages.

Keywords: arithmetization; probabilistically-checkable proofs; polylogarithmic-time verifier

*The authors thank Eli Ben-Sasson and Eran Tromer for insightful comments and discussions. They took an active part in stages of this research, and yet declined to be co-authors. The authors also thank Silvio Micali for valuable comments on this write up. This work is partially supported by the Skolkovo Foundation (with agreement dated 10/26/2011), and two Simons Graduate Student Awards (under grant no. 282907, 284059).

1 Introduction

Efficient *arithmetization* of computation is a significant component of many fundamental results in complexity theory [BFL90, BF91, BFLS91, LFKN92, Sha92, She92, FL93, FGL+96, FK97, AS98, ALM+98]. Roughly, arithmetization re-expresses a computation's correctness in a more "robust" language: an *algebraic* constraint satisfaction problem, whose satisfiability can be verified via probabilistic-checking procedures. In this work, we study the *efficiency of arithmetizing nondeterministic computations*, from a perspective that is motivated by the construction of short probabilistically-checkable proofs, as we now explain.

1.1 Probabilistically-Checkable Proofs

A *probabilistically-checkable proof* (PCP) [BFLS91, FGL+96, AS98, ALM+98] for a language $\mathcal{L} \in \mathrm{NEXP}$ is a probabilistic oracle machine V (the *verifier*) that works as follows: given as input an instance x and given oracle access to a candidate proof $\pi \in \{0,1\}^*$ for the statement "$x \in \mathcal{L}$", the verifier V reads only a few bits of π and then decides if to accept or reject. If "$x \in \mathcal{L}$" is true then there is a proof that makes V always accept; else, if "$x \in \mathcal{L}$" is false, no proof makes V accept with more than $1/2$ probability.

Definition 1.1. *A* **PCP** *for a language $\mathcal{L} \in \mathrm{NEXP}$ is a probabilistic oracle machine V that works as follows.*
- Completeness: *For every instance $x \in \mathcal{L}$, there exists a string $\pi \in \{0,1\}^*$ such that* $\Pr[V_x^\pi = 1] = 1$.
- Soundness: *For every instance $x \notin \mathcal{L}$ and any string $\pi \in \{0,1\}^*$ it holds that* $\Pr[V_x^\pi = 1] < \frac{1}{2}$.

Interpretation. PCPs are a *semantic analogue* of locally-decodable codes (LDCs). An LDC can be used to encode a long **message** m into a codeword w so that, even if w is adversarially corrupted at a few places, any bit of the original message m can still be decoded by reading only a few places of the (corrupted) codeword. Analogously, PCPs can be used to encode a long **(nondeterministic) computation** into a *holographic* (or *transparent*) proof π so that, even if the computation is adversarially corrupted at a few places, the result of the computation can still be verified by reading only a few places of the proof π. Therefore, instead of merely robustly encoding information, *PCPs robustly encode computation*.

Algebraic PCP Constructions. Many algebraic PCP constructions consist of two basic components:
1. a *proximity tester* (or, more generally, a *PCP of proximity* [DR04, BGH+06]) for a certain code; and
2. a *computation tester*.

The proximity tester first checks if a proof π is close to the code. Then, given that π is close to the code, the computation tester can check, via a few random queries, the correctness of the computation's output.

Other PCP constructions. There are PCP constructions that are combinatorial or do not fit the above 2-step paradigm [Din07, MR08, DH09, Mei09, Mei12]. In this paper, we focus on the above paradigm.

1.2 Arithmetization In PCPs

Within the setting of algebraic PCP constructions:

> An *arithmetization* of a language $\mathcal{L}$ is the procedure of constructing a computation tester for $\mathcal{L}$.

In this paper, the intuitive notion of a computation tester for $\mathcal{L}$ is formalized via a modification of the PCP model, called a *code PCP*, that "ignores" proximity testing (the other component of a PCP). Namely, we define a *code PCP* to be a PCP where the proof π is not an arbitrary binary string, but is restricted to lie in a (possibly non-binary) linear code C, specified together with the verifier.[1] In a (standard) PCP, an adversary trying to fool the verifier may select any π; but, in a code PCP, the adversary may select proofs only among codewords in C. More generally, in a code PCP, we actually allow proofs $\vec{\pi} = (\pi_i)_i$ where each π_i is restricted to lie in the i-th code C_i of a sequence $\vec{C} = (C_i)_i$ of linear codes, specified together with the verifier.

[1] A *linear code* C of block length $n \in \mathbb{N}$, message length $k \in \mathbb{N}$, and alphabet size $q \in \mathbb{N}$ is a k-dimensional linear subspace of $\mathbb{F}_q^n$.

Definition 1.2. *A* **code PCP (cPCP)** *for a language $\mathcal{L} \in \mathrm{NEXP}$ is a pair $(V, \vec{C}) = \big((V_x)_x, (\vec{C}_x)_x\big)$, where each V_x is a probabilistic oracle machine and $\vec{C}_x = (C_{x,i})_i$ is a sequence of linear codes, satisfying:*

- Completeness: *For every instance $x \in \mathcal{L}$ there exist codewords $\vec{\pi} \in \vec{C}_x$ such that $\Pr\left[V_x^{\vec{\pi}} = 1\right] = 1$.*
- Soundness: *For every instance $x \notin \mathcal{L}$ and any codewords $\vec{\pi} \in \vec{C}_x$ it holds that $\Pr\left[V_x^{\vec{\pi}} = 1\right] < \frac{1}{2}$.*

Every PCP is also a code PCP, by fixing the (trivial) code $C = \{0,1\}^*$ of all binary strings. However, not every code PCP is a PCP. Nonetheless, as mentioned, a code PCP $(V, \vec{C})$ can be used to construct a (standard) PCP if $\vec{C}$ is a locally testable code (or, more generally, if $\vec{C}$ has a PCP of proximity): the PCP verifier first tests that the candidate proof $\vec{\pi}$ is close to $\vec{C}$, and then invokes V on $\vec{\pi}$.[2] In general, though, $\vec{C}$ need not be locally testable. Hence, in the rest of this paper, we focus our attention to a specific, yet large, class of codes for which we know of good proximity testers: codes consisting of low-degree polynomials. (This class includes Reed–Solomon and Reed–Muller codes, which are popular choices.)

Definition 1.3. *A (linear) code C is a* low-degree polynomial code *if C is the evaluations over $\mathbb{F}_q^n$ of all $f \in \mathbb{F}_q[z_1, \ldots, z_n]$ of degree d, for positive integers n, d and prime power q such that $2nd < q$.*[3] *Then, $\vec{C} = (\vec{C}_x)_x$ is a* **low-degree polynomial code** *if every $C_{x,i}$ in each $\vec{C}_x = (C_{x,i})_i$ is a polynomial code.*

For this paper, we define an *arithmetization for $\mathcal{L}$* as an efficient procedure to construct a code PCP for $\mathcal{L}$:

Definition 1.4. *An* **arithmetization** *of $\mathcal{L} \in \mathrm{NEXP}$ is a polynomial-time function f such that $(V, \vec{C}) = \big((V_x)_x, (\vec{C}_x)_x\big)$, where $(V_x, \vec{C}_x) = f(x)$, is a code PCP for $\mathcal{L}$ and $\vec{C}$ is a low-degree polynomial code.*[4]

The above definition captures arithmetization in algebraic PCP constructions, because an "arithmetization of $\mathcal{L}$", in such a setting, has the following form. First, design an algebraic constraint such that $x \in \mathcal{L}$ iff there is a witness/computation that, when suitably encoded, satisfies the constraint. Second, due to the low-degree properties of the constraint and encoding, the statement "$x \in \mathcal{L}$" can be probabilistically checked by testing a few polynomial identities at random points. These two steps imply a code PCP $(V, \vec{C})$ for $\mathcal{L}$ where $\vec{C}$ is a low-degree polynomial code and V is a machine that probabilistically checks the polynomial identities.

1.3 Efficiency Measures for Arithmetization

We consider the following efficiency measures for a code PCP and an arithmetization:

Definition 1.5. *A code PCP $(V, \vec{C})$ has:*

- proof length $\mathsf{pl}\colon \{0,1\}^* \to \mathbb{N}$ *if the sum of the bit lengths of codes in $\vec{C}_x$ is at most $\mathsf{pl}(x)$ (i.e., if $\sum_i n_{x,i} \log_2 q_{x,i} \leq \mathsf{pl}(x)$ where $n_{x,i}, q_{x,i}$ are the block length and alphabet size of $C_{x,i}$ in $\vec{C}_x$);*
- query complexity $\mathsf{qc}\colon \{0,1\}^* \to \mathbb{N}$ *if V_x makes at most $\mathsf{qc}(x)$ queries to $\vec{C}_x$; and*
- verifier complexity $\mathsf{vc}\colon \{0,1\}^* \to \mathbb{N}$ *if V_x runs in at most $\mathsf{vc}(x)$ time.*

Definition 1.6. *The* proof length*,* query complexity*, and* verifier complexity *of an arithmetization are defined to be those of the code PCP induced by the arithmetization.*

A universal language. We fix the NEXP-complete *universal language* $\mathcal{L}_{\mathsf{TM}}$ of instances $x = (M, T)$, where M is a Turing machine and T is a time bound, such that there is a witness w for which $M(w)$ accepts in T steps. Then, one should think of $\mathsf{pl}(x)$ as polynomial in T (i.e., enough to encode M's computation) and $\mathsf{qc}(x), \mathsf{vc}(x)$ as polynomials in $\log T$ (i.e., V_x queries $\mathrm{poly}\log T$ locations in $\vec{C}_x$ and runs in $\mathrm{poly}\log T$ time).

[2] More precisely, for this step the queries asked by V must be "sufficiently random", so that proximity to the code is helpful. But we shall ignore this technicality in definitions because, for natural constructions, queries are always sufficiently random.

[3] The condition $2nd < q$ *defines* what "low degree" means, and is sufficient to imply local testability [BFL90, BF91, LFKN92].

[4] A pair $(V_x, \vec{C}_x)$ is efficiently represented as follows: V_x is specified via a Turing machine, and $\vec{C}_x = (C_{x,i})_i$ via a sequence of triples $\big((n_{x,i}, d_{x,i}, q_{x,i})\big)_i$ where each $(n_{x,i}, d_{x,i}, q_{x,i})$ are the parameters of the polynomial code $C_{x,i}$ (see Definition 1.3).

1.4 Our Question

The efficiency of a PCP depends on the efficiency of *both* the code PCP and proximity tester used to construct it. E.g., the proof length, query and verifier complexity of a PCP obtained from a code PCP $(V, \vec{C})$ (and a proximity tester for $\vec{C}$) depend *(at least) linearly* on the corresponding efficiency measures of the code PCP.

There has been extensive research on proximity testers in both coding theory and complexity theory, especially with regard to testing proximity to low-degree polynomial codes [GLR$^+$91, PS94, RS96, FGL$^+$96, RS97, AS97, AS98, ALM$^+$98, BSVW03, GS06]. The improved understanding gleaned from this research program has almost always translated into better PCP (and LDC) constructions.

In contrast, not much research has studied the efficiency of arithmetization (i.e., constructing code PCPs), and known arithmetizations are not particularly efficient. Many PCP constructions inherit this complexity (and thus have their proof length depend at least linearly on this complexity).

How efficiently can we arithmetize nondeterministic computations?

From a technical perspective, *proof length is our main focus*. Minimizing proof length is important in the construction of *short* PCPs [BFLS91, PS94, HS00, BSVW03, BGH$^+$04, BGH$^+$05, GS06, Din07, BS08, MR08, Mie09, BCGT13b, BCGT13a, BKK$^+$13]. For instance, [BS08, BGH$^+$05, BCGT13b, BCGT13a] obtain $\tilde{O}(T)$-size PCPs, and an arithmetization with proof length $\tilde{O}(T)$ is a crucial ingredient in all these works. In sum, in this paper our goal is to minimize the proof length required to arithmetize languages in NEXP, subject to verifier (and thus query) complexity being polynomial in $\log T$. (We are motivated by the setting of succinct verification [BFLS91], where $O(1)$ query complexity is not as crucial as other parameters.)

1.5 Our Results

In this paper we introduce new arithmetization techniques that enable us to obtain an arithmetization of nondeterministic computations that is both simpler and more efficient than in previous work.

Theorem 1 (informal)**.**
Any T-step computation on a nondeterministic (multi-tape) Turing machine can be arithmetized with:

- *proof length $O(T(\log T)^2)$;*
- *query complexity $O(\log T)$; and*
- *verifier complexity* $\mathrm{poly}(\log T)$.

Previously, the best proof length was $\Theta(T(\log T)^4)$ [BCGT13a]. In Appendix A we argue that improving proof length beyond our result requires significantly new ideas, and we propose a concrete open problem.

Shorter PCPs. Theorem 1 gives a code PCP with proof length $O(T(\log T)^2)$, query complexity $O(\log T)$, and verifier complexity $\mathrm{poly}(\log T)$. The code PCP in our construction relies on the *Reed–Solomon code*. Combining our code PCP with a proximity tester for the Reed–Solomon (RS) code yields a PCP.

At present, known proximity testers for the RS code [BS08, BCGT13b] require auxiliary "proximity proofs" that are not short enough. Thus, while our result implies a PCP that is shorter than in all prior work (by at least two logarithmic factors), its proof length remains $\omega(T(\log T)^2)$. Future constructions of proximity testers may be more efficient and, when combined with our code PCP, would yield a PCP of proof length $O(T(\log T)^2)$. (This seems plausible, given that RS-code proximity testing has steadily improved over the years [PS94, BS08, BCGT13b].) For now, some complexity-theoretic results already benefit from shorter quasilinear-size PCP proofs [SW13].

Overall, one can interpret Theorem 1 as progress towards a PCP of proof length $O(T(\log T)^2)$. Indeed, it shows that the cost of arithmetizing nondeterministic computation is not a bottleneck.

In comparison, Ben-Sasson et al. [BKK$^+$13] construct a PCP where, for all $\varepsilon > 0$, the proof length is $O_\varepsilon(T \log T)$, but the query and verifier complexity are T^ε and $O_\varepsilon(T)$ respectively. Instead, we require (and achieve) query complexity $O(\log T)$ and verifier complexity $\mathrm{poly}(\log T)$.

Computational efficiency. Our improved proof length does not sacrifice computational properties of an arithmetization that are crucial for some applications. First, our arithmetization is *succinct*, i.e., the verifier

in the resulting code PCP runs in only $\mathrm{poly}(\log T)$ time (and not $\Omega(T)$ as, e.g., in [PS94]). This feature is necessary when relying on Theorem 1 to construct PCPs with a *polylogarithmic-time verifier* [BFLS91, BGH+05, Mei09, Mie09, BCGT13b]. As we shall see, succinctness is a quite delicate property.

Furthermore, our arithmetization comes with two functions F and F^{-1}, both running in quasilinear sequential time and polylogarithmic parallel time, such that (i) F maps a T-step accepting trace of execution of a machine to a satisfying proof for the code PCP and (ii) F^{-1} maps a satisfying proof for the code PCP to an accepting trace of execution of the machine. Such functions are important when relying on Theorem 1 to construct PCPs with a quasilinear-time prover and with a proof of knowledge [BCGT13b].

Beyond Turing machines. Nondeterministic computations on Turing machines are very special: the transition function of a Turing machine M has size, as a boolean circuit, that is *constant* (i.e., independent of the time bound T); hence, the dependence on M in Theorem 1 is not shown. But this is *not* the case for many other models of computation, inducing NEXP-complete languages, that we may want to arithmetize.

For example, for the language of satisfiable succinctly-described circuits, M is a boolean circuit that describes a T-gate circuit: given index $i \in [T]$, M outputs the two inputs, two outputs, and type of the i-th gate; so M has size $\Omega(\log T)$. As another example, in the case of random-access machine computations, M is a boolean circuit that computes (at least) the next address for memory, which consists of $\Omega(\log T)$ bits.

Simply reducing other models of computation to (multi-tape) Turing machines incurs polylogarithmic factors in overhead [GS89, Rob91], which is too expensive. Instead, we demonstrate that our techniques can directly, and efficiently, arithmetize other computation models besides nondeterministic Turing machines, by stating (and proving) our results also for nondeterministic random-access machines and the satisfiability of succinctly-described circuits. We selected these familiar NEXP-complete languages because they are very different from one another. (At least when paying attention, as we do, to polylogarithmic-factor overheads!)

Informally, when the dependence on M is made explicit, our arithmetization's proof length is:

$$\mathsf{pl}(M,T) = O(T \log T) \cdot \min\left\{ \log T + \mathsf{size}(M) \ , \ \log T \cdot (\log T + \mathsf{deg}(M)) \right\} \ ,$$

where $\mathsf{size}(M)$ is the size of the boolean circuit (corresponding to) M, and $\mathsf{deg}(M)$ the total degree of this circuit when viewed as an $\mathbb{F}_2$-arithmetic circuit. Below, we compare proof lengths of by prior and our work.

reference	proof length of arithmetization for T-step computation on machine M
[BFL90]	$(T \cdot \mathsf{size}(M))^{O(\log \log T)}$
[BFLS91]	$(T \cdot \mathsf{size}(M))^{1+\Theta(1/\varepsilon)}$
[BS08, BGH+05]	$T \cdot 2^{(\mathsf{size}(M))^{O(1)}} \cdot (\log(T))^{O(1)} \cdot \mathsf{deg}(M)$
[BCGT13a]	$T \cdot (\log T)^3 \cdot (\log T + \mathsf{deg}(M)) \cdot O(1)$
this work	$T \cdot \log T \cdot \min\left\{ \log T + \mathsf{size}(M) \ , \ \log T \cdot (\log T + \mathsf{deg}(M)) \right\} \cdot O(1)$

We note that, while incomparable,[5] $\mathsf{size}(M)$ is typically much smaller than $\mathsf{deg}(M)$; when this holds, we achieve even greater savings over prior work. For instance, in a "typical" universal random-access machine M (supporting basic arithmetic instructions and simple load/store instructions), one can verify that $\mathsf{size}(M) = (\log T)^{1+o(1)}$ and $\mathsf{deg}(M) = \Omega((\log T)^2)$. For such machines, we obtain proof length $T(\log T)^{2+o(1)}$ while prior work only obtained $\Omega(T(\log T)^5)$. In fact, since a "transcript of execution" of a random-access machine has $\Omega(T \log T)$ bits, our proof length is *only a logarithmic factor away from the information-theoretic lower bound*. See Appendix A for a further discussion about how tight our result is.

