Customizable constraint systems for succinct arguments

Srinath Setty* Justin Thaler† Riad Wahby‡

Abstract

This paper introduces *customizable constraint system (CCS)*, a generalization of R1CS that can simultaneously capture R1CS, Plonkish, and AIR without overheads. Unlike existing descriptions of Plonkish and AIR, CCS is not tied to any particular proof system. Furthermore, we observe that the linear-time polynomial IOP for R1CS in Spartan (CRYPTO 20) extends easily to CCS, and when combined with a polynomial commitment scheme, it yields a family of SNARKs for CCS, which we refer to as *SuperSpartan*. SuperSpartan supports high-degree constraints without its prover incurring cryptographic costs that scale with the degree of constraints (only field operations scale with the constraint degree). Moreover, as in Spartan, it does not employ superlinear-time and hard-to-distribute operations such as FFTs. Similar properties were achieved for Plonkish by HyperPlonk (EUROCRYPT 23) via a different route. However, it is unclear how to prove CCS instances (or even R1CS instances) with HyperPlonk (or Plonk itself), without overheads. Furthermore, unlike HyperPlonk, SuperSpartan can prove uniform instances of CCS (including AIR) without requiring a linear-time preprocessing for the verifier.

SuperSpartan for AIR is the first SNARK for AIR with a linear-time prover, transparent and sublinear-time pre-processing, polylogarithmic proof size, and plausible post-quantum security. In particular, SuperSpartan for AIR provides a faster prover than existing transparent SNARKs for AIR (which are sometimes referred to as STARKs).

1 Introduction

A succinct non-interactive argument of knowledge (SNARK) [Kil92, Mic94, GW11, BCCT12] allows an untrusted prover $\mathcal{P}$ to prove the knowledge of a witness w satisfying some property. For example, w could be a pre-image of a designated value y of a cryptographic hash function h, i.e., a w such that $h(w) = y$. A trivial proof is for $\mathcal{P}$ to send w to the verifier $\mathcal{V}$, who directly checks that w satisfies the claimed property. A SNARK achieves the same, but with better verification costs (and proof sizes). Succinct means that verifying a proof is exponentially faster than checking the witness directly (this also implies that proofs are exponentially smaller than the size of the statement proven).

When SNARKs are used in practice, they are typically applied to prove the correct execution of programs (e.g., verifiable computation), possibly in zero-knowledge. Furthermore, SNARKs are built using protocols that perform certain probabilistic checks, so to apply SNARKs to program executions, one must express the execution of a program in a specific form that is amenable to probabilistic checking (e.g., as arithmetic circuits). Many deployed systems include circuits that are written and optimized by hand [bel]. There is also a long line of academic research [SVP+12, SBV+13, PGHR13, BFR+13, WSR+15, SAGL18, LNS20, OWB20] and industrial work (e.g., zokrates, circom, noir, zinc, cairo) to build automated tools that transform programs expressed in a high level languages into circuits.

There are several generalizations of arithmetic circuits, which we refer to as *intermediate representations* (IRs), that are in use today: rank-one constraint system (R1CS), algebraic intermediate representation (AIR), and Plonkish.[1] We provide more precise definitions of these generalizations of arithmetic circuits in Section 2. Of these three generalizations of arithmetic circuits, many practitioners apparently prefer Plonkish. Unlike AIR,

*Microsoft Research
†a16z crypto research and Georgetown University
‡Carnegie Mellon University

[1] Plonkish is sometimes also referred to as constraint systems with "custom gates" or as RAPs (short for randomized AIR with pre-processing) or as TurboPLONK/UltraPLONK.

it does not require circuits to be uniform, and unlike R1CS, it supports constraints with degree larger than two. As a result of this expressiveness, Plonkish can represent certain program executions more succinctly than R1CS or AIR. On the other hand, the prover may incur a higher per-gate (or per-constraint) cost to prove Plonkish. For many applications of interest, savings from reduced circuit sizes might outweigh the prover's cost of supporting more expressive gates.

As a result, Plonkish is often considered to be substantially different from, and superior to, R1CS. Furthermore, it is widely believed that, to support Plonkish, one must build on the the Plonk proof system [GWC19], though one may change the underlying building blocks for better efficiency. Indeed, this is the approach taken in recent work called HyperPlonk [CBBZ23]. As elaborated upon later, HyperPlonk closely adheres to the proof system underlying Plonk, but it replaces "gadgets" for "zero checks", "permutation checks", and "product checks" based on univariate polynomials with equivalent gadgets based on multilinear polynomials [Set20, SL20]. The motivation for this replacement is that these gadgets based on multilinear polynomials employ the *sum-check protocol* [LFKN90], which has a unique cost profile for the prover (e.g., it can be implemented with a linear-time prover [Tha13, Set20]).

1.1 Our contributions

CCS. We introduce a new generalization of R1CS that we call *Customizable Constraint Systems* (CCS). We show that CCS simultaneously generalizes Plonkish, AIR, and R1CS, in the sense that there are essentially costless reductions from instances of each of these IRs to equivalent CCS instances. In Appendix B, we define a natural extension of CCS that additionally allows lookup operations against read-only tables/memories.