Arithmetizing yet other languages. As we shall describe, our techniques can be directly extended to yet other NEXP-complete languages. Concretely, our main technical contribution is showing how to arithmetize a quite general *constraint satisfaction problem* (CSP). Because this problem supports reductions, with only constant overhead, from Turing machines, random-access machines, and circuits, we believe that it can

[5] For a boolean circuit C, while $\mathsf{deg}(C) \leq 2^{\mathsf{size}(C)}$, $\mathsf{deg}(C)$ and $\mathsf{size}(C)$ are *incomparable*. E.g., the circuit for computing $\oplus_{i=1}^{n} x_i$ has $\Omega(n)$ size and $O(1)$ degree, while that for x^{2^n} has $O(n)$ size and $O(2^n)$ degree. Similarly, $\mathsf{size}(M)$ and $\mathsf{deg}(M)$ are incomparable. (In particular, the $\min$ between the two terms in the last line of the table cannot be avoided.)

efficiently express many other natural NEXP-complete languages. And reducing a language to the CSP we consider is a *much easier* task than arithmetizing the language from scratch in the general case.

Remark 1.7 (linear PCPs)**.** A special case of a code PCP is a *linear PCP* [BCI$^+$13]. A linear PCP is a code PCP where $C = (C_x)_x$ is fixed so that C_x is the code of all linear functions $f\colon \mathbb{F}^{\ell(x)} \to \mathbb{F}$ for some $\ell(x) \in \mathbb{N}$. One can view constructions of linear PCPs as "linear arithmetizations". Compared to arithmetizations in the PCP literature, linear arithmetizations [Gro10, Lip12, GGPR13, BCI$^+$13, SBV$^+$13, PGHR13, BCG$^+$13] are conceptually and technically simpler, due to two significant relaxations: they make use of *exponential-size* proof length and are *not succinct* (i.e., V_x runs in time that is exponential, and not polynomial, in $|x|$.)

1.6 Prior Challenges

If we only wanted to encode a witness w into a code with constant relative distance, while minimizing block length, this would be a very simple task: simply encode w with any good error-correcting code. However, our goal is not to encode w for storage purposes, but to encode w (or, more generally, information about w such as $M(w)$'s execution trace) into codewords $\vec{\pi} \in \vec{C}_x$ —where $\vec{C}_x$ is a sequence of codes of our choice— so that a verifier V_x can easily check, with few random queries to $\vec{\pi}$, if $M(w)$ accepts in T steps.

Alternatively, if we only wanted to allow V_x to check the statement without minimizing block length, this would also be a relatively simple task. As Babai at al. [BFL90] showed, it is straightforward to reduce the statement "$x \in \mathcal{L}$" to a problem of the following form: "for given integers $m, m' = O(\log T)$ and $(m' + 3m)$-variate polynomial P, does there exist a multilinear polynomial $A \in \mathbb{F}[z_1, \ldots, z_m]$ such that $P(\vec{\gamma}, A(\vec{\gamma}_1), A(\vec{\gamma}_2), A(\vec{\gamma}_3)) = 0$ for every $\vec{\gamma} \in \{0,1\}^{m'}$ and $\vec{\gamma}_1, \vec{\gamma}_2, \vec{\gamma}_3 \in \{0,1\}^m$?". Intuitively, A is a multilinear extension of $M(w)$'s execution trace, and the algebraic constraint imposed by P verifies that A is valid and accepting. By using standard techniques for verification of vanishing properties, this fact immediately yields a code PCP with proof length $T^{O(\log \log T)}$ and query complexity $\mathrm{poly}(\log T)$.

Significant technical difficulties arise when minimizing the block length of a codeword encoding information about w, while maintaining the ability to easily check the statement. At high level, this task consists of proving "algebraic analogues" of the *efficient Cook–Levin theorem* (where, by relying on oblivious Turing machines [PF79], the resulting constraint satisfaction problem has size $O(T \log T)$ instead of $O(T^2)$).

Prior work has tackled these challenges by leveraging *routing networks* [Ofm65, BFLS91] as a powerful tool for sorting (crucial for efficient reductions [Sch78, SH86, GS89, Rob91]) as well as for "structuring" a computation. In particular, routing networks having convenient algebraic properties, such as *De Bruijn graphs*, have proved to be very useful [PS94, BS08, BGH$^+$05, BCGT13a].

1.7 Current Challenges and Our Techniques

Several significant inefficiencies remain unaddressed. We introduce techniques to overcome these and obtain a code PCP with proof length $O(T(\log T)^2)$, query complexity $O(\log T)$, and verifier complexity $\mathrm{poly}(\log T)$.

First, unlike earlier arithmetizations that rely on *multivariate* low-degree extensions to encode $M(w)$'s execution trace, recent arithmetizations achieving proof length $\tilde{\Theta}(T)$ rely on *univariate* low-degree extensions over fields of size $\Omega(T)$. Intuitively, this makes sense: for a given relative distance, the rate (ratio of block to message length) of the Reed–Solomon code is better than that of Reed–Muller. However, the tighter encoding causes *projecting* a field element to a bit in its representation to be a costly high-degree operation (since the field is large); the verifier needs these bits in order to manipulate them and evaluate certain constraints. Prior work paid the price of projections, or simply stored one bit per field element, thus incurring at least a $\Omega(\log T)$ overhead in proof length. Second, previous arithmetizations incurred inefficiencies arising from the total degree of the constraints that the verifier needs to check, which is at least $\Omega(\log T)$; and it seems unlikely that this degree can be reduced to $o(\log T)$. Yet, even when ignoring other costs, these constraints alone require an arithmetization of proof length $\Theta(T(\log T)^4)$.

At high level, in this work we circumvent the first difficulty by showing how the witness can still be verified even when a large fraction of its bits are stored in *blocks* of bits; we store bit-by-bit only a small

fraction of the witness. We thus avoid *both* the use of costly projection operations, as well the wasteful encoding of a single bit per field element. To circumvent the second difficulty, we use a *degree reduction* technique that reduces checking a constraint polynomial of total degree $\Omega(\log T)$ to checking a collection of $O(1)$-degree polynomials, essentially without incurring any cost for the original constraint's degree.

In both of the above, our main technical tool is leveraging computational properties of *selector polynomials* [BCGT13a] and *linearized polynomials* [LN97, Section 2.5] in finite field extensions of $\mathbb{F}_2$ [BGH+05].

1.8 Roadmap

The rest of this paper is organized as follows. In Section 2, we formally state our main result: Theorem 1. In Section 3, we explain our high-level proof strategy, and introduce the main technical lemma proved in this paper. In Section 4, we explain the proof intuition behind the main technical lemma, leaving details to subsequent sections. Specifically, in Section 5, we introduce basic definitions and facts, and then prove the main technical lemma in three steps, respectively given in Section 6, Section 7, and Section 8.

2 Formal Statement of Main Theorem

We state our theorem for three familiar NEXP-complete languages that are quite different from each other. As argued in Section 3, our techniques extend (in a well-defined sense) to arithmetize any language in NEXP.

The first language we consider, denoted $\mathcal{L}_{\mathsf{TM}}$, is *bounded-halting problems on Turing machines*; it consists of pairs (M, T), where M is a (multi-tape) Turing machine and T is an integer time bound, for which there exists a witness w such that $M(w)$ accepts in T steps:

$$\mathcal{L}_{\mathsf{TM}} = \big\{(M, T) \,:\, \exists\, w \text{ s.t. the Turing machine } M \text{ accepts } w \text{ in } T \text{ steps}\big\}.$$

The second language we consider, denoted $\mathcal{L}_{\mathsf{CSAT}}$, is *satisfiability of succinctly-described circuits*; it consists of pairs (M, T), where M is a circuit descriptor that specifies a circuit C with T gates,[6] for which there exists a witness w such that $C(w) = 1$:

$$\mathcal{L}_{\mathsf{CSAT}} = \big\{(M, T) \,:\, \exists\, w \text{ s.t. the } T\text{-gate circuit described by the descriptor } M \text{ accepts } w\big\}.$$

Finally, the third language we consider, denoted $\mathcal{L}_{\mathsf{RAM}}$, is *bounded-halting problems on random-access machines* [CR72, AV77]; it consists of pairs (M, T), where M is a random-access machine and T is an integer time bound, for which there exists a witness w such that $M(w)$ accepts in T steps:

$$\mathcal{L}_{\mathsf{RAM}} = \big\{(M, T) \,:\, \exists\, w \text{ s.t. the random-access machine } M \text{ accepts } w \text{ in } T \text{ steps}\big\}.$$

In both $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{RAM}}$, we identify the machine M with the boolean circuit that computes the machine's transition function (the circuit M has different syntax in the case of Turing and random-access machines).

Overall, in each of $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$, we have that M specifies a certain boolean circuit. We denote by $\mathsf{size}(M)$ the size of this circuit (with AND and NOT gates), and $\mathsf{deg}(M)$ its total degree when viewed as an $\mathbb{F}_2$-arithmetic circuit. (Degrees across multiple output wires are added to obtain the total.)

In light of the above definitions, our main theorem can be stated as follows.

Theorem 1 (formally restated)**.** *Let $\mathcal{L}$ be $\mathcal{L}_{\mathsf{TM}}$, $\mathcal{L}_{\mathsf{CSAT}}$, or $\mathcal{L}_{\mathsf{RAM}}$. There is an arithmetization f for $\mathcal{L}$ with:*

- *proof length* $\mathsf{pl}(M, T) = O(T \log T) \cdot \min\big\{\ \log T + \mathsf{size}(M)\ ,\ \log T \cdot (\log T + \mathsf{deg}(M))\ \big\}$*;*
- *query complexity* $\mathsf{qc}(M, T) = O(\mathsf{size}(M) + \log T)$*;*
- *verifier complexity* $\mathsf{vc}(M, T) = \mathrm{poly}(\mathsf{size}(M) + \log T)$.

We stress that, while the definitions of $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ look similar *typographically*, these languages represent *very different* computational problems. (At least when counting polylogarithmic factors in simulation overheads!) First of all, $\mathcal{L}_{\mathsf{CSAT}}$ is about an "oblivious" model of computation, while $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{RAM}}$ are not. (And recall that making a Turing machine oblivious incurs a logarithmic factor overhead [PF79].) Second, the way that a machine accesses memory in $\mathcal{L}_{\mathsf{TM}}$ or $\mathcal{L}_{\mathsf{RAM}}$ is entirely different: sequential in the first

[6] A circuit descriptor M is a circuit that takes as input a gate $g \in [T]$, and outputs the type of the gate g, as well as the names of the gates connected to the (at most 2) input and (at most 2) output wires of g.

case, and random-access in the second. Third, the sizes and degrees of the circuit M in the three languages are quite different: we can consider a Turing machine M to be of size independent of T (as T increases); in contrast, a descriptor M for a T-gate circuit takes as input and produces as output $\log T$-bit gate names, requiring at least a size (or a degree) of $\Omega(\log T)$; for a random-access machine, M's size and degree also depend on T, because M needs to be able to output an address for memory.

In the next section, we describe our high-level proof strategy and state our main technical lemma.

3 Proof Structure and Statement of Main Lemma

We break the proof of Theorem 1 into sub-parts by using two intermediate NEXP-complete languages:

- The language of *succinct indirect-access constraint satisfaction problems* (sICSP).
- The language of *succinct algebraic constraint satisfaction problems* (sACSP).

Then, given the above languages, our proof of Theorem 1 consists of three parts:

- **Part (i):** An efficient reduction from each of $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ to sICSP.
- **Part (ii):** An efficient reduction from sICSP to sACSP.
- **Part (iii):** A code PCP construction for sACSP of suitable efficiency.

We note that:

> *Our technical contribution is part (ii):* it is our arithmetization's *engine*, and is tasked with efficiently translating an "unstructured and combinatorial" CSP into a "structured and algebraic" CSP.

Parts (i) and (iii) can be obtained either from prior work or familiar techniques.

Next, we define the language sICSP (Definition 3.1) and the language sACSP (Definition 3.3). Then we provide three formal statements: Proposition 3.4 for part (i); Lemma 1 for part (ii); and Proposition 3.5 for part (iii). Later, in Section 4, we sketch our proof of Lemma 1, this paper's technical contribution; proof details are in the appendices. For completeness, proofs for Propositions 3.4, 3.5 are also in the appendices.

sICSP: The language of succinct indirect-access constraint satisfaction problems. An instance of a *constraint satisfaction problem* (CSP) is typically specified by a list of constraints over a set of variables; the instance is satisfiable if there is an assignment to the variables such that every constraint is satisfied. The language sICSP is a language of satisfiable CSPs with the following special properties: (a) the numbers of constraints and variables are the same (and we denote it by T); (b) the list of constraints is *succinct* (i.e., they can be described by a single circuit K); (c) constraints only have *indirect-access* to variables (roughly, each constraint gets as input three variables, of which two are "adversarially selected" in a certain way).

Definition 3.1. *The language* sICSP *consists of triples* (T, c, K)*, where*

- $T \in \mathbb{Z}_+$ *is the number of variables/constraints,*
- $c \in \mathbb{Z}_+$ *is the number of bits in the value assigned to a variable, and*
- $K \colon [T] \times \{0,1\}^{3c} \to \{0,1\}$ *is a boolean constraint circuit,*

such that there is a witness (χ, π_1, π_2)*, where* $\chi \colon [T] \to \{0,1\}^c$ *assigns a* c*-bit value to each* $v \in [T]$ *and* $\pi_1, \pi_2 \colon [T] \to [T]$ *are two permutations, for which* $K\big(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))\big) = 0$ *for every* $v \in [T]$.

While odd-looking, the language sICSP is a succinct CSP that can efficiently express many types of "unstructured" computation. (E.g., Proposition 3.4 gives efficient reductions from $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ to sICSP.) Since our techniques apply to any sICSP instance, our results extend to any NEXP-complete language $\mathcal{L}$ by reducing $\mathcal{L}$ to sICSP. (Reducing to sICSP is a *much easier* task than arithmetizing $\mathcal{L}$ from scratch.)

We find it useful to visualize instances of sICSP via a certain graph-coloring problem over T vertices, in which χ specifies a c-bit color to each vertex, the permutations π_1, π_2 specify two (adversarial) edge sets, and K specifies coloring constraints. See Figure 1 on page 8 for a diagram.

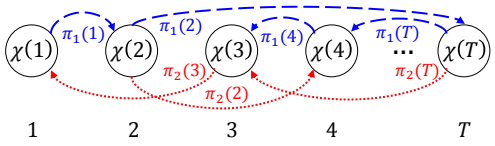

Figure 1: An sICSP instance (T, c, K) and a candidate witness (χ, π_1, π_2).

Remark 3.2. One can define sICSP$[k]$ to be a generalization of sICSP having k, instead of 2, permutations in the witness. So what is special about $k = 2$? One can show that sICSP$[k]$ reduces to sICSP $=$ sICSP$[2]$ with only a blowup of k in the number of variables/constraints. Therefore, it suffices to study sICSP.

sACSP: The language of succinct algebraic constraint satisfaction problems. We present the language sACSP via an analogy with succinct (not necessarily algebraic) CSP problems. A succinct CSP problem is specified by triples $(H, \vec{N}, P)$, where H is a (large) vertex set, $\vec{N} = (N_i \colon H \to H)_{i=1}^{d}$ is a sequence of neighbor functions inducing the edge set $\{(\gamma, N_i(\gamma))\}_{\gamma \in H, i \in [d]} \subset H \times H$, and P is a constraint circuit. The instance $(H, \vec{N}, P)$ is satisfiable if there is an assignment A on H satisfying all constraints induced by P, i.e., for each vertex γ in H it holds that $P(\gamma, A(\vec{N}(\gamma))) \stackrel{\text{def}}{=} P\big(\gamma, A(N_1(\gamma)), \ldots, A(N_d(\gamma))\big) = 0$. At high level, the language sACSP is a variant of succinct CSP with *algebraic constraints*. Formally:

Definition 3.3 ([BGH$^+$05, BCGT13b])**.** *The language* sACSP *consists of all tuples* $(\mathbb{F}, H, \vec{N}, P)$ *where,*

- $\mathbb{F}$ *is a field of characteristic* 2*,*
- $H \subseteq \mathbb{F}$ *is an* $\mathbb{F}_2$*-linear subspace,*
- $\vec{N} = (N_i \colon \mathbb{F} \to \mathbb{F})_{i=1}^{d}$ *is a sequence of neighbor polynomials, and*
- $P \colon \mathbb{F}^{1+d} \to \mathbb{F}$ *is a constraint polynomial,*

such that there is an assignment polynomial $A \colon \mathbb{F} \to \mathbb{F}$ *of degree* $\leq |H|$ *for which the following holds:*

1. DEGREE CONSTRAINT: *the degree of* $P\big(z, A(N_1(z)), \ldots, A(N_d(z))\big)$ *is at most* $|\mathbb{F}|/4$*, and*
2. SATISFIABILITY CONSTRAINT: *for every* $\gamma \in H$ *it holds that* $P\big(\gamma, A(N_1(\gamma)), \ldots, A(N_d(\gamma))\big) = 0$.

An instance $(\mathbb{F}, H, \vec{N}, P)$ is efficiently represented as follows: the finite field $\mathbb{F}$ is specified via an irreducible polynomial over $\mathbb{F}_2$ of degree $\log |\mathbb{F}|$; the $\mathbb{F}_2$-linear subspace H via a basis $(\gamma_1, \ldots, \gamma_{\log |H|})$ of elements in $\mathbb{F}$; and the polynomials $N_1, \ldots, N_d, P$ via $\mathbb{F}$-arithmetic circuits.