Prior descriptions of Plonkish are typically closely tied to how the Plonk proof system internally works (e.g., "Plonkish circuits" are typically described in terms of gate checks, copy checks, permutation checks, etc. and the Plonk proof system is a set of gadgets to prove these types of checks). In contrast, the description of CCS is clearly separated from how it actually gets proven. Indeed, similar to R1CS itself, our definition of CCS involves only matrix-vector products, Hadamard (i.e., entry-wise) vector products, and summation. We hope that this clarifies that Plonkish, like AIR, R1CS, and circuits themselves, is an intermediate representation that can be efficiently proven by many different proof systems, not only variants of Plonk.

Similarly, descriptions of AIR satisfiability (e.g., [BSCKL23, Sta21, BBHR19a]) are often tied to the proof systems used to prove satisfiability of the AIR, and thereby explicitly involve cyclic subgroups of a finite field, cosets thereof, univariate polynomials, etc. We give a definition of AIR that involves only matrix-vector products, Hadamard products, and summation, clearly separating the definition of the AIR from the proof system used to prove its satisfiability.

We hope that our definitions of these IRs and the associated transformations to CCS further clarifies the relationship between Plonkish, AIR, and R1CS.

SNARKs for CCS. We describe SNARKs for CCS. Our technical contribution amounts to observing that several polynomial IOPs for CCS are essentially already known from prior polynomial IOPs for R1CS, and when combined with a polynomial commitment scheme, they yield SNARKs for R1CS. In particular, we describe generalizations of existing polynomial IOPs for R1CS, namely Spartan [Set20] and Marlin [CHM$^+$20]. We refer to these generalizations as SuperSpartan and SuperMarlin respectively. Of these, we find SuperSpartan to achieve a particularly compelling cost profile: the prover's cryptographic costs do not scale with the degree of constraints, and as in Spartan, it does not employ superlinear-time and hard-to-parallelize operations such as FFTs. As noted above, a similar cost profile was recently achieved by HyperPlonk [CBBZ23] via a different route.

Remark 1. While we show that known polynomial IOPs for R1CS (Spartan's and Marlin's) extend easily to handle CCS, it is not clear how to extend—without asymptotic overheads—known polynomial IOPs for Plonkish (Plonk's and Hyperplonk's) to handle CCS (or even R1CS!). The primary issue is that Plonk and HyperPlonk are restricted to a specific type of linear constraints (which are sometimes referred to as "copy constraints") that can only enforce equality between a pair of values in the satisfying assignment. In contrast,

both Spartan and Marlin have machinery (via a so-called *sparse polynomial commitment* scheme, see, e.g., [Tha20, Sections 10.3.2 and 16.2] for an exposition) to handle general linear constraints.

Remark 2. Spartan's and SuperSpartan's polynomial IOP are *interactive oracle protocols* [BFL92], a proof model that is a *predecessor* of and simpler than (polynomial) IOPs. In the interactive oracle protocol model of Babai et al. [BFL92], the verifier has access to certain oracles from the prover, in the form of functions or polynomials, that it can query, but otherwise the prover and the verifier engage in an interactive proof (a closely related proof model is interactive PCPs [KR08] where the prover's oracle is a PCP). In particular, in an interactive oracle protocol, no oracles are sent during the interaction.

SNARKs for uniform CCS (including AIR). We also describe how SuperSpartan can prove "uniform" circuits (in particular, all AIR instances) while providing a succinct verifier without requiring any preprocessing. Achieving this relies on a more novel technical contribution: the verifier has to evaluate certain multilinear polynomials that capture the "wiring" of the circuit (or more generally, CCS instance) under consideration. Allowing a preprocessing phase would let the verifier commit to those polynomials in an offline phase and then have the prover prove their evaluations during the online phase. We observe that for instances of CCS arising from AIR, the verifier can accomplish this in logarithmic time *without* any preprocessing. Theorem 2 provides details. This ensures that SuperSpartan's verifier runs in logarithmic time (plus the time necessary to verify a single evaluation proof from the polynomial commitment scheme). We thereby achieve SNARKs for AIR with a novel cost profile. Perhaps most notably, by instantiating SuperSpartan with the Orion's polynomial commitment scheme, we obtain the first SNARK for AIR with a polylogarithmic time verifier and linear-time prover. If SuperSpartan uses the predecessor of Orion [XZS22], called Brakedown [GLS+21], we obtain a *field-agnostic* SNARK with linear-time prover, but proofs that are of size square root in the size of the witness to the AIR instance.

Although HyperPlonk [CBBZ23] provides a SNARK for AIR, it inherently requires a circuit-dependent preprocessing phase of time linear in the circuit size, which is exponentially larger than the verifier time of SuperSpartan for AIR. This preprocessing algorithm is referred to as the *indexer* in HyperPlonk and in earlier works that make use of such preprocessing [CHM+20, COS20].

2 Customizable constraint system (CCS)

R1CS and CCS. R1CS is an NP-complete problem implicit in quadratic arithmetic programs (QAPs) [GGPR13]. An R1CS instance comprises a set of m constraints, with a vector z over $\mathbb{F}$ said to *satisfy* the instance if it satisfies all m constraints. The term "rank-1" means that the constraints should have a specific form. Specifically, each constraint asserts that the product of two specified linear combinations of the entries of z equals a third linear combination of those entries. One typically also requires that the trailing entries of z equal some public input (i.e., an input known to both the prover and verifier) followed by a constant of one.

Typically, an R1CS instance is split into a fixed "structure" that describes constraints and an "instance" consisting only of the public input. Such a decomposition allows instance-independent preprocessing, which in turn allows the verifier runtime to be succinct in the size of the structure. Definition 2.1 provides details.

Below, we use $\mathbb{F}$ to denote a finite field. Furthermore, we use the convention that vector indexes begin with 1.

Definition 2.1 (R1CS)**.** An R1CS structure $\mathcal{S}$ consists of size bounds $m, n, N, \ell \in \mathbb{N}$ where $n > \ell$, and three matrices $A, B, C \in \mathbb{F}^{m \times n}$ with at most $N = \Omega(\max(m, n))$ non-zero entries in total. An R1CS instance consists of public input $x \in \mathbb{F}^{\ell}$. An R1CS witness consists of a vector $w \in \mathbb{F}^{n-\ell-1}$. An R1CS structure-instance tuple $(\mathcal{S}, x)$ is satisfied by an R1CS witness w if

$$(A \cdot z) \circ (B \cdot z) - C \cdot z = \mathbf{0}, \tag{1}$$

where $z = (w, 1, x) \in \mathbb{F}^n$, $\cdot$ is a matrix-vector multiplication operation, $\circ$ is the Hadamard (i.e., entry-wise) product between vectors, and $\mathbf{0}$ is an m-sized vector with entries equal to the the additive identity in $\mathbb{F}$.

We now provide a generalization of R1CS, which we refer to as CCS.

Definition 2.2 (CCS)**.** A CCS structure $\mathcal{S}$ consists of:

- size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$;
- a sequence of matrices $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times n}$ with at most $N = \Omega(\max(m, n))$ non-zero entries in total;
- a sequence of q multisets $[S_0, \ldots, S_{q-1}]$, where an element in each multiset is from the domain

$$\{0, \ldots, t-1\}$$

  and the cardinality of each multiset is at most d.
- a sequence of q constants $[c_0, \ldots, c_{q-1}]$, where each constant is from $\mathbb{F}$.

A CCS instance consists of public input $x \in \mathbb{F}^{\ell}$. A CCS witness consists of a vector $w \in \mathbb{F}^{n-\ell-1}$. A CCS structure-instance tuple $(\mathcal{S}, x)$ is satisfied by a CCS witness w if

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = \mathbf{0}, \tag{2}$$

where

$$z = (w, 1, x) \in \mathbb{F}^n, \tag{3}$$

$M_j \cdot z$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard product between vectors, and $\mathbf{0}$ is an m-sized vector with entries equal to the the additive identity in $\mathbb{F}$.

CCS+. We provide a natural extension of CCS, which we refer to as CCS+, that additionally allows specifying lookup operations (in the constraint system) against read-only tables (Definition B.1).

2.1 Representing R1CS with CCS

Recall that an *NP checker* is an algorithm that takes as input an NP instance and an NP witness, and checks if the witness satisfies the instance. We use the runtime of such a checker to assess the efficiency of our transformations. Note that this also acts as a proxy for size overheads (e.g., of the CCS witness transformed from an R1CS witness).

Lemma 1 (R1CS to CCS transformation)**.** *Given an R1CS structure $\mathcal{S}_{R1CS} = (m, n, N, \ell, A, B, C)$ there exists a CCS structure $\mathcal{S}_{CCS}$ such that for any instance $\mathcal{I}_{R1CS} = x$ and witness $w_{R1CS} = w$, the tuple $(\mathcal{S}_{CCS}, x)$ is satisfied by w if and only if $(\mathcal{S}_{R1CS}, x)$ is satisfied by w. In fact, the transformation from $\mathcal{S}_{R1CS}$ to $\mathcal{S}_{CCS}$ runs in* constant *time. The natural NP checker's time to verify that the tuple $((\mathcal{S}_{CCS}, x), w)$ satisfies Equation* (2) *is identical to the natural NP checker's time to verify that the tuple $((\mathcal{S}_{R1CS}, x), w)$ satisfies Equation* (1) *.*

Proof. Let $\mathcal{S}_{\text{CCS}} = (m, n, N, \ell, t, q, d, [M_0, M_1, M_2], [S_1, S_2], [c_1, c_2])$, where m, n, N, ℓ are from $\mathcal{S}_{\text{R1CS}}$, and $t = 3, q = 2, d = 2$, $M_0 = A, M_1 = B, M_2 = C$, $S_1 = \{0, 1\}, S_2 = \{2\}$, and $c_0 = 1, c_1 = -1$. Here, A, B, C are the constraint matrices from $\mathcal{S}_{\text{R1CS}}$. By inspection, it is easy to see that the tuple $(\mathcal{S}_{\text{CCS}}, x)$ is satisfied by w if and only if $(\mathcal{S}_{\text{R1CS}}, x)$ is satisfied by w.

The asserted bounds on the NP checker's time are immediate from the construction. □

2.2 Representing Plonkish with CCS

We first recall the Plonkish constraint system, and then provide a transformation from Plonkish to CCS.