Formal statements for parts (i), (ii), (iii)..

As described earlier, part (i) in our proof of Theorem 1 is an efficient reduction from each of the languages $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ to sICSP. Formally:

Proposition 3.4. *There exist linear-time reductions* $f_{\mathsf{TM}}, f_{\mathsf{CSAT}}, f_{\mathsf{RAM}} \colon \{0,1\}^* \to \{0,1\}^*$ *respectively from* $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ *to* sICSP *with the following efficiency.*

language	*reduction*	*color bit length*	*size of constraint circuit*	*degree of constraint circuit*
$\mathcal{L}_{\mathsf{TM}}$	$(M,T) \stackrel{f_{\mathsf{TM}}}{\longmapsto} (O(T), c, K)$	$c = O(\log T)$	$\mathsf{size}(K) = O(\mathsf{size}(M) + \log T)$	$\mathsf{deg}(K) = O(\mathsf{deg}(M) + \log T)$
$\mathcal{L}_{\mathsf{CSAT}}$	$(M,T) \stackrel{f_{\mathsf{CSAT}}}{\longmapsto} (O(T), c, K)$	$c = O(\log T)$	$\mathsf{size}(K) = O(\mathsf{size}(M) + \log T)$	$\mathsf{deg}(K) = O(\mathsf{deg}(M) + \log T)$
$\mathcal{L}_{\mathsf{RAM}}$	$(M,T) \stackrel{f_{\mathsf{RAM}}}{\longmapsto} (O(T), c, K)$	$c = O(\log T)$	$\mathsf{size}(K) = O(\mathsf{size}(M) + \log T)$	$\mathsf{deg}(K) = O(\mathsf{deg}(M) + \log T)$

Part (ii) in our proof of Theorem 1 is an efficient reduction from sICSP to sACSP. As mentioned, part (ii) is this paper's technical contribution, and is the "core" of our arithmetization. Formally:

Lemma 1 (Main Lemma)**.** *There is a polynomial-time reduction* $f \colon \{0,1\}^* \to \{0,1\}^*$ *from* sICSP *to* sACSP *such that, for all* (T, c, K)*, letting* $\big(\mathbb{F}, H, (N_i)_{i=1}^{d}, P\big) = f(T, c, K)$*, it holds that each* N_i *has degree* 1*,* $d = O\big(\log T + \mathsf{size}(K)\big)$*, and* $|\mathbb{F}| = O(T) \cdot \min\big\{\ \log T + c + \mathsf{size}(K)\ ,\ (\log T + c) \cdot \mathsf{deg}(K)\ \big\}$*. (In particular, the instance* $x = (\mathbb{F}, H, (N_i)_{i=1}^{d}, P)$ *can be specified in* $\mathrm{poly}(\mathsf{size}(K) + \log T)$ *bits.)*

Part (iii) in our proof of Theorem 1 is a construction of a code PCP for sACSP. Formally:

Proposition 3.5. *There exists a code PCP* $(V, \vec{C})$ *for* sACSP *such that* $(V, \vec{C})$ *has proof length* $O(|\mathbb{F}| \log |\mathbb{F}|)$*, query complexity* $O(d)$*, and verifier complexity* $\mathrm{poly}(|x|)$ *for any instance* $x = (\mathbb{F}, H, (N_i)_{i=1}^{d}, P)$.

It is easy to verify that our Theorem 1 follows by combining Proposition 3.4, Lemma 1, and Proposition 3.5.

Where are the proofs?. Lemma 1 is the technical contribution of this paper, and we discuss the ideas that go into its proof in the next section (which also contains pointers the sections with the full proof). For completeness: in Appendix B we sketch the proof of Proposition 3.4 (the proof relies on techniques from Ben-Sasson et al. [BCGT13a] and Polishchuk and Spielman [PS94]); and in Appendix C we give the proof of Proposition 3.5 (the proof relies on familiar algebraic techniques, e.g., see [BS08]).

4 High-Level Proof of Main Lemma

We describe, at high-level, the proof of our main lemma (Lemma 1). We want to construct a polynomial-time reduction from sICSP to sACSP, mapping a sICSP instance (T, c, K) to a sACSP instance $(\mathbb{F}, H, \vec{N}, P)$. The main efficiency goal is to *minimize the size of the field* $\mathbb{F}$ *in the* sACSP *instance*, because (per Proposition 3.5) proof length of the corresponding code PCP is $O(|\mathbb{F}| \log |\mathbb{F}|)$. Throughout, we assume familiarity with the definition of the languages sICSP (Definition 3.1) and sACSP (Definition 3.3).

In our description, we simultaneously describe how to construct $(\mathbb{F}, H, \vec{N}, P)$ from (T, c, K), as well as how to construct a witness A for $(\mathbb{F}, H, \vec{N}, P)$ from a witness (χ, π_1, π_2) for (T, c, K). From the description, it is clear that the witness construction is reversible, so that $(T, c, K) \in$ sICSP iff $(\mathbb{F}, H, \vec{N}, P) \in$ sACSP.

From a very high-level point of view, we use the assignment polynomial A to encode (χ, π_1, π_2), and define the constraint polynomial P and neighbor polynomials $\vec{N}$ so that if the polynomial $P\big(z, A(\vec{N}(z))\big)$ vanishes for every $z \in H$ then A encodes a valid witness for the instance (T, c, K). Intuitively, the main technical challenge is that A cannot "just" efficiently encode the witness (χ, π_1, π_2), but must encode the witness in a special way so that its validity can be established via the function P that, being a polynomial of bounded degree, cannot be "too complicated" of a function. This is the core of the arithmetization: we must translate, efficiently, a purely combinatorial CSP into a CSP with *very restrictive algebraic properties*.

We begin by observing that any assignment $\tilde{A} \colon H \to \mathbb{F}$ can be interpolated to a univariate polynomial $A \colon \mathbb{F} \to \mathbb{F}$ of degree no more than $|H|$. Therefore, in the rest of the discussion, we interchangeably view A both as a polynomial and as an arbitrary function from H to $\mathbb{F}$. In particular, our choice of P and $\vec{N}$ will be so that, for any $z \in H$, the value of $P\big(z, A(\vec{N}(z))\big)$ does not depend on $A(\gamma)$ for any $\gamma \notin H$.

In order to minimize the size of the field $\mathbb{F}$,

it suffices to minimize the degree of the constraint polynomial $P\big(z, A(\vec{N}(z))\big)$

because $|\mathbb{F}|$ can be taken to be 4 times this degree (see Definition 3.3). As we will use only neighbor polynomials $\vec{N}$ that are linear, the final degree of $P\big(z, A(\vec{N}(z))\big)$ is (essentially) the multiplication of $|H|$ (which bounds the degree of A) and the total degree of P.

Techniques from previous work give a reduction to sACSP where H has size $\Theta(T \cdot \log T \cdot c)$ and P had total degree $\Theta(\mathsf{deg}(K))$ [BCGT13a]. In this paper, we develop three new techniques: one technique for reducing the size of H, and two for reducing the total degree of P.

Technique #1: block routing. The goal of our first technique is to reduce the size of the linear subspace H. We provide here the intuitive idea behind the technique, from an information-theoretic perspective.

The coloring $\chi \colon [T] \to \{0,1\}^c$ is an sICSP witness, and requires $\Theta(T \cdot c)$ bits to specify. All previous work (aiming at a quasilinear proof length) incurred a $\log T$ overhead in the number of witness bits, due to the use of a routing network for "structuring" the constraint satisfaction problem at hand. We too incur this overhead, and have $\Theta(T \cdot \log T \cdot c)$ witness bits that we need to encode into the assignment polynomial A.

Storing blocks of bits into a single field element was considered problematic in prior work. To retrieve a bit from a block of bits stored in a field element γ, the constraint polynomial P must apply a *projection*

operation. But such projections are very expensive: they have degree $|H| = \Omega(T)$. Thus, if P were to use projections, the degree of $P\big(z, A(\vec{N}(z))\big)$ would blow up to $\Omega(T^2)$, causing $|\mathbb{F}| = \Omega(T^2)$. (Recall that A has degree $|H| = \Omega(T)$.) As a result, prior work encoded witness bits inefficiently by storing only a single bit per field element (i.e., choosing $A(\gamma) \in \{0,1\}$ for a $\gamma \in H$). Doing so requires H of size $\Omega(T \cdot \log T \cdot c)$.

In contrast, we reduce the size of H to only $O(T \cdot (\log T + c))$ by using a more compact encoding of witness bits, and *do not* require P to use any high-degree bit projections. The main observation is that it suffices for P to access as bits only a $(\frac{1}{\log T} + \frac{1}{c})$-fraction of the witness, and access the rest as blocks of bits. Indeed, these other bits (allegedly) encode, e.g., routing information, and their correctness can be verified via simple checks operating on blocks of bits. Thus, we let A store only a $(\frac{1}{\log T} + \frac{1}{c})$-fraction of the witness as bits, and the rest as $\log |\mathbb{F}| = \Omega(\log T)$ bits per field element. Overall, A relies on only $O(T \cdot (\log T + c))$ field elements to store all witness bits.

The above intuition does not touch on several details (e.g., ensuring that that different parts of the witness stored in different ways are consistent with one another), which we address in the full proof.

Technique #2: degree reduction via nondeterministic guesses. The goal of our second technique is to reduce the total degree of the constraint polynomial P. Because P needs to (at least) evaluate the boolean circuit K, the total degree of P in prior work was $O(\mathsf{deg}(K))$. While this appears necessary, we avoid this via a *degree reduction*.

We proceed as follows. For each $v \in [T]$, we need to verify that $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))) = 0$. For each $v \in [T]$, we introduce $\mathsf{size}(K)$ *non-deterministic bits* corresponding to the wires in the computation of $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v)))$. We can store these additional $O(T \cdot \mathsf{size}(K))$ witness bits in A, by increasing the size of H from $O(T \cdot (\log T + c))$ to $O(T \cdot (\log T + c + \mathsf{size}(K)))$.

Then, instead of constructing P to compute K in one shot, we construct P to separately verify correct computation of *each* of the $\mathsf{size}(K)$ gates in K. The technical details are quite involved (due to the many algebraic requirements imposed by the language sACSP), and we defer these to the full proof. But the crucial point here is that the additional witness bits allow us to reduce the task of computing a circuit K (which has degree $\mathsf{deg}(K)$) to computing $\mathsf{size}(K)$ constant-degree polynomials. As we shall see in the next technique, it is advantageous for us to do so because we construct P so that its total degree only depends on the maximum degree of the $\mathsf{size}(K)$ polynomials, which is constant, in contrast to $O(\mathsf{deg}(K))$ in prior work.

Technique #3: further degree reduction via constraint bundling. Applying Technique #1 and #2 yields T identical sets of constant-degree polynomials; each set contains $m := O(\log T + c + \mathsf{size}(K))$ polynomials. Collectively, these T sets of polynomials induce T sets of constraints on the assignment polynomial A. We need to construct P to check all the polynomials in a set, without blowing up the total degree of P.

Our final technique is to apply known *constraint bundling* techniques, based on *selector polynomials* [BS08, BGH+05, BCGT13a], in order to further reduce the degree of P. Namely, we construct P to verify the "logical AND" of all the polynomials in a set, while only paying for the *maximum* of their degrees, rather than the *sum* of their degrees. Selector polynomials have been used before in arithmetizations for other goals, and ours can be viewed as a new application of selector polynomials: amortizing the cost, in degree, of verifying many constraints.

We illustrate the idea via a simple example. Suppose we have two polynomials $P_1(z, A(z + a_1))$ and $P_2(z, A(z + a_2))$ in z, and we need to verify that both vanish at every point $\gamma \in S$ for a subset S of H.

Let $R_{\text{AND}}(x, y) \stackrel{\text{def}}{=} xy + x + y$, and note that for any $\gamma_1, \gamma_2 \in \{0,1\}$ it holds that $R_{\text{AND}}(\gamma_1, \gamma_2) = 0$ if and only if $\gamma_1 = \gamma_2 = 0$. Thus (if there is a way to ensure that P_1, P_2 are boolean valued), a simple method of checking if both P_1 and P_2 vanish on S would be to construct P and (affine) $\vec{N}$ so that

$$P\big(z, A(\vec{N}(z))\big) = R_{\text{AND}}\big(P_1(z, A(z + a_1)), P_2(z, A(z + a_2))\big)$$

and check if $P\big(\gamma, A(\vec{N}(\gamma))\big) = 0$ for all $\gamma \in S$. However, the degree of $P\big(z, A(\vec{N}(z))\big)$ is at least $\big(\mathsf{deg}(P_1) + \mathsf{deg}(P_2)\big) \cdot \mathsf{deg}(A)$; thus, the degree depends on the sum of the total degrees of P_1 and P_2.

In general, with more than two polynomials, the total degree of P depends on *the sum of total degrees*.

We construct P differently. Suppose there is a constant θ such that S and $U \stackrel{\text{def}}{=} S + \theta$ are disjoint and $U \subset H$. (As we will discuss later, in our proof this technical requirement can be easily satisfied.) Let $Y_{H,S}(z)$ be the polynomial that is 1 in S and 0 in $H \setminus S$ (and arbitrary outside H); the polynomial $Y_{H,S}(z)$ is called the *selector polynomial* for S with respect to H [BS08, BGH$^+$05, BCGT13a]. It is easy to see, by interpolation, that the degree of $Y_{H,S}(z)$ is at most $|H|$. We construct P and (affine) $\vec{N}$ so that

$$P\big(z, A(\vec{N}(z))\big) = Y_{H,S}(z) \cdot P_1\big(z, A(z + a_1)\big) + Y_{H,S}(z + \theta) \cdot P_2\big(z + \theta, A(z + \theta + a_2)\big)$$

and verify if $P\big(\gamma, A(\vec{N}(\gamma))\big) = 0$ for every $\gamma \in S \cup U$. Because the characteristic of $\mathbb{F}$ is 2, at each $\gamma \in S$ only P_1 is "activated"; and at each $\gamma \in U = S + \theta$ only P_2 is "activated". In sum, $P\big(\gamma, A(\vec{N}(\gamma))\big)$ vanishes on $S \cup U$ if and only if P_1 and P_2 vanish on S. (Also note that we no longer rely on P_1, P_2 outputting boolean values.) Furthermore, the degree of $P\big(z, A(\vec{N}(z))\big)$ is at most $|H| + \max\{\mathsf{deg}(P_1), \mathsf{deg}(P_2)\} \cdot \mathsf{deg}(A)$. Because $\mathsf{deg}(A)$ at worst equals $|H|$, we only pay for the *maximum* total degree of P_1 and P_2, rather than the sum as before.

In summary, the example above shows that if we have two constraint polynomials P_1 and P_2 to check on S, we can *relocate* one of them to U, an affine shift of S, and pay less total degree.

In general, when we have more than two polynomials to check on a subset S of H, we can relocate each polynomial to a new affine shift of S in H, and use selector polynomials to check each of the polynomials separately. This allows us to pay for the maximum of the degrees rather than their sum — and this is a big saving when the polynomials' degrees are constant and we have many polynomials to check, as in our case.

Of course, H needs to be large enough to allow for all the necessary disjoint affine shifts. This is not a problem in our proof, because $|S| \approx T$ and $|H| = O(T \cdot (\log T + c + \mathsf{size}(K)))$; hence, H has enough space for $m = O(\log T + c + \mathsf{size}(K))$ shifts of S. Thus, we can use these m shifts of S to relocate each set of m constant-degree polynomials produced by our Technique #1 and #2.

Lastly, prior work implies that P can be computed in polylogarithmic time: previous work showed how to evaluate a selector polynomial $Y_{H,S}(z)$ with only $(\log |H|)^{O(1)}$ field operations when S is an $\mathbb{F}_2$-linear subspace of H (which we ensure in our proof), despite $Y_{H,S}$ being a polynomial of degree $|H|$ that is not sparse [BCGT13a]. This fact follows by suitably leveraging computational properties of *linearized polynomials* [LN97, Section 2.5] in finite field extensions of $\mathbb{F}_2$ [BGH$^+$05].

4.1 Proof of Lemma 1: Roadmap

When combined, the aforementioned techniques yield the field size claimed in Lemma 1. We provide a roadmap to the technical sections containing the proof of Lemma 1. First, we shall assume throughout, without loss of generality, that T is a power of 2: it equals 2^t for some $t \in \mathbb{Z}_+$.

We recall that we want to construct a polynomial-time reduction from sICSP to sACSP, mapping a sICSP instance $(2^t, c, K)$ to a sACSP instance $(\mathbb{F}, H, \vec{N}, P)$. Our goal is to *minimize the size of the field* $\mathbb{F}$ *in the* sACSP *instance*. Throughout, we assume familiarity with the definition of the languages sICSP (Definition 3.1) and sACSP (Definition 3.3).

Let us first discuss how the field $\mathbb{F}$ and linear subspace H in the sACSP instance are specified.

- The language sACSP requires working over finite fields of characteristic 2, so we let $\mathbb{F} = \mathbb{F}_{2^f}$ for an $f \in \mathbb{Z}_+$ that we specify at the end of the proof so to satisfy the sACSP degree constraint (see Definition 3.3).
- The subspace H is chosen to be any $(t + \ell + s)$-dimensional $\mathbb{F}_2$-linear subspace containing a special subspace H_0 of dimension $t + \ell$ (which we shall discuss in a moment), for ℓ such that $4t \leq 2^\ell < 8t$ and for some $s \in \mathbb{Z}_+$ that we also specify at the end of the proof. (Intuitively, s will be chosen to be just large enough for us to encode all the necessary witness bits in the assignment polynomial $A\colon H \to \mathbb{F}$.) The subspace H_0 of dimension $t+\ell$ is the vertex set of an *affine graph* that is the embedding of a t*-dimensional de Bruijn graph* (a routing network with convenient algebraic properties).