Definition 2.3 (Plonkish)**.** A Plonkish structure $\mathcal{S}$ consists of:

- size bounds $m, n, \ell, t, q, d, e \in \mathbb{N}$;
- a multivariate polynomial g in t variables, where g is expressed as a sum of q monomials and each monomial has a total degree at most d;

- a vector of constants called *selectors* $s \in \mathbb{F}^e$; and
- a set of m constraints. Each constraint i is specified via a vector T_i of length t, with entries in the range $\{0, \ldots, n+e-1\}$. T_i is interpreted as specifying t entries of a purported satisfying assignment z to feed into g.

A Plonkish instance consists of public input $x \in \mathbb{F}^\ell$. A Plonkish witness consists of a vector $w \in \mathbb{F}^{n-\ell}$. A Plonkish structure-instance tuple $(\mathcal{S}, x)$ is satisfied by a Plonkish witness w if:

$$\text{for all } i \in \{0, \ldots, m-1\},\ g(z[T_i[1]], \ldots, z[T_i[t]]) = 0, \tag{4}$$

where

$$z = (w, x, s) \in \mathbb{F}^{n+e}. \tag{5}$$

Remark 3. Plonkish is often specified with a combination of gate constraints and copy constraints, where a copy constraint enforces that a particular pair of elements in the satisfying assignment hold the same value. Our definition of Plonkish provided above eschews copy constraints by essentially using a "deduplicated" version of the satisfying assignment that is at least $2\times$ shorter.

Remark 4. A Plonkish instance typically consists of *multiple* multivariate polynomials g, say, $g_0, \ldots, g_{k-1}$, rather than just one. However, in practice, by using a single polynomial g in conjunction with $\log k$ selectors s to "turn on" or "turn off" terms of g, to emulate one of the different possible polynomials $g_0, \ldots, g_{k-1}$. The total degree of g is at most $\log k$ plus the maximum total degree amongst $g_0, \ldots, g_{k-1}$.

Lemma 2 (Plonkish to CCS transformation)**.** *Given a Plonkish structure $\mathcal{S}_{Plonkish} = (m, n, \ell, t, q, d, e, g, T, s)$ there exists a CCS structure $\mathcal{S}_{CCS}$ such that the following holds. For any instance $\mathcal{I}_{Plonkish} = x$ and any witness vector $w_{Plonkish} = w$, the tuple $(\mathcal{S}_{CCS}, x)$ is satisfied by w if and only if $(\mathcal{S}_{Plonkish}, x)$ is satisfied by w. The natural NP checker's time to verify the tuple $((\mathcal{S}_{CCS}, x), w)$ is identical to the NP checker's time to verify the tuple $((\mathcal{S}_{Plonkish}, x), w)$ that evaluates g monomial-by-monomial when checking that Equation (4) holds.*

Proof. Let $w_{\text{CCS}} = w_{\text{Plonkish}}$ and $\mathcal{I}_{\text{CCS}} = \mathcal{I}_{\text{Plonkish}}$.

Let $\mathcal{S}_{\text{CCS}} = (m, n, N, \ell, t, q, d, [M_0, \ldots, M_{t-1}], [S_0, \ldots, S_{q-1}], [c_0, \ldots, c_{q-1}])$, where m, n, ℓ, t, q, d are from $\mathcal{S}_{\text{Plonkish}}$. We now specify the remaining entries in $\mathcal{S}_{\text{CCS}}$.

Deriving $M_0, \ldots, M_{t-1}$, and N. Recall that g in $\mathcal{S}_{\text{Plonkish}}$ is a multivariate polynomial in t variables. Furthermore the vectors $T_0, \ldots, T_{m-1}$ in $\mathcal{S}_{\text{Plonkish}}$ specify indexes into a vector of size $n+e$ (indexed from 0), where the trailing e entries are provided by the selector constants s in $\mathcal{S}_{\text{Plonkish}}$.

Unless set to a specific value below, any entry in $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times n}$ holds the additive identity 0 of $\mathbb{F}$. There is a row for each constraint in $\mathcal{S}_{\text{Plonkish}}$, so, it suffices to specify how the ith row of these matrices is set. For all $j \in \{0, 1, \ldots, t-1\}$, let $k_j = T_i[j]$.

- If $k_j \geq n$, we set $M_j[i][0] = s[k_j - n]$. Here, note that 0 indexes the entry of vector z in Equation (3) that is fixed to 1.
- Otherwise, we set $M_j[i][k_j] = 1$.

We set $\mathcal{S}_{\text{CCS}}.N$ to be the total number of non-zero entries in $M_0, \ldots, M_{t-1}$.

Deriving $S_0, \ldots, S_{q-1}$, and $c_0, \ldots, c_{q-1}$. Recall that g is a multivariate polynomial in t variables with q monomials where the degree of each monomial is at most d. For $i \in \{0, 1, \ldots, q-1\}$, set c_i to the coefficient of the ith monomial of g. For $i \in \{0, 1, \ldots, q-1\}$, if the ith monomial contains a variable j, where $j \in \{0, 1, \ldots, t-1\}$, add j to multiset S_i with multiplicity equal to the degree of the variable.