The rest of the proof focuses first on giving technical preliminaries, which define the subspace H_0 and specify the properties we need from the de Bruijn graph (and its embedding), and then on constructing the neighbor polynomials $\vec{N}$ and the constraint circuit P. Along the way, we shall discuss why our choices of $\vec{N}$ and P are good. More precisely, we proceed as follows.

Technical preliminaries (Section 5). We give the definition of a t-dimensional de Bruijn graph and explain its routing properties. We specify in what sense such a graph can be embedded in an affine graph, and also define the subspace H_0. This material is not new, and our contribution lies in the following steps.

Step 1 (Section 6). We describe how to convert an sICSP witness (χ, π_1, π_2) into an sACSP assignment polynomial $A\colon \mathbb{F} \to \mathbb{F}$, using an "encoding strategy" that is more efficient than in prior work. We also specify constraints on A that, if satisfied, imply that A encodes a witness (χ, π_1, π_2) satisfying the constraint circuit K. Step 1 uses our Technique #1 (see Section 4) of *block routing*.

Step 2 (Section 7). We describe how to construct $\vec{N}$ and P to verify the aforementioned set of constraints with a field size of $O(2^t) \cdot \big((t+c) \cdot \mathsf{deg}(K)\big)$ — this already yields the second term in the field size claimed in Lemma 1. (The first term will be achieved by the next step.) Informally, we describe how to efficiently "bundle" all the constraints for A into a constraint polynomial P of total degree $O(\mathsf{deg}(K))$. Step 2 uses our Technique #3 (see Section 4) based on *selector polynomials*.

Step 3 (Section 8). Finally, we describe how to construct $\vec{N}$ and P to verify the same set of constraints, but this time with a field size of $O(2^t) \cdot \big(t+c+\mathsf{size}(K)\big)$ — this yields the first term in the field size claimed in Lemma 1. Step 3 uses our Technique #2 (see Section 4) based on *degree reduction*.

5 Proof of Lemma 1: Technical Preliminaries

We introduce definitions and facts necessary for the full proof of Lemma 1.

Routing on de Bruijn graphs. A *de Bruijn graph* [Lei92, PS94, BS08, BGH$^+$05, BCGT13a] is a routing network. Informally, a t-dimensional de Bruijn graph is a directed 2-regular graph consisting of $O(t)$ columns, each containing 2^t vertices, such that, for any permutation $\pi\colon [2^t] \to [2^t]$, there are 2^t vertex-disjoint paths for which the i-th path connects (or *routes*) the i-th vertex in the first column to the $\pi(i)$-th vertex in the last column. At each column, a packet is sent to either of its two neighbors in the next column.

Definition 5.1. *Let $t \in \mathbb{Z}_+$. Define ℓ to be the integer such that $4t \le 2^\ell < 8t$ and* $\textsc{Shift}\colon \{0,1\}^t \to \{0,1\}^t$ *to be the function that cyclically shifts left the input by 1 bit. The t-*dimensional de Bruijn graph DB_t *is the graph whose vertex set is*

$$V^{\mathsf{DB}_t} \stackrel{\text{def}}{=} \big\{(i-1, v) : i \in \{1, \ldots, 2^\ell - 1\},\ v \in \{0,1\}^t\big\}$$

and whose edge set is the one induced by the following two neighbor functions

$$\Gamma_1^{\mathsf{DB}_t}\big((i-1,v)\big) \stackrel{\text{def}}{=} \big(i \bmod (2^\ell - 1),\ \textsc{Shift}(v)\big)\ ,$$
$$\Gamma_2^{\mathsf{DB}_t}\big((i-1,v)\big) \stackrel{\text{def}}{=} \big(i \bmod (2^\ell - 1),\ \textsc{Shift}(v) \oplus 1\big)\ .$$

For $i \in [2^\ell - 1]$, the i-th column *of the graph is the subset of vertices $V_i^{\mathsf{DB}_t} \stackrel{\text{def}}{=} \{(i-1, v)\}_{v \in \{0,1\}^t}$; each vertex in $V_i^{\mathsf{DB}_t}$ is connected to exactly two vertices in the next column. We call $(i-1, v)$ the v-*th element *in column $V_i^{\mathsf{DB}_t}$; note that we can view v as a string in $\{0,1\}^t$ or an integer in $[2^t]$.*

A routing on a de Bruijn graph can be interpreted from a perspective of *coloring constraints*. Specifically, a routing of c-bit packets on a t-dimensional de Bruijn graph according to a permutation $\pi\colon [2^t] \to [2^t]$ can be interpreted as a pair of functions $\chi^{\mathsf{data}}\colon V^{\mathsf{DB}_t} \to \{0,1\}^c$ and $\chi^{\mathsf{bit}}\colon V^{\mathsf{DB}_t} \to \{0,1\}$. Since $V_1^{\mathsf{DB}_t}$ is the first column of the de Bruijn graph, $\chi := \chi^{\mathsf{data}}|_{V_1^{\mathsf{DB}_t}}$ is a function specifying the 2^t packets to be routed. Allegedly, for each $(i,v) \in V^{\mathsf{DB}_t}$, $\chi^{\mathsf{data}}(i,v)$ is the packet traveling through vertex (i,v), and $\chi^{\mathsf{bit}}(i,v)$ is the *routing bit* associated with the vertex (i,v); the routing bit indicates whether the packet should be routed

on the first or second edge going to the next column. Crucially, it is possible to ensure that χ^{data} and χ^{bit} correspond to *some* routing, by only checking simple coloring constraints, as in the following lemma.

Lemma 5.2. *The following two statements hold.*

1. *From permutation to coloring. For any function $\chi\colon [2^t] \to \{0,1\}^c$ and permutation $\pi\colon [2^t] \to [2^t]$ there are functions $\chi^{\mathsf{data}}\colon V^{\mathsf{DB}_t} \to \{0,1\}^c$ and $\chi^{\mathsf{bit}}\colon V^{\mathsf{DB}_t} \to \{0,1\}$ such that:*
   (a) for every v in $[2^t]$, $\chi^{\mathsf{data}}(0,v) = \chi(v)$ and $\chi^{\mathsf{data}}(2^\ell - 2, v) = \chi(\pi(v))$;
   (b) for every vertex (i,v) in V^{DB_t} that is not in the last column,
   if $\chi^{\mathsf{bit}}(i,v) = 0$ then $\chi^{\mathsf{data}}(i,v) = \chi^{\mathsf{data}}(\Gamma_1^{\mathsf{DB}_t}(i,v))$, and
   if $\chi^{\mathsf{bit}}(i,v) = 1$ then $\chi^{\mathsf{data}}(i,v) = \chi^{\mathsf{data}}(\Gamma_2^{\mathsf{DB}_t}(i,v))$.
2. *From coloring to permutation. For any functions $\chi^{\mathsf{data}}\colon V^{\mathsf{DB}_t} \to \{0,1\}^c$ and $\chi^{\mathsf{bit}}\colon V^{\mathsf{DB}_t} \to \{0,1\}$ satisfying Item (b) above, if χ^{data} is injective on $V_1^{\mathsf{DB}_t}$, then there exist a unique function $\chi\colon [2^t] \to \{0,1\}^c$ and unique permutation $\pi\colon [2^t] \to [2^t]$ satisfying Item (a) above.*

Embedding de Bruijn graphs into affine graphs. De Bruijn graphs are routing networks that enjoy various convenient *algebraic* properties [PS94, BS08, BGH+05, BCGT13a]. One such property is the fact that they can be embedded into "low-degree" graphs, i.e., graphs whose vertices are in a finite field and neighbor functions are low-degree polynomials.[7] It is known how to embed a t-dimensional de Bruijn graph inside an 8-regular low-degree (in fact, even *affine*) graph, over any field $\mathbb{F}$ of characteristic 2 with $|\mathbb{F}| = \Theta(2^t t)$ [BS08, BCGT13a]. We now formally state this embedding, along with additional algebraic properties that we use; for completeness, the proof of the statement is in Section 5.1 below.

Lemma 5.3 ([BCGT13a])**.** *For any $t \in \mathbb{Z}_+$ and field $\mathbb{F}$ of characteristic 2 with $|\mathbb{F}| \geq 2^{t+\ell}$, there is an embedding $\Phi\colon V^{\mathsf{DB}_t} \to \mathbb{F}$ and affine functions $\Gamma_1, \ldots, \Gamma_8\colon \mathbb{F} \to \mathbb{F}$ such that for all $(i,v) \in V^{\mathsf{DB}_t}$, letting $\gamma = \Phi(i,v)$,*

$$\big(\Phi(i,v), \Phi(\Gamma_1^{\mathsf{DB}_t}(i,v))\big) \in \big\{\big(\gamma, \Gamma_j(\gamma)\big)\big\}_{j=1}^{4} \quad \text{and}$$
$$\big(\Phi(i,v), \Phi(\Gamma_2^{\mathsf{DB}_t}(i,v))\big) \in \big\{\big(\gamma, \Gamma_j(\gamma)\big)\big\}_{j=5}^{8} \ .$$

Moreover, letting $\hat{V} \stackrel{\text{def}}{=} \Phi(V^{\mathsf{DB}_t})$ and $\hat{V}_i \stackrel{\text{def}}{=} \Phi(V_i^{\mathsf{DB}_t})$ for $i \in \{1, \ldots, 2^\ell - 1\}$, the following holds.

1. *Linear structure. There exist a t-dimensional $\mathbb{F}_2$-linear subspace $\hat{V}_0$ of $\mathbb{F}$, a $(t+\ell)$-dimensional $\mathbb{F}_2$-linear subspace H_0 of $\mathbb{F}$, and $2^\ell - 1$ constants $\xi_0, \ldots, \xi_{2^\ell - 1} \in \mathbb{F}$ with $\xi_0 = 0$ such that:*

$$\text{(a) } \hat{V}_i = \hat{V}_0 + \xi_i\,, \qquad \text{(b) } \hat{V} = \textstyle\bigcup_{i=1}^{2^\ell-1} \hat{V}_i\,, \qquad \text{(c) } H_0 = \hat{V}_0 \cup \hat{V}\,.$$

   Thus, $\hat{V}_0, \ldots, \hat{V}_{2^\ell-1}$ partition H_0. Also, $\Phi(i,v) = \sum_{j=1}^{t} v_j a_j + \xi_i$ where $a_1, \ldots, a_t$ is a basis of $\hat{V}_0$.

2. *Finding "true" neighbors. For every $\gamma = \Phi(i,v) \in \hat{V}$, there is a unique index $i(\gamma) \in [4]$ satisfying*

$$\big(\big(\Phi(i,v), \Phi(\Gamma_1^{\mathsf{DB}_t}(i,v))\big) = \big(\gamma, \Gamma_{i(\gamma)}(\gamma)\big)\ ,$$
$$\big(\big(\Phi(i,v), \Phi(\Gamma_2^{\mathsf{DB}_t}(i,v))\big) = \big(\gamma, \Gamma_{4+i(\gamma)}(\gamma)\big)\ .$$

   In addition, there exists a polynomial $\hat{\Gamma}\colon \mathbb{F}^{1+4} \to \mathbb{F}$ such that $\hat{\Gamma}(\gamma, \beta_1, \ldots, \beta_4) = \beta_{i(\gamma)}$ for all $\gamma \in \hat{V}$; moreover, $\hat{\Gamma}$ is of degree $2|H_0|$ in the first variable and linear in the other 4 variables.

Remark 5.4. Since $\hat{V}_0$ is an $\mathbb{F}_2$-linear subspace, there is a canonical order on its elements: the one induced by the basis for $\hat{V}_0$. Similarly for $\hat{V}_i$ (which is a shift of $\hat{V}_0$) and H_0. Thus, we can talk about the *v-th element* of $\hat{V}_i$ for $v \in [2^t]$.

Finally, an analogue of Lemma 5.2 holds in the algebraic setting: a routing of c-bit packets (χ, π) on the *embedded* de Bruijn graph can be thought as functions $\chi^{\mathsf{data}}\colon \hat{V} \to \{0,1\}^c$ and $\chi^{\mathsf{bit}}\colon \hat{V} \to \{0,1\}$.

[7] The requirement of working with low-degree graphs is quite restrictive. For instance, Beneš networks [Ben65, Wak68, OTW71, Lei92] are a more common type of routing network than de Bruijn graphs, but it is not known how to embed a t-dimensional Beneš network for 2^t packets into a low-degree graph over a finite field of size $O(2^t t)$.

5.1 Proof of Lemma 5.3

Let V^{DB_t} be the vertex set of the t-dimensional de Bruijn graph, and let $\Gamma_1^{\mathsf{DB}_t}$ and $\Gamma_2^{\mathsf{DB}_t}$ be its two neighbor functions. (See Definition 5.1.) Let $f \in \mathbb{Z}_+$ with $f \geq t + \ell$, and let $\mathbb{F}_{2^f} = \mathbb{F}_2[X]/P(X)$, where $P(X) \in \mathbb{F}_2[X]$ is a monic irreducible polynomial of degree f. Let $\Xi(X) \in \mathbb{F}_2[X]$ be a primitive polynomial of degree ℓ, and define $\xi_i(X) \stackrel{\text{def}}{=} X^{i-1} \bmod \Xi(X)$ for $i \in [2^\ell - 1]$. It can be shown that $\xi_i \equiv \xi_j \pmod{\Xi(X)}$ if and only if $i \equiv j \pmod{2^\ell - 1}$ (see [BS08, BCGT13a]). As a consequence, $\{\xi_1(X), \xi_2(X), \ldots, \xi_{2^\ell-1}(X)\} = \mathsf{span}(1, X, \ldots, X^{\ell-1}) \backslash \{0\}$; separately define $\xi_0 \stackrel{\text{def}}{=} 0$. Because $\deg \xi_i(X) \leq \ell - 1$ and $\ell - 1 < f$, we can view each $\xi_i(X)$ as an element in $\mathbb{F}_{2^f}$.

Vertex embedding. Define the map Φ as follows [BS08, BCGT13a]:

$$\forall\, i \in [2^\ell - 1]\,,\ \forall\, v \in \{0,1\}^t\,, \quad (i, v) \mapsto \Phi(i, v) \stackrel{\text{def}}{=} \left(\sum_{j=1}^{t} v_j X^{j-1} \right) \cdot X^\ell + \xi_i(X)\ . \tag{1}$$

Define $\hat{V}_0 \stackrel{\text{def}}{=} \mathsf{span}(X^\ell, X^{\ell+1}, \ldots, X^{\ell+t-1})$; $\hat{V}_0$ is a t-dimensional $\mathbb{F}_2$-linear subspace of $\mathbb{F}_{2^f}$. For $i \in [2^\ell - 1]$, define $\hat{V}_i = \hat{V}_0 + \xi_i$. Note that $\hat{V}_i = \Phi(V_i^{\mathsf{DB}_t})$ and $\hat{V} = \Phi(V^{\mathsf{DB}_t})$; this is the claimed linear structure.

Edge embedding. Let $(i, v) \in V^{\mathsf{DB}_t}$. We let $i + 1$ denote $\big(i \bmod (2^\ell - 1)\big) + 1$ for notational simplicity. The neighbor function $\Gamma_1^{\mathsf{DB}_t}$ induces the edge $\big((i, v), (i + 1, \textsc{Shift}(v))\big)$. Note that:

$$\begin{aligned}
\Phi(i + 1, \textsc{Shift}(v)) &= \left(\sum_{j=1}^{t-1} v_j X^j + v_t \right) \cdot X^\ell + \xi_{i+1}(X) \\
&= \left(\sum_{j=1}^{t} v_j X^j \right) \cdot X^\ell + \big(\xi_i(X) X + b\Xi(X)\big) \\
&= \Phi(i, v) \cdot X + (X^\ell + X^{t+\ell}) v_t + b\Xi(X)\ ,
\end{aligned} \tag{2}$$

where $b \in \{0, 1\}$ is such that $\xi_{i+1}(X) = \xi_i(X)X + b\Xi(X)$. Similarly, the neighbor function $\Gamma_2^{\mathsf{DB}_t}$ induces the edge $\big((i, v), (i + 1, \textsc{Shift}(v) \oplus 1)\big)$. Note that:

$$\Phi(i + 1, \textsc{Shift}(v) \oplus 1) = \Phi(i, v) \cdot X + (X^\ell + X^{t+\ell}) v_t + b\Xi(X) + X^\ell\ , \tag{3}$$

where $b \in \{0, 1\}$ is such that $\xi_{i+1}(X) = \xi_i(X)X + b\Xi(X)$. Thus, we let the affine neighbors $\Gamma_1(z), \ldots, \Gamma_8(z)$ be defined as

$$\Gamma_{b_3 \cdot 4 + b_2 \cdot 2 + b_1 + 1}(z) \stackrel{\text{def}}{=} X \cdot z + (X^\ell + X^{t+\ell}) b_1 + \Xi(X) b_2 + X^\ell b_3\ .$$

The above definition ensures that for all $(i, v) \in V^{\mathsf{DB}_t}$, letting $\gamma = \Phi(i, v)$,

$$\big(\Phi(i, v), \Phi(\Gamma_1^{\mathsf{DB}_t}(i, v))\big) \in \big\{\big(\gamma, \Gamma_j(\gamma)\big)\big\}_{j=1}^{4}\ , \text{ and } \big(\Phi(i, v), \Phi(\Gamma_2^{\mathsf{DB}_t}(i, v))\big) \in \big\{\big(\gamma, \Gamma_j(\gamma)\big)\big\}_{j=5}^{8}\ .$$

Finding "true" neighbors. Because $H_0 = \hat{V}_0 \cup \hat{V} = \mathsf{span}(1, X, \ldots, X^{t+\ell-1})$, any $\gamma \in H_0$ can be uniquely written as $\gamma = \sum_{p=0}^{t+\ell-1} b_p X^p$ with $b_p \in \{0, 1\}$. Let $\Pi_p \colon \mathbb{F} \to \mathbb{F}$ be the "projection polynomial" that, given $\gamma \in H_0$, outputs b_p; the degree of Π_p is $|H_0| = 2^{t+\ell}$.

From Eq. (1), we deduce that, for $\gamma = \Phi(i, v)$, it holds that $v_t = b_{t+\ell-1} = \Pi_{t+\ell-1}(\gamma)$ and $b = b_{\ell-1} = \Pi_{\ell-1}(\gamma)$. From Eq. (2) and Eq. (3), we deduce that $i(\gamma) = 2b + v_t + 1 = 2\Pi_{\ell-1}(\gamma) + \Pi_{t+\ell-1}(\gamma) + 1$. Thus we can define the polynomial $\hat{\Gamma}$ as follows:

$$\begin{aligned}
\hat{\Gamma}(z, \beta_1, \beta_2, \beta_3, \beta_4) \stackrel{\text{def}}{=}\ & (1 + \Pi_{t+\ell-1}(z)) \cdot (1 + \Pi_{\ell-1}(z)) \cdot \beta_1 \\
& + (1 + \Pi_{t+\ell-1}(z)) \cdot \Pi_{\ell-1}(z) \cdot \beta_2 \\
& + \Pi_{t+\ell-1}(z) \cdot (1 + \Pi_{\ell-1}(z)) \cdot \beta_3 \\
& + \Pi_{t+\ell-1}(z) \cdot \Pi_{\ell-1}(z) \cdot \beta_4\ .
\end{aligned}$$

Note that $\hat{\Gamma}$ has degree $2|H_0|$ in the first variable, and degree 1 in all the other variables. In addition, for $\gamma = \Phi(i, v) \in V^{\mathsf{DB}_t}$, it satisfies

$$\hat{\Gamma}(\gamma, \Gamma_1(\gamma), \Gamma_2(\gamma), \Gamma_3(\gamma), \Gamma_4(\gamma)) = \Phi(\Gamma_1^{\mathsf{DB}_t}(i, v)) \quad \text{and}$$
$$\hat{\Gamma}(\gamma, \Gamma_5(\gamma), \Gamma_6(\gamma), \Gamma_7(\gamma), \Gamma_8(\gamma)) = \Phi(\Gamma_2^{\mathsf{DB}_t}(i, v)) \ .$$

□

6 Proof of Lemma 1: Step 1 (Constraints For A Color Encoding With Block Routing)

Let DB_t be the t-dimensional de Bruijn graph. From Lemma 5.3, we know that DB_t can be embedded into an 8-regular affine graph over $\mathbb{F}_{2^f}$, provided that $f \geq t + \ell$. (We will pick f so that this is the case.) The vertex set V^{DB_t} of the graph DB_t is mapped to $\hat{V}$, which is contained in H_0. (See Figure 2.)

Let (χ, π_1, π_2) be an witness for the instance sICSP instance $(2^t, c, K)$. By following Lemma 5.2 and the embedding of DB_t from V^{DB_t} to $\hat{V}$, we can route χ according to π_1 in order to obtain functions

$$\chi^{\pi_1,\mathsf{data}} \colon \hat{V} \to \{0,1\}^c \text{ and } \chi^{\pi_1,\mathsf{bit}} \colon \hat{V} \to \{0,1\} \ .$$

Similarly, we can route χ according to π_2 in order to obtain functions

$$\chi^{\pi_2,\mathsf{data}} \colon \hat{V} \to \{0,1\}^c \text{ and } \chi^{\pi_2,\mathsf{bit}} \colon \hat{V} \to \{0,1\} \ .$$

Useful notation. We introduce notation that induces a convenient "coordinate system" in the subspace H. We write $H = \cup_{j=0}^{2^s-1}(H_0 + \theta_j)$ for some 2^s distinct constants $\theta_0, \ldots, \theta_{2^s-1} \in \mathbb{F}$ with $\theta_0 = 0$, and $s \in \mathbb{Z}_+$ to be chosen later. We define $H_j \stackrel{\text{def}}{=} H_0 + \theta_j$ and call H_j the j-th *layer* of H. From Lemma 5.3, $H_0 = \bigcup_{i=0}^{2^\ell-1} \hat{V}_i$ and $\hat{V}_i = \hat{V}_0 + \xi_i$ for 2^ℓ distinct constants $\xi_0, \xi_1, \ldots, \xi_{2^\ell-1} \in \mathbb{F}$ with $\xi_0 = 0$. We define $\hat{V}_{i,j} \stackrel{\text{def}}{=} \hat{V}_i + \theta_j = \hat{V}_0 + \xi_i + \theta_j$ and call $\hat{V}_{i,j}$ the (i,j)-th *column* of H. See Figure 2 for a "layer view" and a "column view" of H.

Encoding and verifying block routings. To encode the information (χ, π_1) in the assignment polynomial A, we define A to store $\chi^{\pi_1,\mathsf{data}}$ on layers $H_0, \ldots, H_{c'-1}$ and to store $\chi^{\pi_1,\mathsf{bit}}$ on layer $H_{c'}$ for some c'. (See Figure 3(a) on page 17.) Specifically, for each $\gamma \in \hat{V}$, we encode $\chi^{\pi_1,\mathsf{data}}(\gamma)$ and $\chi^{\pi_1,\mathsf{bit}}(\gamma)$ as follows.

- We divide the c-bit color $\chi^{\pi_1,\mathsf{data}}(\gamma)$ into $c' \stackrel{\text{def}}{=} \lceil \frac{c}{f} \rceil$ chunks $\chi^{\pi_1,\mathsf{data}}(\gamma)_0, \ldots, \chi^{\pi_1,\mathsf{data}}(\gamma)_{c'-1}$ of f bits each (padding the last chunk with 0's if necessary). Each f-bit chunk $\chi^{\pi_1,\mathsf{data}}(\gamma)_j$ can be identified with the field element of $\mathbb{F}_{2^f}$ whose f coefficients over $\mathbb{F}_2$ are exactly the bits of $\chi^{\pi_1,\mathsf{data}}(\gamma)_j$. For $j \in \{0, \ldots, c'-1\}$, A stores $\chi^{\pi_1,\mathsf{data}}(\gamma)_j$ at location $\gamma + \theta_j \in H_j$, i.e., we choose $A(\gamma + \theta_j) \stackrel{\text{def}}{=} \chi^{\pi_1,\mathsf{data}}(\gamma)_j$.
- Next, we let A store the bit $\chi^{\pi_1,\mathsf{bit}}(\gamma)$ at location $\gamma + \theta_{c'} \in H_{c'}$, i.e., we choose $A(\gamma + \theta_{c'}) \stackrel{\text{def}}{=} \chi^{\pi_1,\mathsf{bit}}(\gamma)$.

To check that the information stored by A in $H_0, \ldots, H_{c'}$ corresponds to *some* routing, it is necessary and sufficient to check that, for any $\gamma \in \hat{V}$ and $j \in \{0, \ldots, c'-1\}$, $A(\gamma + \theta_j)$ is equal to the value assigned by A to one of the two neighbors of $\gamma + \theta_j$, depending on the corresponding routing bit $A(\gamma + \theta_{c'})$. (See Item (1b) in Lemma 5.2.) This can be done by ensuring that

$$R^{\mathsf{bool}}(\gamma, A) = 0 \text{ for all } \gamma \in H_{c'},^{8} \text{ and}$$
$$R_j^{\mathsf{route}}(\gamma, A) = 0 \text{ for all } \gamma \in H_j \setminus (\hat{V}_{0,j} \cup \hat{V}_{2^\ell-1,j}) \text{ and } j \in \{0, \ldots, c'-1\},$$

[8] Verifying for all $\gamma \in H_{c'+1} \setminus (\hat{V}_{0,c'+1} \cup \hat{V}_{2^\ell-1,c'+1})$ suffices, but it does not hurt to oververify and simplify notation.

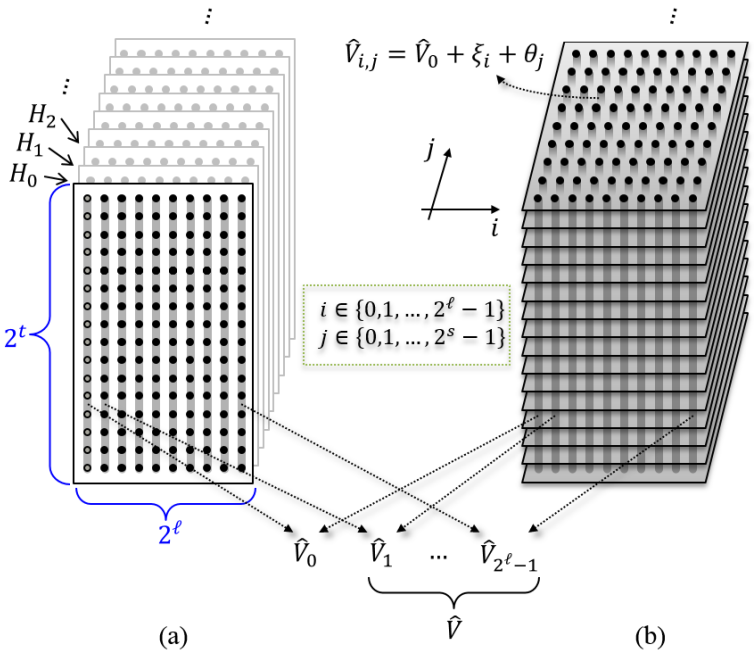

Figure 2: (a) *Layer view* of the $\mathbb{F}_2$-linear subspace H, in which H is partitioned into 2^s layers such that the j-th layer H_j is equal to $H_0 + \theta_j$ for some constant θ_j. (b) *Column view* of the $\mathbb{F}_2$-linear subspace H, in which H is partitioned into $2^\ell 2^s$ columns such that the (i,j)-th *column* $\hat{V}_{i,j}$ is equal to $\hat{V}_0 + \xi_i + \theta_j$ for some constants ξ_i, θ_j.

where R^{bool} is the *boolean check* and R_j^{route} is the j-th *routing check*, which are defined as follows.

$$R^{\mathsf{bool}}(z, A) \stackrel{\text{def}}{=} A(z) \cdot (A(z) + 1)$$

$$R_j^{\mathsf{route}}(z, A) \stackrel{\text{def}}{=} Q\left(A(z), A(z + \theta_j + \theta_{c'}), \hat{\Gamma}\Big(z + \theta_j, A(\Gamma_{1,j}(z)), \ldots, A(\Gamma_{4,j}(z))\Big), \hat{\Gamma}\Big(z + \theta_j, A(\Gamma_{5,j}(z)), \ldots, A(\Gamma_{8,j}(z))\Big)\right)$$

Above,

- $Q(x, b, x_1, x_2) \stackrel{\text{def}}{=} (1 + b)(x + x_1) + b(x + x_2)$;
  (Note that, whenever $b \in \{0, 1\}$, it holds that $Q(x, b, x_1, x_2) = 0$ if and only if $(b = 0 \wedge x = x_1)$ or $(b = 1 \wedge x = x_2)$.)
- $\Gamma_{k,j}(z) \stackrel{\text{def}}{=} \Gamma_k(z + \theta_j) + \theta_j$ for $k \in \{1, \ldots, 8\}$, and $\Gamma_1, \ldots, \Gamma_8$ are the 8 affine functions obtained when embedding the t-dimensional de Bruijn graph (see Lemma 5.3); and
- $\hat{\Gamma}$ is the polynomial that selects the "true" neighbor (see Lemma 5.3).

The above description concludes the discussion of how to encode (χ, π_1) into A by storing information on $H_0, \ldots, H_{c'}$, and then how to verify this information.

To encode the information (χ, π_2) in the assignment polynomial A, we proceed similarly: we define A to store $\chi^{\pi_2,\mathsf{data}}, \chi^{\pi_2,\mathsf{bit}}$ on layers $H_{c'+1}, \ldots, H_{2c'+1}$, define a j-th routing check R_j^{route} for $j \in \{c' + 1, \ldots, 2c'\}$, and then consider analogous boolean and routing checks. (See Figure 3(a) on page 17.)

Furthermore, we also need to ensure that the two routings are about the same χ: for each $\gamma \in \hat{V}_1$, we must have $A(\gamma + \theta_j) = A(\gamma + \theta_{c'+1+j})$ for $j \in \{0, \ldots, c' - 1\}$. To ensure this, we define the check

$$R_j^{\mathsf{cons}}(z, A) \stackrel{\text{def}}{=} A(z) + A(z + \theta_j + \theta_{(c'+1)+j}) \ ,$$

and ensure that $R_j^{\mathsf{cons}}(\gamma, A) = 0$ for all $j \in \{0, 1, \ldots, c' - 1\}$ and $\gamma \in \hat{V}_{1,j}$.

In summary, we need to ensure that an assignment polynomial A satisfies the following constraints:

$$\gamma \in H_{c'} \cup H_{2c'+1} \implies R^{\mathsf{bool}}(\gamma, A) = 0 \ , \tag{C1}$$

$$j \in \{0, 1, \ldots, c'-1, c'+1, \ldots, 2c'\} \, , \, \gamma \in H_j \setminus (\hat{V}_{0,j} \cup \hat{V}_{2^\ell - 1, j}) \implies R_j^{\mathsf{route}}(\gamma, A) = 0 \ , \text{ and} \tag{C2}$$

$$j \in \{0, 1, \ldots, c'-1\} \, , \, \gamma \in \hat{V}_{1,j} \implies R_j^{\mathsf{cons}}(\gamma, A) = 0 \ . \tag{C3}$$

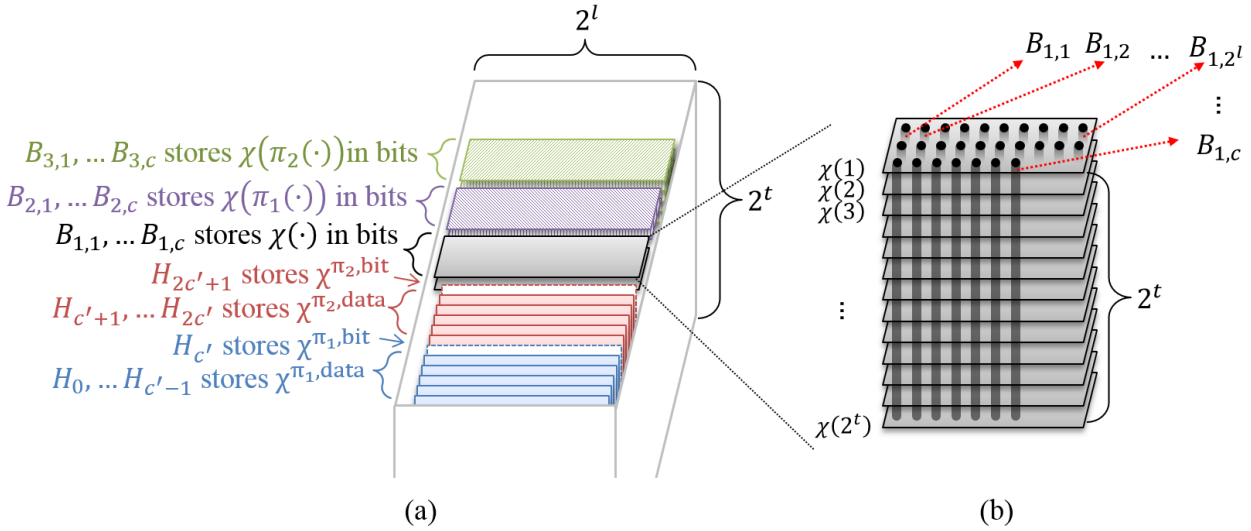

Figure 3: How the assignment polynomial A encodes an sICSP witness (χ, π_1, π_2) in H.

Non-compact encoding of χ. Recall that the constraint circuit K from the sICSP instance $(2^t, c, K)$ needs to access the bits of $\chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))$ for $v \in [2^t]$, as well as v itself. Information about $\chi, \chi(\pi_1(\cdot)), \chi(\pi_2(\cdot))$ is already stored in A: specifically, A stores a compact encoding

- of χ in columns $\hat{V}_{1,0}, \ldots, \hat{V}_{1,c'-1}$,
- of $\chi(\pi_1(\cdot))$ in columns $\hat{V}_{2^\ell-1,0}, \ldots, \hat{V}_{2^\ell-1,c'-1}$, and
- of $\chi(\pi_2(\cdot))$ in columns $\hat{V}_{2^\ell-1,c'+1}, \ldots, \hat{V}_{2^\ell-1,2c'}$.

While these compact encodings are sufficient for the purpose of routing checks (because routing checks do not need access to the bits stored in a field element, but only to the field element itself), they are not good enough for the purpose of checking that K is satisfied. Indeed, retrieving any of the bits stored in a field element $A(\gamma)$ is a high-degree operation that we cannot afford, because it forces us to take the field size to be at least *quadratic* in $T = 2^t$.

Therefore, we additionally store in A the bits of $\chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))$ for $v \in [2^t]$ in a "non-compact" encoding (i.e., one bit per field element). Of course, these non-compact encodings must be consistent with the corresponding compact ones, so we must also introduce constraints to verify this.

Let us first specify how we encode χ in a non-compact way. Consider c shifts of the column $\hat{V}_0$ in sufficiently many new layers of H beyond layer $H_{2c'+1}$ (the last layer we used so far); call these columns $B_{1,1}, \ldots, B_{1,c}$ and let $B_{1,i} = \hat{V}_0 + \theta^\star_{1,i}$ for some constant $\theta^\star_{1,i} \in \mathbb{F}_{2^f}$. (See Figure 3(b) on page 17.) We use these columns to store the bits of χ, in the following sense: for each $v \in [2^t]$, letting $\gamma_v \in \hat{V}_0$ be the v-th element in $\hat{V}_0$ (see Remark 5.4), we define $A(\gamma_v + \theta^\star_{1,i})$ to be the i-th bit of $\chi(v)$ for $i = 1, \ldots, c$.