By inspection, the tuple $(\mathcal{S}_{\text{CCS}}, \mathcal{I}_{\text{CCS}})$ is satisfied by w_{CCS} if and only if $(\mathcal{S}_{\text{Plonkish}}, \mathcal{I}_{\text{Plonkish}})$ is satisfied by w_{Plonkish}. Finally, the asserted bounds on the NP checker's time are immediate from the construction. □

Remark 5. The NP-checker for Plonkish that evaluates g in $O(qd)$ time by proceeding monomial-by-monomial may be sub-optimal, as some polynomials g may have a faster evaluation procedure. Even in this situation, our reduction from Plonkish to CCS may not introduce overheads that are relevant to SNARK prover time. This is because while the number of *field operations* performed by our prover in our SNARK for CCS grows with q, the amount of cryptographic operations (i.e., the number of field elements that must be cryptographically committed) is independent of q. Often, it is the cryptographic work that is the runtime bottleneck for the SNARK prover.

Remark 6. For CCS arising from Plonkish, certain sparse matrices in the CCS structure have the property that for a sparse matrix M in the CCS structure, there exists a fixed vector $v \in \mathbb{F}^n$ such that $M \cdot z = v$ for any satisfying assignment $z \in \mathbb{F}^n$. Specifically, such matrices arise in our Plonkish to CCS transformation when encoding "selectors" in Plonkish. For such matrices, SNARKs for CCS such as SuperSpartan can commit to v instead of M in the preprocessing phase. Furthermore, at the time of proving, the prover can use v in place of $M \cdot z$ (this minimizes the complexity and costs associated with sparse polynomial commitment schemes).

2.3 Representing AIR with CCS

We first recall the AIR constraint system, and then provide a transformation from AIR to CCS. WLOG, we assume that the AIR constraint system uses a public input.

Definition 2.4 (AIR)**.** An AIR structure $\mathcal{S}$ consists of:

- size bounds $m, t, q, d \in \mathbb{N}$, where t is even;
- a multivariate polynomial g in t variables, where g is expressed as a sum of q monomials and each monomial has total degree at most d.

An AIR instance consists of public input and output $x \in \mathbb{F}^t$. An AIR witness consists of a vector $w \in \mathbb{F}^{(m-1)\cdot t/2}$. Let

$$z = (x[..t/2], w, x[t/2+1..]) \in \left(\mathbb{F}^{t/2}\right)^{m+1}, \tag{6}$$

and $x[..t/2], x[t/2+1..] \in \mathbb{F}^{t/2}$ refer to the first half and second half of x. Let us index the $m+1$ "rows" of z as $0, 1, \ldots, m$. An AIR structure-instance tuple $(\mathcal{S}, x)$ is satisfied by an AIR witness w if:

$$\text{for all } i \in \{1, \ldots, m\},\ g(z[(i-1)\cdot t/2+1], \ldots, z[i \cdot t/2], z[i \cdot t/2+1], \ldots, z[(i+1)\cdot t/2]) = 0. \tag{7}$$

Intuition for Definition 2.4 and comparison to prior definitions of AIR. Conceptually, the AIR assignment z consists of $m+1$ rows each with $t/2$ columns, and Definition 2.4 requires that the "constraint polynomial" g evaluates to zero when evaluated on every pair of adjacent rows of z, i.e., rows $i-1$ and i for $i = 1, \ldots, m$. The first row of z, namely the first $t/2$ entries of x, should be thought of as the public input to the computation. The last row of z, namely the last $t/2$ entries of x, should be thought of as the claimed public output of the computation.

Each of the $t/2$ columns of z are often viewed as the "execution trace" of a specific register when a CPU is run for m steps. The constraint polynomial g then takes as input the state of the machine's registers at two adjacent steps $i-1$ and i and checks that the value assigned to the registers at step i correctly follows by applying the CPU for one step starting in the state given by row $i-1$ of z.

Handling many AIR constraint polynomials rather than 1**.** To represent the constraints capturing a single step of a CPU, typically many polynomials g will be needed, say $g_1, \ldots, g_k$. In practice, k may be in the many dozens to hundreds [GPR21, BGtRZt23]. There is, however, a straightforward and standard randomized reduction from AIR with $k > 1$ constraint polynomials to AIR with a single constraint polynomial g: the verifier picks a random $r \in \mathbb{F}$ and sends it to the prover, and the prover replaces $g_1, \ldots, g_k$ with the

single constraint polynomial

$$g \coloneqq \sum_{i=0}^{k-1} r^i \cdot g_i. \tag{8}$$

If Equation (7) holds for all $g_1, \ldots, g_k$ in place of g, then with probability 1 over the choice of r the same will hold for the random linear combination g. Meanwhile, if Equation (7) fails to hold for any g_i then with probability at least $1 - (k-1)/|\mathbb{F}|$, it will fail to hold for the random linear combination g.[2]

Comparison to previous definitions of AIR. As discussed in the introduction, many prior works have defined AIR instances in a manner tailored to the proof systems typically used to prove their satisfiability, sometimes called STARKs [BBHR19b, Sta21, BSCKL23]. These definitions index rows of the witness vector z by powers h^i of a generator h of a cyclic subgroup G of $\mathbb{F}$, with h^i indexing the i'th row of z. Our definition of AIR is essentially equivalent to these definitions, though we naturally index rows of z by integers $\{0, \ldots, m-1\}$. One difference is that prior definitions of AIR allow each constraint polynomial to be coupled with an associated subset of rows, such that the constraint is only required to hold for rows in that subset. The subset must be a subgroup of G, which in practice limits one to "periodic" constraints, meaning the constraint applies to every k'th row where k is a power of 2. Our SNARKs in this work are easily modified to support such periodic constraints (see Remark 9).

Lemma 3 (AIR to CCS transformation)**.** *Given an AIR structure $\mathcal{S}_{AIR} = (m, t, q, d, g)$, instance $\mathcal{I}_{AIR} = x$, and witness $w_{AIR} = w$, there exists a CCS structure $\mathcal{S}_{CCS}$ such that the tuple $(\mathcal{S}_{CCS}, x)$ is satisfied by w if and only if $(\mathcal{S}_{AIR}, x)$ is satisfied by w. The natural NP checker's time to verify the tuple $((\mathcal{S}_{CCS}, x), w)$ is identical to the runtime of the NP checker that evaluates g monomial-by-monomial when checking that $((\mathcal{S}_{AIR}, x), w)$ satisfies Equation* (7)*.*

Proof. Let $w_{\text{CCS}} = w_{\text{AIR}}$ and $\mathcal{I}_{\text{CCS}} = \mathcal{I}_{\text{AIR}}$.

Let $\mathcal{S}_{\text{CCS}} = (m, n, N, \ell, t, q, d, [M_0, \ldots, M_{t-1}], [S_0, \ldots, S_{q-1}], [c_0, \ldots, c_{q-1}])$, where m, t, q, d are from $\mathcal{S}_{\text{AIR}}$. We now specify the remaining entries in $\mathcal{S}_{\text{CCS}}$.

Deriving ℓ and n. Let $\ell \leftarrow t/2$ and $n \leftarrow m \cdot t/2$.

Deriving $M_0, \ldots, M_{t-1}$, and N. Recall that g in $\mathcal{S}_{\text{AIR}}$ is a multivariate polynomial in t variables.

Unless set to a specific value below, any entry in $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times n}$ equals 0, the additive identity of $\mathbb{F}$. There is a CCS row for each of the m constraints in $\mathcal{S}_{\text{AIR}}$, so, if we index the CCS rows by $\{0, \ldots, m-1\}$, it suffices to specify how the ith row of these matrices is set, for $i = 0, \ldots, m-1$. We consider three cases. For all $j \in \{0, 1 \ldots, t-1\}$, let $k_j = i \cdot t/2 + j$.

- If $i = 0$ and $j < t/2$, we set $M_j[i][j + |w_{\text{AIR}}|] = 1$.
- If $i = m - 1$ and $j \geq t/2$, we set
$$M_j[i][j + |w_{\text{AIR}}| + t/2] = 1.$$
- Otherwise, we set $M_j[i][k_j] = 1$.

Conceptually, the third case ensures that the i'th CCS constraint applies the polynomial g to the variables in rows i and $i+1$ of the AIR witness w_{AIR}. The first two cases address a nuisance that the vector z for AIR (Equation (6)) has the public input (first half of x) in the first "row" of z and the public output (second half of x) as the final "row" of z, while the corresponding vector $z = (w, x, 1)$ for CCS (Equation (3)) has the entirety of x after w. Specifically, the first case addresses that in CCS the first half of x comes immediately after w, and ensures the zero'th CCS constraint applies g to the first half of x and the first row of w. The second case addresses that in CCS the second half of x begins $t/2$ locations after w and ensures that the final CCS constraint applies g to the final row of w and the second half of x.

We set $\mathcal{S}_{\text{CCS}}.N$ to be the total number of non-zero entries in $M_0, \ldots, M_{t-1}$.

[2] As is standard, an alternative reduction is to have the verifier choose k independent random values $r_1, \ldots, r_k$ and let $g = \sum_{i=1}^{k} r_i g_i$. This introduces soundness error just $1/|\mathbb{F}|$ rather than $(k-1)/|\mathbb{F}|$.

Deriving $S_0, \ldots, S_{q-1}$, and $c_0, \ldots, c_{q-1}$. Recall that g is a multivariate polynomial in t variables with q monomials where the degree of each monomial is at most d. For $i \in \{0, 1, \ldots, q-1\}$, set c_i to the coefficient of the ith monomial of g. For $i \in \{0, 1, \ldots, q-1\}$, if the ith monomial contains a variable j, where $j \in \{0, 1, \ldots, t-1\}$, add j to multiset S_i with multiplicity equal to the degree of the variable.

By inspection, the tuple $(\mathcal{S}_{\text{CCS}}, \mathcal{I}_{\text{CCS}})$ is satisfied by w_{CCS} if and only if $(\mathcal{S}_{\text{AIR}}, \mathcal{I}_{\text{AIR}})$ is satisfied by w_{AIR}. Finally, the asserted bounds on the NP checker's time are immediate from the construction. □

As with Remark 5, the NP-checker for AIR that evaluates g in $O(qd)$ time by proceeding monomial-by-monomial may be sub-optimal, as some polynomials g may have a faster evaluation procedure. Even in this situation, our reduction from AIR to CCS may not not introduce overheads that are relevant to SNARK prover time, because the cryptographic work performed by the SNARK prover (i.e., number of field elements that are cryptographically committed) does not grow with q, only the number of field operations do.