To verify consistency between the two encodings of χ, we need to ensure that

$$R^{\mathsf{bool}}(\gamma, A) = 0 \text{ for all } \gamma \in B_{1,1} \cup \cdots \cup B_{1,c} \text{ and}$$
$$R^{\mathsf{pack}}_{1,j}(\gamma, A) = 0 \text{ for all } \gamma \in \hat{V}_0 \text{ and } j \in \{0, 1, \ldots, c'-1\},$$

where R^{bool} is the boolean check (which we already defined) and $R^{\mathsf{pack}}_{1,j}$ is the $(1,j)$-th *packing check*, defined as follows:

$$R^{\mathsf{pack}}_{1,j}(z,A) \stackrel{\text{def}}{=} Q'\left(A\big(z+\xi_1+\theta_j\big), A\big(z+\theta^\star_{1,jf+1}\big), \ldots, A\big(z+\theta^\star_{1,(j+1)f}\big)\right) \ .$$

Above,

- $Q'(c,b_0,b_1,\ldots,b_{f-1}) \stackrel{\text{def}}{=} c + \big(b_0e_0 + b_1e_1 + \cdots + b_{f-1}e_{f-1}\big)$ and $(e_0,\ldots,e_{f-1})$ is the $\mathbb{F}_2$-linear basis of $\mathbb{F}_{2^f}$ used for storing bits in a field element; note that, whenever $b_0,\ldots,b_{f-1} \in \{0,1\}$, Q' is zero if and only if c is a compact encoding of the bits $b_0,\ldots,b_{f-1}$.

Also recall that, by construction, $\chi^{\pi_1,\mathsf{data}}(\gamma_v+\xi_1) = \chi(v)$, and A encodes $\chi^{\pi_1,\mathsf{data}}(\gamma_v+\xi_1)$ compactly in c' chunks of f bits each at $A(\gamma_v+\xi_1+\theta_0),\ldots,A(\gamma_v+\xi_1+\theta_{c'-1})$, while non-compactly at $A(\gamma_v+\theta^\star_{1,1}),\ldots,A(\gamma_v+\theta^\star_{1,c})$.

We can similarly encode, in a non-compact way, $\chi(\pi_1(\cdot))$ and $\chi(\pi_2(\cdot))$ in new columns $B_{2,1},\ldots,B_{2,c}$ and $B_{3,1},\ldots,B_{3,c}$ respectively. We can then define corresponding constraints, which include suitably defined polynomials $R^{\mathsf{pack}}_{2,j}$ and $R^{\mathsf{pack}}_{3,j}$. (See Figure 3(a) on page 17.)

Verifying the constraint circuit. We are left to specify constraints to verify the constraint circuit K. First, note that K can also be viewed as an $\mathbb{F}_{2^f}$-arithmetic circuit (via the direct arithmetization of AND and NOT gates). Next, we define $\vec{\Pi}(z)$ to be the collection of t polynomials of degree $|\hat{V}_0| = 2^t$ such that, for every $i \in [t]$ and $v \in [2^t]$, if γ_v is the v-th element in $\hat{V}_0$ (see Remark 5.4), then $\Pi_i(\gamma_v)$ equals the i-th bit of v; such *projection polynomials* can be shown to be computable in $\mathrm{polylog}|\hat{V}_0|$ field operations [BGH+05, BCGT13a]. We then define the following polynomial:

$$R^K(z,A) \stackrel{\text{def}}{=} K\Big(\vec{\Pi}(z), A\big(z+\theta^\star_{1,1}\big),\ldots,A\big(z+\theta^\star_{1,c}\big),\ A\big(z+\theta^\star_{2,1}\big),\ldots,A\big(z+\theta^\star_{2,c}\big)\ A\big(z+\theta^\star_{3,1}\big),\ldots,A\big(z+\theta^\star_{3,c}\big)\Big) \ ,$$

Because we ensured that $A\big(\gamma_v+\theta^\star_{p,1}\big),\ldots,A\big(\gamma_v+\theta^\star_{p,c}\big)$ are the c bits of $\chi(v)$ when $p=1$, of $\chi(\pi_1(v))$ when $p=2$, and of $\chi(\pi_2(v))$ when $p=3$, we deduce that $R^K(\gamma,A)$ is zero for all $\gamma \in \hat{V}_0$ if and only if $K(v,\chi(v),\chi(\pi_1(v)),\chi(\pi_2(v))) = 0$ for all $v \in [2^t]$.

In summary, we have the following constraints for verifying the constraint circuit:

$$\gamma \in \textstyle\bigcup_{p\in\{1,2,3\}\,,\,j\in[c]} B_{p,j} \implies R^{\mathsf{bool}}(\gamma,A) = 0 \ , \tag{C4}$$

$$p \in \{1,2,3\}\,,\ j \in \{0,1,\ldots,c'-1\}\,,\ \gamma \in \hat{V}_0 \implies R^{\mathsf{pack}}_{p,j}(\gamma,A) = 0 \ , \text{ and} \tag{C5}$$

$$\gamma \in \hat{V}_0 \implies R^K(\gamma,A) = 0 \ . \tag{C6}$$

Summary. We need A to store on layers $H_0,\ldots,H_{2c'+1+3\lceil c/2^\ell\rceil}$ all the information required by the constraints we defined. Recalling that $H = \cup_{j=0}^{2^s-1} H_j$, if we choose s so that $2^s \geq 2c'+2+3\lceil \frac{c}{2^\ell}\rceil$, then this can be done. (See Figure 3(a) on page 17.) We can therefore conclude that:

Lemma 6.1. *An instance $(2^t,c,K)$ is in* sICSP *if and only if there exists an assignment polynomial A over $\mathbb{F}_{2^f}$ that satisfies the six constraints* (C1)*,* (C2)*,* (C3)*,* (C4)*,* (C5) *and* (C6)*. Moreover, it suffices to consider polynomials A with $\deg_z A(z) \leq |H|$.*

7 Proof of Lemma 1: Step 2 (Bundling Constraints Via Selector Polynomials)

We describe a constraint polynomial P and sequence of neighbor polynomials $\vec{N}$ such that $P\big(\gamma, A(\vec{N}(\gamma))\big) = 0$ for every $\gamma \in H$ if and only if A satisfies the six constraints (C1), (C2), (C3), (C4), (C5), and (C6). To do so, we *bundle* these constraints, while trying to keep the degree of $P\big(z, A(\vec{N}(z))\big)$ as small as possible.

A naive way to bundle two given constraints is the following: given polynomials $G_1(z)$ and $G_2(z)$, define $R_{\text{AND}}(G_1(z),G_2(z)) \stackrel{\text{def}}{=} G_1(z)G_2(z) + G_1(z) + G_2(z)$. Whenever $G_1(z),G_2(z) \in \{0,1\}$, it

holds that $R_{\text{AND}}(G_1(z), G_2(z)) = 0$ if and only if $G_1(z) = G_2(z) = 0$; in other words, the polynomial $R_{\text{AND}}(G_1(z), G_2(z))$ serves as a "boolean AND" of the polynomials $G_1(z)$ and $G_2(z)$.

However, when it is not the case that $G_1(z), G_2(z) \in \{0, 1\}$, it is *not* true that $R_{\text{AND}}(G_1(z), G_2(z)) = 0$ if and only if $G_1(z) = G_2(z) = 0$. Because some of the constraints we need to bundle are *not* boolean-valued (e.g., (C5)), we need another method. So, instead, we suitably use *selector polynomials*. Given two sets W and S in $\mathbb{F}_{2^f}$ with $S \subseteq W$, the selector polynomial $Y_{W,S}(z)$ is equal to 1 if $z \in S$ and 0 if $z \in W \setminus S$; note that $Y_{W,S}$ has degree $|W|$. It can be shown that $Y_{W,S}$, despite not being sparse, can be computed in $\text{polylog}|W|$ field operations when W and S are $\mathbb{F}_2$-linear subspaces (or shifts of such spaces) [BCGT13a].

For example, to bundle constraints (C1) and (C2), we define the following polynomial:

$$\begin{aligned} P_{\text{C1+C2}}\big(z, A\big) \stackrel{\text{def}}{=} & \Big(Y_{H,H_{c'}}(z) + Y_{H,H_{2c'+1}}(z)\Big) \cdot R^{\mathsf{bool}}(z, A) && \text{(for constraint (C1))} \\ & + \sum_{j \in \{0,1,\ldots,c'-1,c'+1,\ldots,2c'\}} \left(\sum_{i=1}^{2^\ell - 2} Y_{H,\hat{V}_{i,j}}(z)\right) \cdot R_j^{\mathsf{route}}(z, A) && \text{(for constraint (C2))} \end{aligned}$$

Note that $P_{\text{C1+C2}}\big(\gamma, A\big) = 0$ for every $\gamma \in H$ if and only if constraints (C1) and (C2) hold. Also, the degree of $P_{\text{C1+C2}}\big(z, A\big)$ depends on the *maximum* (and not the sum!) of the degrees of $R^{\mathsf{bool}}(z, A)$ and $R^{\mathsf{route}}(z, A)$.

Bundling constraints in this way must be done with care: for correctness, it is necessary that for any $\gamma \in H$ at most one selector polynomial is equal to 1. (For, else, different constraints may cancel each other out even if they are not both equal to 0.) While this is not an issue when bundling constraints (C1) and (C2), this issue does come up, for instance, when bundling the checks in (C5), because (C5) requires more than one check for any $\gamma \in \hat{V}_0$. To address this, we use constant shifts to move conflicting checks to different "locations" in H. In this way, we correctly bundle the checks in constraints (C3), (C4), and (C5) as follows. (See Figure 4(b) on page 21.)

$$\begin{aligned} P_{\text{C3+C4+C5}}\big(z, A\big) \stackrel{\text{def}}{=} & \textstyle\sum_{j=0}^{c'-1} Y_{H,\hat{V}_{0,j}}(z) \cdot R_j^{\mathsf{cons}}(z + \xi_1, A) && \text{(for constraint (C3))} \\ & + \textstyle\sum_{j=0}^{c'-1} Y_{H,\hat{V}_{2^\ell-1,j}}(z) \cdot R_{1,j}^{\mathsf{pack}}(z + \xi_{2^\ell-1} + \theta_j, A) && \text{(for constraint (C5))} \\ & + \textstyle\sum_{j=0}^{c'-1} Y_{H,\hat{V}_{0,c'+1+j}}(z) \cdot R_{2,j}^{\mathsf{pack}}(z + \theta_{c'+1+j}, A) && \text{(for constraint (C5))} \\ & + \textstyle\sum_{j=0}^{c'-1} Y_{H,\hat{V}_{2^\ell-1,c'+1+j}}(z) \cdot R_{3,j}^{\mathsf{pack}}(z + \xi_{2^\ell-1} + \theta_{c'+1+j}, A) && \text{(for constraint (C5))} \\ & + \Big(\textstyle\sum_{p=1}^{3}\sum_{j=1}^{c} Y_{H,B_{p,j}}(z)\Big) \cdot R^{\mathsf{bool}}(z, A)\ . && \text{(for constraint (C4))} \end{aligned}$$

After that, we can similarly define a polynomial $P_{\text{C6}}(z, A)$ for constraint (C6), and ensure that $P_{\text{C1+C2}}(\gamma, A) + P_{\text{C3+C4+C5}}(\gamma, A) + P_{\text{C6}}(\gamma, A) = 0$ for every $\gamma \in H$ if and only if the six constraints are satisfied.

While at first sight the above sum of polynomials may seem to depend in a complex way on the values of A, it is possible to choose a constraint polynomial P and a sequence of affine functions $\vec{N}$ so that $P\big(z, A(\vec{N}(z))\big) = P_{\text{C1+C2}}(z, A) + P_{\text{C3+C4+C5}}(z, A) + P_{\text{C6}}(z, A)$. We will explain how to do this in Step 3 (Section 8); an analogous discussion holds for this step.

For now, let us bound the degree of $P\big(z, A(\vec{N}(z))\big)$. Because A has degree $|H|$ (by interpolation), $P_{\text{C1+C2}}(z, A) + P_{\text{C3+C4+C5}}(z, A)$ has degree at most $2|H_0| + 3|H| = O(|H|)$. However, since $P_{\text{C6}}(z, A)$ forwards values of A to K, $P_{\text{C6}}(z, A)$ has degree $O(|H| \cdot \mathsf{deg}(K))$, where $\mathsf{deg}(K)$ is the total degree of the constraint circuit K. Thus, the degree for $P_{\text{C6}}(z, A)$ dominates in the sum.

By suitably choosing s (see summary at the end of Section 6) and because $2^\ell < 8t$ (see Definition 5.1) and $f \geq t$ (by our eventual choice), it holds that

$$|H| = |H_0| \cdot 2^s \leq |H_0| \cdot 2 \cdot \Big(2c' + 2 + 3\lceil \frac{c}{2^\ell} \rceil\Big) = 2^{t+\ell+1}\Big(2\lceil \frac{c}{f} \rceil + 2 + 3\lceil \frac{c}{2^\ell} \rceil\Big) \leq O(2^t \cdot (t + c))\ .$$

Thus, we deduce that

$$\deg P\big(z, A(\vec{N}(z))\big) = O\big(2^t \cdot (t + c) \cdot \mathsf{deg}(K)\big)\ .$$

Recalling that the field size $|\mathbb{F}_{2^f}| = 2^f$ can be chosen to be a constant factor larger than the degree of $P\big(z, A(\vec{N}(z))\big)$ (see Definition 3.3), we deduce that we have proved Lemma 1 for a field size that is bounded by the second term in the statement's $\min$ expression.

Next, in Step 3 (Section 8), we discuss a different construction of P and $\vec{N}$ for which the degree of $P\big(z, A(\vec{N}(z))\big)$ is $O(2^t \cdot (t + c + \mathsf{size}(K)))$. By choosing which construction of P and $\vec{N}$ to use (namely, from Step 2 or Step 3), we can guarantee a field size that is the minimum between these two bounds, thereby completing the proof of Lemma 1.

8 Proof of Lemma 1: Step 3 (Degree Reduction Via Selector Polynomials & Non-Determinism)

We describe *another* choice of constraint polynomial P and sequence of neighbor polynomials $\vec{N}$ for which $P\big(\gamma, A(\vec{N}(\gamma))\big) = 0$ for every $\gamma \in H$ if and only if A satisfies the six constraints (C1), (C2), (C3), (C4), (C5), and (C6). As before, $P\big(z, A(\vec{N}(z))\big)$ contains the term $P_{\text{C1+C2}}(z, A) + P_{\text{C3+C4+C5}}(z, A)$ for verifying the first five constraints; however, this time we verify constraint (C6) differently.

Consider the constraint circuit K; it has $\mathsf{size}(K)$ gates. Instead of verifying the evaluation of these gates all at once (as done by $P_{\text{C6}}(z, A)$ in Section 7), we verify them one at a time, in the following sense. Taking advantage of nondeterminism, we require A to additionally store the output values of all $\mathsf{size}(K)$ gates in K. In order to do so, we consider sufficiently many additional new layers in H, and in these layers we take $\mathsf{size}(K) + 1$ new columns; call these columns $B_{4,1}, \ldots, B_{4,\mathsf{size}(K)+1}$ and let $B_{4,j} = \hat{V}_0 + \theta^\star_{4,j}$ for some constant $\theta^\star_{4,j} \in \mathbb{F}_{2^f}$. (See Figure 4(a) on page 21.) We use these columns to store the gate values in the following sense: for each $v \in [2^t]$, letting $\gamma_v \in \hat{V}_0$ be the v-th element in $\hat{V}_0$ (see Remark 5.4), we define $A(\gamma_v + \theta^\star_{4,j})$ to be the output of the j-th gate in the evaluation of $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v)))$, for each $j \in \{1, 2, \ldots, \mathsf{size}(K)\}$.

We use selector polynomials to verify, one gate at a time, that the gate values stored in A correspond to a correct evaluation of K; verifying any gate only involves a constant-degree check. Specifically, we define the following polynomial:

$$P'_{\text{C6}}(z, A) \stackrel{\text{def}}{=} \sum_{j=1}^{\mathsf{size}(K)} Y_{H,B_{4,j}}(z) \cdot R^K_j(z, A) + \overbrace{Y_{H,B_{4,\mathsf{size}(K)+1}} \cdot A\big(z + \theta^\star_{4,\mathsf{size}(K)+1} + \theta^\star_{4,\mathsf{size}(K)}\big)}^{\text{ensure the output gate of } K \text{ equals } 0} \qquad \text{(C6)}$$

where $R^K_j(z, A)$ is the check that verifies if the j-th gate of K is correctly computed. For instance, if the j-th gate of K is an AND gate and the two inputs of the j-th gate are the x-th and y-th gates, then $R^K_j(z, A) \stackrel{\text{def}}{=} Q_{\text{AND}}(A(z), A(z + \theta^\star_{4,j} + \theta^\star_{4,x}), A(z + \theta^\star_{4,j} + \theta^\star_{4,y}))$ for $Q_{\text{AND}}(c, a, b) \stackrel{\text{def}}{=} c + ab$, which is 0 if and only if c equals to ab.

When one or both inputs of a gate come from input bits to $K\big(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))\big)$, we can check it similarly: if an input comes from $\chi(v), \chi(\pi_1(v))$ or $\chi(\pi_2(v))$, we can use the appropriate shift to retrieve it since it is already stored $H_{1,\star}$, $H_{2,\star}$ or $H_{3,\star}$, or if an input comes from v, we can use the projection polynomial $\Pi_i(z)$ to retrieve the i-th bit of v.

We can now define:

$$P(z, A) \stackrel{\text{def}}{=} P_{\text{C1+C2}}(z, A) + P_{\text{C3+C4+C5}}(z, A) + P'_{\text{C6}}(z, A) \ .$$

At high level, $P(z, A)$ consists of a summation of many terms, and each term consists of a selector polynomial multiplied by a simple constant-degree check. Since we have carefully chosen our selector polynomials to be "turned on" at disjoint locations, for any $\gamma \in H$, at most one selector polynomial can equal to 1, and $P(\gamma, A)$ verifies its corresponding simple check. As γ ranges over H, all the constraints that we need to verify are eventually checked one by one. (See a summary in Figure 4(b) on page 21.)