Remark 7. The CCS instances arising from the AIR → CCS of Lemma 3 have public input $x \in \mathbb{F}^t$ that is much shorter than the witness $w \in \mathbb{F}^{(m-1)t/2}$ (unless $m \leq 3$). As discussed later, in the proof of Theorem 1 and Section 5, it will eventually be convenient to treat x as having length (one less than) that of w by padding x with zeros. This ensures that the length n of the CCS vector z (Equation (3)), as well as length of the AIR witness w and vector $(1, x)$, can all be powers of 2. This is important for avoiding pre-processing costs for the verifier when our SNARK for AIR is applied to the CCS instances arising from the AIR → CCS transformation of Lemma 3 (see Section 5). As discussed later (see the paragraph on having the verifier evaluate $\widetilde{z}$ efficiently in the proof of Theorem 1), this padding technique does *not* result in a verifier that runs in time linear in m and n, and in fact does not increase the verifier's time by more than an additive constant.

3 Preliminaries

3.1 Multilinear extensions

An ℓ-variate polynomial $p\colon \mathbb{F}^\ell \to \mathbb{F}$ is said to be *multilinear* if p has degree at most one in each variable. Let $f\colon \{0,1\}^\ell \to \mathbb{F}$ be any function mapping the ℓ-dimensional Boolean hypercube to a field $\mathbb{F}$. A polynomial $g\colon \mathbb{F}^\ell \to \mathbb{F}$ is said to *extend* f if $g(x) = f(x)$ for all $x \in \{0,1\}^\ell$. It is well-known that for any $f\colon \{0,1\}^\ell \to \mathbb{F}$, there is a unique *multilinear* polynomial $\widetilde{f}\colon \mathbb{F} \to \mathbb{F}$ that extends f. The polynomial $\widetilde{f}$ is referred to as the *multilinear extension* (MLE) of f.

A particular multilinear extension that arises frequently in the design of interactive proofs is the $\widetilde{\mathsf{eq}}$ is the MLE of the function $\mathsf{eq} : \{0,1\}^s \times \{0,1\}^s \to \mathbb{F}$ defined as follows:

$$\mathsf{eq}(x, e) = \begin{cases} 1 & \text{if } x = e \\ 0 & \text{otherwise.} \end{cases}$$

An explicit expression for $\widetilde{\mathsf{eq}}$ is:

$$\widetilde{\mathsf{eq}}(x, e) = \prod_{i=1}^{s} \left(x_i e_i + (1 - x_i)(1 - e_i)\right). \tag{9}$$

Indeed, one can easily check that the right hand side of Equation (9) is a multilinear polynomial, and that if evaluated at any input $(x, e) \in \{0,1\}^s \times \{0,1\}^s$, it outputs 1 if $x = e$ and 0 otherwise. Hence, the right hand side of Equation (9) is the unique multilinear polynomial extending eq. Equation (9) implies that $\widetilde{\mathsf{eq}}(r_1, r_2)$ can be evaluated at any point $(r_1, r_2) \in \mathbb{F}^s \times \mathbb{F}^s$ in $O(s)$ time.[3]

[3]Throughout this manuscript, we consider any field addition or multiplication to require constant time.

Multilinear extensions of vectors. Given a vector $u \in \mathbb{F}^m$, we will often refer to the *multilinear extension of* u and denote this multilinear polynomial by $\widetilde{u}$. $\widetilde{u}$ is obtained by viewing u as a function mapping $\{0,1\}^{\log m} \to \mathbb{F}$ in the natural way[4]: the function interprets its $(\log m)$-bit input $(i_0, \ldots, i_{\log m - 1})$ as the binary representation of an integer i between 0 and $m-1$, and outputs u_i. $\widetilde{u}$ is defined to be the multilinear extension of this function.

Lagrange interpolation. An explicit expression for the MLE of any function is given by the following standard lemma (see [Tha20, Lemma 3.6]).

Lemma 4. *Let $f\colon \{0,1\}^\ell \to \mathbb{F}$ be any function. Then the following multilinear polynomial $\widetilde{f}$ extends f:*

$$\widetilde{f}(x_0, \ldots, x_{\ell-1}) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \chi_w(x_0, \ldots, x_{\ell-1}), \tag{10}$$

where, for any $w = (w_0, \ldots, w_{\ell-1})$, $\chi_w(x_0, \ldots, x_{\ell-1}) \coloneqq \prod_{i=0}^{\ell} (x_i w_i + (1-x_i)(1-w_i))$. Equivalently, $\chi_w(x_0, \ldots, x_{\ell-1}) = \widetilde{\mathsf{eq}}(x_0, \ldots, x_{\ell-1}, w_0, \ldots, w_{\ell-1})$.

The polynomials $\{\chi_w\colon w \in \{0,1\}^\ell\}$ are called the *Lagrange basis polynomials* for ℓ-variate multilinear polynomials. The evaluations $\{\widetilde{f}(w)\colon w \in \{0,1\}^\ell\}$ are sometimes called the coefficients of $\widetilde{f}$ *in the Lagrange basis*, terminology that is justified by Equation (10).