We now explain how $P(z, A)$ can be thought of as $P\big(z, A(\vec{N}(z))\big)$ for an appropriate sequence $\vec{N}$ of neighbor polynomials, each of degree 1. While at first sight $P(z, A)$ may seem to depend in complex ways

on the values of A, by carefully inspecting $P(z, A)$ one can verify that it only depends on (i) the value of z, (ii) $A(z+\delta_i)$ for $i = 1, \ldots, \Delta$ with $\Delta = O(c + \mathsf{size}(K))$ and $\delta_i \in \mathbb{F}_{2^f}$, and (iii) $A(\Gamma_k(z+\sigma_i))$ for $k \in \{1, \ldots, 8\}$, $i = 1, \ldots, 2c'$, and $\sigma_i \in \mathbb{F}_{2^f}$. So let $\vec{N}(z)$ be the sequence containing these $O(c + \mathsf{size}(K))$ affine functions:

$$\vec{N}(z) \stackrel{\text{def}}{=} \big(z, z+\delta_1, \ldots, z+\delta_\Delta, \Gamma_1(z+\sigma_1), \ldots, \Gamma_8(z+\sigma_{2c'})\big) \ .$$

In light of the above, it is easy to see that $P(z, A)$ can be written as $P\big(z, A(\vec{N}(z))\big)$.

We are now left to bound the size of H and then, based on that, bound the degree of $P\big(z, A(\vec{N}(z))\big)$. Recall that $H = \cup_{j=0}^{2^s-1} H_j$ for some value s that we are now going to determine. Because of the additional information of the gate values that we need to store in $A(\gamma)$, as well as the additional checks that we need to verify in constraint (C6), we need a slightly larger choice of H than we did in Section 7, namely (cf. Figure 4):

$$|H| = |H_0| \cdot 2^s \geq \left(2^t \cdot 2^\ell\right) \cdot \left((2\lceil \frac{c}{f} \rceil + 2) + 3\lceil \frac{c}{2^\ell} \rceil + \left\lceil \frac{\mathsf{size}(K)+1}{2^\ell} \right\rceil\right) \ .$$

Thus, it suffices to choose $|H|$ to be a power of 2 that is smaller than twice the right-hand side. Therefore (recalling that $2^\ell < 8t$ by Definition 5.1 and $f \geq t$ by our eventual choice), we can choose

$$\begin{aligned} |H| &\leq 2 \cdot \left(2^t \cdot 2^\ell\right) \cdot \left((2\lceil \frac{c}{f} \rceil + 2) + 3\lceil \frac{c}{2^\ell} \rceil + \left\lceil \frac{\mathsf{size}(K)+1}{2^\ell} \right\rceil\right) \\ &\leq 2 \cdot \left(2^t \cdot 2^\ell\right) \cdot \left(\frac{2c}{f} + \frac{3c}{2^\ell} + \frac{\mathsf{size}(K)}{2^\ell} + 8\right) \leq 2^t \cdot (38c + 128t + 2\mathsf{size}(K)) \ . \end{aligned}$$

Next, one can verify that the degree of $P\big(z, A(\vec{N}(z))\big)$ is upper bounded by $3|H| + 2|H_0|$. (Indeed, the selector polynomials are all of degree $|H|$, and the highest-degree checks are those in constraint (C2), which have degree $2|H| + 2|H_0|$.) Thus, it suffices to choose $\mathbb{F}_{2^f}$ such that

$$4(3|H|+2|H_0|) \leq |\mathbb{F}_{2^f}| < 8(3|H|+2|H_0|) = 2^t(912c + 3200t + 48\mathsf{size}(K)) = 2^t \cdot O(c + t + \mathsf{size}(K)) \ .$$

Once again, by choosing the construction of P and $\vec{N}$ in this section or that in Section 7, we can in fact ensure a reduction with a field of size

$$|\mathbb{F}_{2^f}| = O(2^t) \cdot \min\Big\{c + t + \mathsf{size}(K)\,,\ (c+t) \cdot \mathsf{deg}(K)\Big\} \ .$$

This concludes the proof of Lemma 1. □

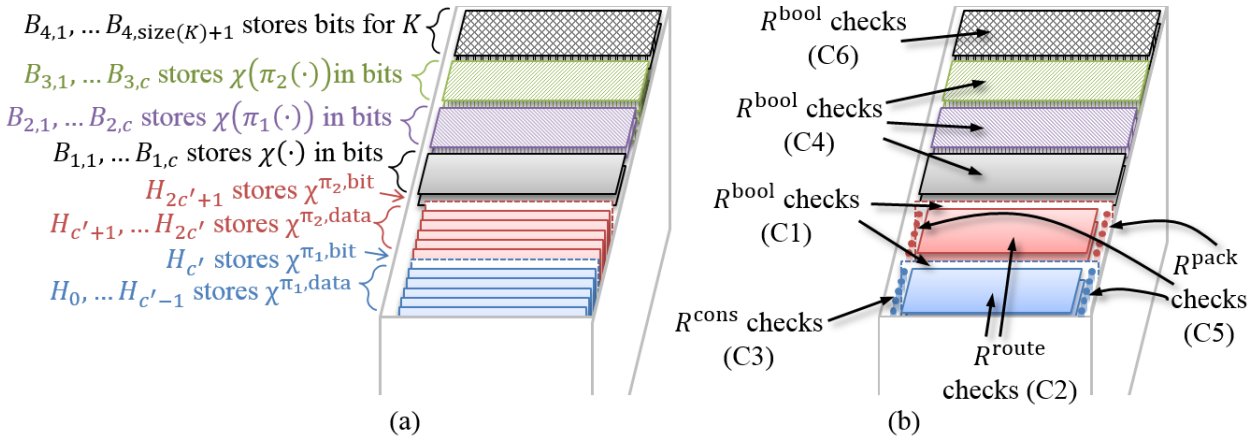

Figure 4: (a) The information encoded in the assignment polynomial A.
(b) The "layout" of checks of the constraint polynomial P from Step 3 in Section 8; all together, these enforce the six constraints of Lemma 6.1.

APPENDIX

A An Open Problem

Can the proof length in Theorem 1 be improved? Say, to $O(T \log T)$? We argue that further improvements in arithmetization proof length are likely to require significantly new ideas and, conversely, ruling out improvements implies circuit lower bounds. We propose an open problem in this direction.

Concretely, consider an sICSP instance (T, c, K), and suppose that $c = \Theta(\log T)$.[9] A candidate witness $\chi\colon [T] \to \{0,1\}^c$ for (T, c, K) requires $Tc = \Theta(T \log T)$ bits to represent. Our proof of Lemma 1 reduces (T, c, K) to an sACSP instance $(\mathbb{F}, H, \vec{N}, P)$ where H has size $O(T \cdot (\log T + c))$. Since a candidate witness A for $(\mathbb{F}, H, \vec{N}, P)$ has degree $\leq |H|$, one needs $|H| \log \mathbb{F} = \Theta(T(\log T)^2)$ bits to represent A. Thus, Lemma 1 incurs a $\Theta(\log T)$ overhead in the number of bits stored in a witness. But is the overhead inherent?

While the $\Theta(\log T)$ overhead can be traced to more than one technical reason, perhaps the most fundamental one is the use of routing techniques for reducing the sICSP instance to a constraint-satisfaction problem (CSP) with "regular structure". In fact, the use of routing (and sorting) is a ubiquitous technique in efficient simulation results for various machine models [Ofm65, Sch78, PF79, SH86, BFLS91, GS89, Rob91, PS94, BS08, BGH+05], including all known quasilinear-length arithmetizations.

Routing techniques introduce overhead due to the following reason. Any degree-d *rearrangeable network* supporting node-disjoint routing of any T-element permutation has at least $\Omega(\frac{T \log T}{\log d})$ vertices, via a simple counting argument [Sha50, Pip80].[10] When using such a network to route T packets of c bits each (corresponding to a witness $\chi\colon [T] \to \{0,1\}^c$), we get an "extended" witness χ' that stores $\Omega(c \cdot \frac{T \log T}{\log d}) = \Omega(\frac{T(\log T)^2}{\log d})$ bits. Because, ultimately, query complexity grows at least linearly with d, one usually requires $d = O(1)$ or $d = \mathrm{poly}(\log T)$, so that routing introduces an $\Omega(\frac{\log T}{\log \log T})$ multiplicative overhead.[11]

Thus, improvements in proof length seem to depend on either (i) avoiding the use of routing altogether; or (ii) finding a generalized notion of routing that suffices for the task but for which the simple counting argument does not apply. A solution to (i) is likely to have far-reaching implications to efficient simulation results of machine models. In contrast, we see (ii) as a slightly more tractable route. To guide future research along this route, we suggest the notion of a *permutation CSP* and an open problem about this notion.

Definition A.1. *Consider a tuple $(T, V, \vec{N}, S_1, S_2, c, R)$, where*

- *$T \in \mathbb{Z}_+$;*
- *V is a vertex set;*
- *$\vec{N} = (N_i\colon V \to V)_{i=1}^d$ is a sequence of neighbor functions inducing the edge set $\big\{(v, N_i(v))\big\}_{v \in V, i \in [d]}$;*
- *S_1 is a* source *vertex set of cardinality T;*
- *S_2 is a* target *vertex set of cardinality T;*
- *$c \in \mathbb{Z}_+$ is the number of bits in a color; and*
- *$R\colon V \times \{0,1\}^{d \cdot c} \to \{0,1\}$ is a boolean constraint circuit.*

We say that $(T, V, \vec{N}, S_1, S_2, c, R)$ is a **(T-packet d-regular c-bit) permutation CSP** *if, for every $\chi\colon V \to \{0,1\}^c$,*

$$\chi|_{S_1} \text{ is a permutation of } \chi|_{S_2} \text{ if and only if } R\big(v, \chi(\vec{N}(v))\big) = 0 \text{ for every } v \in V.$$

Question (linear-size permutation CSPs)**.** *For any number of packets $T \in \mathbb{Z}_+$ and packet size $c \in \mathbb{Z}_+$, is there a T-packet d-regular c-bit permutation CSP where $d = \mathrm{poly}(\log T)$ and the vertex set has size $O(T)$?*

[9] This is the case for sICSP instances obtained from Turing machines (see Proposition 3.4). More generally, $c = \Omega(\log T)$ is crucial for sICSP to be NEXP-complete. For instance, the restriction of sICSP to instances where $c = O(1)$ lies in PSPACE, via an integer programming argument; and the restriction to $c = o(\log T)$ is not NEXP-complete.

[10] More precisely, the cited works provide a lower bound on the number of edges, but the analysis can be extended to show a lower bound on the number of vertices.

[11] In contrast, by aiming at query complexity T^ε, Ben-Sasson et al. [BKK+13] only incur a constant overhead for routing.

One can verify that *proving* that the answer to the above question is "no" implies nondeterministic superlinear circuit lower bounds. Roughly, one can construct a nondeterministic function f such that if f has a linear-size circuit then there exists a linear-size permutation CSP. In particular, the counting argument for routing networks does not extend to ruling out the existence of linear-size permutation CSPs.

On the other hand, we do not see provable barriers to the construction of linear-size permutation CSPs. Thus, understanding whether a linear-size permutation CSP can be constructed remains a very interesting open problem. Perhaps investigating if *network coding* [ACLY06] can be leveraged is a natural first step.

B Proof of Proposition 3.4

We sketch the proof of Proposition 3.4, which states that $\mathcal{L}_{\mathsf{TM}}, \mathcal{L}_{\mathsf{CSAT}}, \mathcal{L}_{\mathsf{RAM}}$ can all be efficiently reduced to sICSP. To prove Proposition 3.4, it suffices to:

1. Show an efficient reduction from $\mathcal{L}_{\mathsf{RAM}}$ to sICSP.
2. Show that both $\mathcal{L}_{\mathsf{TM}}$ and $\mathcal{L}_{\mathsf{CSAT}}$ have efficient reductions to $\mathcal{L}_{\mathsf{RAM}}$.

The first step is implied by techniques of Ben-Sasson et al. [BCGT13a]; the second step intuitively follows from the fact that a random-access machine model can simulate Turing machine and circuit computations with only a constant-factor overhead.[12]

For completeness, in this appendix we choose to sketch two proofs:

- In Section B.1, we show a direct reduction from $\mathcal{L}_{\mathsf{CSAT}}$ to sICSP (i.e., without going through $\mathcal{L}_{\mathsf{RAM}}$). This proof relies on ideas of Polishchuk and Spielman [PS94]. We think this proof gives a strong intuition for how sICSP can express NEXP computations, and hence we present even if the next proof suffices.
- In Section B.2, we sketch a reduction from $\mathcal{L}_{\mathsf{RAM}}$ to sICSP. This proof is somewhat more complicated than the previous one. The proof is based on techniques from Ben-Sasson et al. [BCGT13a].

B.1 Reduction from $\mathcal{L}_{\mathsf{CSAT}}$ to sICSP

We show a linear-time reduction f from $\mathcal{L}_{\mathsf{CSAT}}$ to sICSP such that, for any input (M, T) where M is a circuit descriptor and T is the number of gates in the circuit described by M, $f(M, T)$ outputs a tuple $(2T, c, K)$ such that $c = O(\log T)$, $\mathsf{size}(K) = O(\mathsf{size}(M) + \log T)$, and $\mathsf{deg}(K) = O(\mathsf{deg}(M) + \log T)$.

First recall that a circuit descriptor M is (itself) a circuit that takes as input a gate $g \in [T]$, and outputs the type of the gate g, as well as the gates connected to the (at most 2) inputs and (at most 2) outputs of g.

The *trace of execution* of an instance (M, T) is the sequence $S = (S_1, \ldots, S_{2T})$ where, for each $g \in [T]$:

- $S_{2g-1} = ((g, 1), \mathsf{in}_1, \mathsf{out})$ where in_1 is the *first* input bit of gate g and out is the output bit of gate g; and
- $S_{2g} = ((g, 2), \mathsf{in}_2, \mathsf{out})$ where in_2 is the *second* input bit of gate g (and out as above).

For notational convenience, we identify $(g, 1)$ with the integer $2g - 1$, and $(g, 2)$ with the integer $2g$.

To verify that S is a valid (and accepting!) trace, it suffices to verify that each gate is correctly computed, each wire is connected as prescribed by the circuit descriptor M, and the final output is 1. With this in mind, we define two relations, each between two "states" $((g_1, b_1), \mathsf{in}_1, \mathsf{out}_1)$ and $((g_2, b_2), \mathsf{in}_2, \mathsf{out}_2)$.

- *Code consistency.* We write $((g_1, b_1), \mathsf{in}_1, \mathsf{out}_1) \xrightarrow{\mathsf{code}} ((g_2, b_2), \mathsf{in}_2, \mathsf{out}_2)$ if and only if
  - $(g_2, b_2) = (g_1, b_1) + 1$ (with the understanding that $2T + 1$ is 1), **AND**
  - $g_1 = g_2$ implies $\mathsf{out}_1 = \mathsf{out}_2 = \mathsf{GATE}(\mathsf{in}_1, \mathsf{in}_2)$, where the boolean gate GATE is given by $M(g)$. (Also: if g is an output gate, then in_1 must be 1; and if g is an input gate, then out can be arbitrary.)

[12] More precisely, one can verify the following:

- a $\mathcal{L}_{\mathsf{TM}}$ instance (M_{TM}, T) can be reduced to a corresponding $\mathcal{L}_{\mathsf{RAM}}$ instance (M_{RAM}, T') where $T' = O(T)$, $\mathsf{size}(M_{\mathsf{RAM}}) = O(\mathsf{size}(M_{\mathsf{TM}}) + \log T)$, and $\mathsf{deg}(M_{\mathsf{RAM}}) = O(\mathsf{deg}(M_{\mathsf{TM}}) + \log T)$; and
- a $\mathcal{L}_{\mathsf{CSAT}}$ instance (M_{CSAT}, T) can be reduced to a corresponding $\mathcal{L}_{\mathsf{RAM}}$ instance (M_{RAM}, T') where $T' = O(T)$, $\mathsf{size}(M_{\mathsf{RAM}}) = O(\mathsf{size}(M_{\mathsf{CSAT}}) + \log T)$, and $\mathsf{deg}(M_{\mathsf{RAM}}) = O(\mathsf{deg}(M_{\mathsf{CSAT}}) + \log T)$.

- *Topology consistency.* We write $((g_1, b_1), \mathsf{in}_1, \mathsf{out}_1) \xrightarrow{\mathsf{topo}} ((g_2, b_2), \mathsf{in}_2, \mathsf{out}_2)$ if and only if
  - $M(g_1)$ specifies that the b_1-th input of g_1 comes from the b_2-th output of g_2, and $\mathsf{in}_1 = \mathsf{out}_2$, **OR**
  - $M(g_1)$ specifies that the b_1-th input of g_1 is not used.

It is immediate to see that a trace $S = (S_1, \ldots, S_{2T})$ is valid and accepting if

1. $S_v \xrightarrow{\mathsf{code}} S_{v+1}$ for every $v \in [2T]$ and
2. there exists a permutation $\sigma \colon [2T] \to [2T]$ such that $S_v \xrightarrow{\mathsf{topo}} S_{\sigma(v)}$ for every $v \in [2T]$.

Intuitively, σ is defined such that for each $v = (g, b) \in [2T]$, $\sigma(v) = (g', b')$ where the b-th input of g is connected to the b'-th output of g', or an arbitrary number if this input is not used.

Thus, given (M, T), we define f to output the tuple $(2T, c, K)$ where c, K are defined as follows.

- $c = \lceil \log T \rceil + 3 = O(\log T)$.
- $K \colon [2T] \times \{0,1\}^{3c} \to \{0,1\}$ is the boolean circuit that verifies both code and topology consistency. Specifically, $K\big(v, S, S', S''\big) = 0$ if and only if $S \xrightarrow{\mathsf{code}} S'$ and $S \xrightarrow{\mathsf{topo}} S''$. The two relations $S \xrightarrow{\mathsf{code}} S'$ and $S \xrightarrow{\mathsf{topo}} S''$ can be verified by a circuit that contains the circuit descriptor M, plus a simple circuit of size and degree $O(\log T)$ in charge mostly of additions and comparisons. Thus, the size and (total) degree of K are $O(\mathsf{size}(M) + \log T)$ and $O(\mathsf{deg}(M) + \log T)$ respectively.

The tuple $(2T, c, K)$ can be constructed in $O(\mathsf{size}(M) + \log T)$ time so the reduction f runs in linear time.

To argue completeness of the reduction, suppose that it is indeed the case that the circuit described by M accepts some input w, and let $S = (S_1, \ldots, S_{2T})$ be a valid and accepting trace. Construct the witness (χ, π_1, π_2) as follows:

- $\chi(v) \stackrel{\text{def}}{=} S_v$ for every $v \in [2T]$;
- $\pi_1(v) \stackrel{\text{def}}{=} v + 1$ for every $v \in [2T]$ (with the understanding that $2T + 1$ is 1);
- $\pi_2(v) \stackrel{\text{def}}{=} \sigma(v)$ for every $v \in [2T]$, where σ is defined above.

By construction, $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v)) = 0$ for every $v \in [2T]$, so $(2T, c, K)$ is in sICSP.

To argue soundness of the reduction, if there is (χ, π_1, π_2) such that $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))) = 0$ for every $v \in [2T]$, one can argue that $S \stackrel{\text{def}}{=} \chi$ is a valid and accepting trace for M. □

B.2 Reduction from $\mathcal{L}_{\mathsf{RAM}}$ to sICSP

We sketch a linear-time reduction f from $\mathcal{L}_{\mathsf{RAM}}$ to sICSP such that, for any input (M, T) where M is a nondeterministic random-access machine and T is a time step bound, $f(M, T)$ outputs a tuple (T, c, K) such that $c = O(\log T)$, $\mathsf{size}(K) = O(\mathsf{size}(M) + \log T)$, and $\mathsf{deg}(K) = O(\mathsf{deg}(M) + \log T)$.

The random-access machine model. Random-access machines [CR72, AV77] can be defined in many reasonable ways, depending on the "choice of architecture". While the techniques discussed in this section apply to many such choices, for the sake of concreteness, we work with a simple *load-store architecture*.

Concretely, a machine M maintains $k = O(1)$ local *registers*, and has random access to an array of memory cells. The length of a register and of a memory cell, denoted w, is called the *word size*. The machine can directly address 2^w memory cells; we should think of w as $O(\log T)$ bits. Hardcoded in M is a *program*, which is a list of basic instructions. One of the k registers, say the first one, is the *program counter*, and it is a pointer to the next instruction (in M's program) to be executed. Instructions are of two types:

- An *arithmetic/branch instruction*. (For example: "the contents of the second and fourth registers are to be multiplied, and the result must be stored in the fifth register". Another example: "the program counter is to be set to the value of the third register if the second register equals 0".)
- A *memory instruction*. Such an instruction is a load instruction (e.g., requesting the value from memory stored at the address that is equal to the contents of the second register) or a store instruction (e.g., putting the contents of the third register at the address that is equal to the contents of the fourth register).

At every time step, the instruction pointed to by the program counter is executed and (in case there was no branch instruction) the program counter is incremented by 1.

Nondeterminism. We actually need M to be a *non-deterministic* machine. For instance, M may have a special register $\tilde{r}$ that at every time step receives nondeterministic bits, and there is a special "read $\tilde{r}$" instruction that copies the contents of register $\tilde{r}$, say, to the second register. We shall ignore this in our high-level proof sketch; it is not technically difficult to account for nondeterminism.

Transition function checker. The *transition function checker* of M is the function δ_M that, on input two sequences of k registers (i.e., two local states of the machine) S and S', outputs 0 if and only if by executing the instruction pointed to by the program counter in S one obtains S' — up to memory consistency. (Namely, if the instruction to be executed is a load instruction, then δ_M only checks that all registers but the one being written to, remain the same, and that the program counter increases by 1.) In other words, δ_M verifies that arithmetic/branch instructions are executed properly, and that, when a register is not modified, it does not change value from one state to the other.

Execution traces. A T-step *trace of execution* of M is a sequence $S = (S_1, \ldots, S_T)$ where, for every $v \in [T]$, $S_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,k})$ is (allegedly) the state of the k registers in M before executing the v-th instruction. To verify that S is a *valid and accepting trace*, it suffices to verify the following two conditions:

- *Code consistency:* $\delta_M(S_v, S_{v+1}) = 0$ for each $v \in [T-1]$, where δ_M is the transition function checker of M (defined above); S_1 is all 0's (i.e., is "initial"); and S_T is an accepting state.
- *Memory consistency:* for every address a, a load from address a returns the value last stored in address a, or the value 0 if the address a was never accessed.

The code and memory relations. Given two timestamped states (τ, S) and (τ', S'), define two relations $(\tau, S) \xrightarrow{\mathsf{code}} (\tau', S')$ and $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau', S')$ as follows.

- *Code consistency.* We write $(\tau, S) \xrightarrow{\mathsf{code}} (\tau', S')$ if and only if
  1. $\tau' = \tau + 1$ (with the understanding that $T + 1$ is 1),
  2. if $\tau \in [T-1]$ then $\delta_M(S, S') = 0$, and
  3. if $\tau = T$ then S' is accepting and S is all 0's (i.e., S is initial).
- *Memory consistency.* We write $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau', S')$ if and only if
  1. if neither of the program counters in S and S' point to a memory instruction then $(\tau, S) = (\tau', S')$,
  2. if both of the program counters in S and S' point to memory instructions accessing the same memory cell, then $\tau < \tau'$ and the accesses are consistent (i.e., if both are loads then they should load the same value, and if the first is a store and the second a load then the second one should load the stored value), and
  3. if both of the program counters in S and S' point to memory instructions, and they access different memory cells a and a', then $a < a'$ and if the second access is a load, the value loaded should be 0.

It is immediate to see that a T-step trace $S = (S_1, \ldots, S_T)$ is code consistent if and only if $(v, S_v) \xrightarrow{\mathsf{code}} (v+1, S_{v+1})$ for every $v \in [T]$. Moreover, one can argue that, if S is code consistent, then it is also memory consistent if and only if there exists a permutation $\sigma\colon [T] \to [T]$ such that $S_v \xrightarrow{\mathsf{mem}} S_{\sigma(v)}$ for every $v \in [T]$.[13]

Intuitively, σ is defined as follows. Let $(S_{g_1}, \ldots, S_{g_m})$ be those states in S whose program counter points to a memory instruction, sorted by the address in memory accessed by the instruction (breaking ties with timestamps). Then, if $v = g_i$ for some i, we let σ map v to g_{i+1}; otherwise, we let σ map v to itself.

The reduction. Thus, given (M, T), we define f to output the tuple (T, c, K) where c, K are as follows.

- $c = \lceil \log T \rceil + kw = \lceil \log T \rceil + O(1)O(\log T) = O(\log T)$.
- $K\colon [T] \times \{0,1\}^{3c} \to \{0,1\}$ is the boolean circuit that verifies both code and memory consistency. Specifically, $K\big(v, (\tau, S), (\tau', S'), (\tau'', S'')\big) = 0$ if and only if $v = \tau$, $(\tau, S) \xrightarrow{\mathsf{code}} (\tau', S')$ and $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau'', S'')$. Note that:
  - $v = \tau$ can be verified by an equality-checking circuit of both size and degree $O(\log T)$;
  - $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau'', S'')$ can be verified by a circuit where the most expensive computations are on $O(\log T)$-bit comparisons so is also of size and degree $O(\log T)$; and
  - $(\tau, S) \xrightarrow{\mathsf{code}} (\tau', S')$ can be verified by the circuit δ_M, of size and degree $\mathsf{size}(M)$ and $\mathsf{deg}(M)$ respectively, plus a simple sub-circuit of size and degree $O(\log T)$ (e.g., verifying if $\tau' = \tau + 1$).

  The size and (total) degree of K are the summations of the three pieces above, which amounts to $O(\mathsf{size}(M) + \log T)$ and $O(\mathsf{deg}(M) + \log T)$ respectively.

The tuple (T, c, K) can be constructed in $O(\mathsf{size}(M) + \log T)$ time so the reduction f runs in linear time.

To argue completeness of the reduction, suppose that it is indeed the case that M accepts within T steps, and let $S = (S_1, \ldots, S_T)$ be a valid and accepting T-step trace for M. Construct the witness (χ, π_1, π_2) as follows:

- $\chi(v) \stackrel{\text{def}}{=} (v, S_v)$ for every $v \in [T]$;
- $\pi_1(v) \stackrel{\text{def}}{=} v + 1$ for every $v \in [T]$ (with the understanding that $T + 1$ is 1);
- $\pi_2(v) \stackrel{\text{def}}{=} \sigma(v)$ for every $v \in [T]$, where σ is defined above.

By construction, $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v)) = 0$ for every $v \in [T]$, so (T, c, K) is in sICSP.

To argue soundness of the reduction, if there is (χ, π_1, π_2) for which $K(v, \chi(v), \chi(\pi_1(v)), \chi(\pi_2(v))) = 0$ for every $v \in [T]$, one can argue that $S \stackrel{\text{def}}{=} \chi$ is a valid and accepting T-step trace for M.

From sketch to proof. In the above sketchy description, we have omitted many technical details of the construction and arguments of correctness. (For instance, in order for the relation $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau', S')$ to work properly, one needs to modify the machine M slightly so that accesses of the machine M to memory satisfy some simple properties.) We refer the interested reader to [BCGT13a] for a detailed discussion. □

[13] A careful reader will notice that additional details are required for this second, more subtle, step. For instance, $(\tau, S) \xrightarrow{\mathsf{mem}} (\tau', S')$ needs to also check that if $\tau' = 0$ then S' is a store to address 0. For further details, see [BCGT13a].

C Proof of Proposition 3.5

Let $x = (\mathbb{F}, H, \vec{N}, P)$ be an instance, potentially in the language sACSP. We construct a probabilistic oracle machine V_x and low-degree polynomial codes $\vec{C}_x = (C_{x,1}, C_{x,2})$ as follows.

- Let $\mathrm{RS}(\mathbb{F}, S, d)$ be the Reed–Solomon code of degree-d polynomials over $\mathbb{F}$ evaluated on $S \subseteq \mathbb{F}$. We set

$$C_{x,1} \stackrel{\text{def}}{=} \mathrm{RS}(\mathbb{F}, \mathbb{F}, |H|) \quad \text{and} \quad C_{x,2} \stackrel{\text{def}}{=} \mathrm{RS}\left(\mathbb{F}, \mathbb{F}, \frac{|\mathbb{F}|}{4} - |H|\right) \ .$$

- We set V_x to be the probabilistic oracle machine such that, when given oracle access to $\pi_1 \in C_{x,1}$ and $\pi_2 \in C_{x,2}$, proceeds as follows:
  1. picks $\gamma \in \mathbb{F}$ uniformly at random;
  2. verifies if $P(\gamma, \pi_1(N_1(\gamma)), \ldots, \pi_d(N_d(\gamma))) = \pi_2(\gamma) Z_H(\gamma)$, where $Z_H(z) = \prod_{\alpha \in H}(z - \alpha)$ is the polynomial vanishing exactly on H.

To see the completeness property, suppose that x is in sACSP and let A be an assignment polynomial for x. We set π_1 to be the evaluation of $A(z)$ on $\mathbb{F}$, and set π_2 to that of $\frac{1}{Z_H(z)} P\big(\gamma, A(N_1(z)), \ldots, A(N_d(z))\big)$ on $\mathbb{F}$. Because the first polynomial has degree at most $|H|$ and the second has degree at most $\frac{|\mathbb{F}|}{4} - |H|$, we know that $\pi_1 \in C_{x,1}$ and $\pi_2 \in C_{x,2}$. It is then easy to see that $V_x^{\pi_1, \pi_2}$ accepts, with probability 1, by construction.

To see the soundness property, suppose that x is not in sACSP. In such a case, for any choice of $(\pi_1, \pi_2) \in \vec{C}_x$, the polynomial identity $P(z, \pi_1(N_1(z)), \ldots, \pi_d(N_d(z))) = \pi_2(z) Z_H(z)$ cannot hold. Since the degrees in z on both sides do not exceed $\frac{|\mathbb{F}|}{4}$, by the Schwartz–Zippel lemma, $V_x^{\pi_1, \pi_2}$ rejects with at least $\frac{3}{4}$ probability over the random choice of $\gamma \in \mathbb{F}$.

Finally, let us comment on the efficiency parameters of the code PCP $(V, \vec{C})$. The proof length is $|\pi_1| + |\pi_2| = 2 \cdot (|\mathbb{F}| \log |\mathbb{F}|) = O(|\mathbb{F}| \log |\mathbb{F}|)$. (The logarithmic factor comes from the fact that $\mathrm{RS}(\mathbb{F}, S, d)$ has alphabet size $|\mathbb{F}|$.) The query complexity is $d + 1 = O(d)$. In terms of verifier complexity, the only subtlety is how does V_x efficiently evaluate the polynomial Z_H at an element γ. (Note that H is large!) It turns out that, because H is an $\mathbb{F}_2$-linear subspace of $\mathbb{F}$, the polynomial Z_H can be written as $\sum_{i=0}^{\log |H|} c_i z^{2^i}$ for some $c_i \in \mathbb{F}$ and the c_i can be found in $(\log |H|)^{O(1)}$ time [BGH+05, BCGT13a]; the c_i can be computed as part of the reduction and hard-coded in V_x. So V_x only has to perform a number of $\mathbb{F}$-arithmetic operations that is $O(d + \log |\mathbb{F}| + \mathsf{size}(P) + \sum_{i=1}^{d} \mathsf{size}(N_i))$. Thus, overall, the verifier complexity is $\mathrm{poly}(d + \log |\mathbb{F}| + \mathsf{size}(P) + \sum_{i=1}^{d} \mathsf{size}(N_i)) = \mathrm{poly}(|x|)$. □

References

[ACLY06] R. Ahlswede, Ning Cai, S. Y.R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2006.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.

[AS97] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 485–495, 1997.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.

[AV77] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *Proceedings on 9th Annual ACM Symposium on Theory of Computing*, STOC '77, pages 30–41, 1977.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 90–108, 2013.

[BCGT13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*, ITCS '13, pages 401–414, 2013.

[BCGT13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 585–594, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 315–333, 2013.

[Ben65] Václav E. Beneš. *Mathematical theory of connecting networks and telephone traffic*. New York, Academic Press, 1965.

[BF91] László Babai and Lance Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[BFL90] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 16–25, 1990.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.

[BGH+04] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '04, pages 1–10, 2004.

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 120–134, 2005.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006. Preliminary versions of this paper have appeared in STOC and in ECCC.

[BKK+13] Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate PCPs for circuit-sat with sublinear query complexity. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '13, pages ???–???, 2013.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

[BSVW03] Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, STOC '03, pages 612–621, 2003.

[CR72] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, 1972.

[DH09] Irit Dinur and Prahladh Harsha. Composition of low-error 2-query PCPs using decodable PCPs. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 472–481, 2009.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[DR04] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in FOCS '91.

[FK97] Uriel Feige and Joe Kilian. Making games short. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 506–516, 1997.

[FL93] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113(1):55–73, 1993.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 626–645, 2013.

[GLR+91] Peter Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 33–42, 1991.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '10, pages 321–340, 2010.

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, pages 108–118, 1989.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM*, 53:558–655, July 2006. Preliminary version in STOC '02.

[HS00] Prahladh Harsha and Madhu Sudan. Small PCPs with low query complexity. *Computational Complexity*, 9(3–4):157–201, Dec 2000. Preliminary version in STACS '91.

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

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Nisan Noam. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 169–189, 2012.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, Cambridge, UK, second edition edition, 1997.

[Mei09] Or Meir. Combinatorial PCPs with efficient verifiers. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 463–471, 2009.

[Mei12] Or Meir. Combinatorial PCPs with short proofs. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, CCC '12, 2012.

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with o(1) queries. *Annals of Mathematics and Artificial Intelligence*, 56:313–338, 2009.

[MR08] Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *Journal of the ACM*, 57:1–29, June 2008. Preliminary version appeared in FOCS '08.

[Ofm65] Yuri P. Ofman. A universal automaton. *Transactions of the Moscow Mathematical Society*, 14:200–215, 1965.

[OTW71] D. C. Opferman and N. T. Tsao-Wu. On a class of rearrangeable switching networks - part i: Control algorithm. *Bell System Technical Journal*, 50(5):1579–1600, 1971.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26:361–381, 1979.

[PGHR13] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, Oakland '13, pages 238–252, 2013.

[Pip80] Nicholas Pippenger. A new lower bound for the number of switches in rearrangeable networks. *SIAM Journal on Algebraic Discrete Methods*, 1(2):164–167, 1980.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, pages 194–203, 1994.

[Rob91] J. M. Robson. An O(T log T) reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82(1):141–149, May 1991.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials withapplications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, 1997.

[SBV+13] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th EuoroSys Conference*, EuroSys '13, pages 71–84, 2013.

[Sch78] Claus-Peter Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25:136–145, January 1978.

[SH86] Richard E. Stearns and Harry B. III Hunt. On the complexity of the satisfiability problem and the structure of NP. Technical Report 82-21, State University of New York at Albany, Computer Science Department, 1986.

[Sha50] Claude E Shannon. Memory requirements in a telephone exchange. *Bell System Technical Journal*, 29(343–349):540, 1950.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[She92] Alexander Shen. IP = PSPACE: simplified proof. *Journal of the ACM*, 39(4):878–880, 1992.

[SW13] Rahul Santhanam and Ryan Williams. On medium-uniformity and circuit lower bounds. In *Proceedings of the 28th Annual IEEE Conference on Computational Complexity*, CCC '13, pages 15–23, 2013.

[Wak68] Abraham Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, 1968.