The sum-check protocol. Let g be some ℓ-variate polynomial defined over a finite field $\mathbb{F}$. The purpose of the sum-check protocol is for prover to provide the verifier with the following sum:

$$H \coloneqq \sum_{b \in \{0,1\}^\ell} g(b). \tag{11}$$

To compute H unaided, the verifier would have to evaluate g at all 2^ℓ points in $\{0,1\}^\ell$ and sum the results. The sum-check protocol allows the verifier to offload this hard work to the prover. It consists of ℓ rounds, one per variable of g. In round i, the prover sends a message consisting of d_i field elements, where d_i is the degree of g in its i'th variable, and the verifier responds with a single (randomly chosen) field element. If the prover is honest, this polynomia (in the single variable X_i) is

$$\sum_{(b_{i+1}, \ldots, b_{\ell-1}) \in \{0,1\}^{\ell-i}} g(r_0, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_{\ell-1}). \tag{12}$$

Here, we are indexing both the rounds of the sum-check protocol and the variables of g starting from zero (i.e., indexing them by $\{0, 1, \ldots, \ell-1\}$), and $r_0, \ldots, r_{i-1}$ are random field elements chosen by the verifier across rounds $0, \ldots, i-1$ of the protocol.

The verifier's runtime is $O\left(\sum_{i=1}^{\ell} d_i\right)$, plus the time required to evaluate g at a single point $r \in \mathbb{F}^\ell$. In the typical case that $d_i = O(1)$ for each round i, this means the total verifier time is $O(\ell)$, plus the time required to evaluate g at a single point $r \in \mathbb{F}^\ell$. This is exponentially faster than the 2^ℓ time that would generally be required for the verifier to compute H. See [AB09, Chapter 8] or [Tha20, §4.1] for details.

SNARKs We adapt the definition provided in [KST22].

Definition 3.1. Consider a relation $\mathcal{R}$ over public parameters, structure, instance, and witness tuples. A non-interactive argument of knowledge for $\mathcal{R}$ consists of PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$, denoting the generator, the prover, the verifier and the encoder respectively with the following interface.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: On input security parameter λ, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (pk, \mathsf{vk})$: On input structure s, representing common structure among instances, outputs the prover key pk and verifier key vk.

[4] All logarithms in this paper are to base 2.

Scheme	Commit Size	Proof Size	$\mathcal{V}$ time	Commit time	$\mathcal{P}$ time
Brakedown-PC	$1\ \lvert\mathbb{H}\rvert$	$O(\sqrt{N\cdot\lambda})\ \lvert\mathbb{F}\rvert$	$O(\sqrt{N\cdot\lambda})\ \mathbb{F}$	$O(N)\ \mathbb{F}$	$O(N)\ \mathbb{F}$
Orion-PC	$1\ \lvert\mathbb{H}\rvert$	$O(\lambda\log^2 N)\ \lvert\mathbb{H}\rvert$	$O(\lambda\log^2 N)\ \mathbb{H}$	$O(N)\ \mathbb{F}$	$O(N)\ \mathbb{F}$
Hyrax-PC	$O(\sqrt{N})\ \lvert\mathbb{G}\rvert$	$O(\sqrt{N})\ \lvert\mathbb{G}\rvert$	$O(\sqrt{N})\ \mathbb{G}$	$O(N)\ \mathbb{G}$	$O(N)\ \mathbb{F}$
Dory-PC	$1\ \lvert\mathbb{G}_T\rvert$	$O(\log N)\ \lvert\mathbb{G}_T\rvert$	$O(\log N)\ \mathbb{G}_T$	$O(N)\ \mathbb{G}_1$	$O(N)\ \mathbb{F}$
KZG + Gemini	$1\ \mathbb{G}_1$	$O(\log N)\ \mathbb{G}_1$	$O(\log N)\ \mathbb{G}_1$	$O(N)\ \mathbb{G}_1$	$O(N)\ \mathbb{G}_1$

Figure 1: Costs of polynomial commitment schemes, for ℓ-variate multilinear polynomials over $\mathbb{F}$, with $N = 2^\ell$. $\mathcal{P}$ time refers to the time to compute evaluation proofs. PST [PST13, ZGK$^+$17] achieves similar performance as the costs depicted for "KZG + Gemini". Orion-PC dominates Virgo [ZXZS20]. In addition to the reported $O(N)$ field operations, Hyrax-PC and Dory-PC require roughly $O(N^{1/2})$ cryptographic work to compute evaluation proofs. $\mathbb{F}$ refers to a finite field, $\mathbb{H}$ refers to a collision-resistant hash, $\mathbb{G}$ refers to a cryptographic group where DLOG is hard, and $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ refer to pairing-friendly groups. Columns with a suffix of "size" depict to the number of elements of a particular type, and columns with a suffix of "time" depict the number of operations (e.g., field multiplications or the size of multi-exponentiations). Orion-PC also requires $O(\sqrt{N})$ (transparent) pre-processing time for the verifier; the pre-processing phase depends only on the size of the ℓ-variate multilinear polynomial being committed.

- $\mathcal{P}(pk, u, w) \to \pi$: On input instance u and witness w, outputs a proof π proving that $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
- $\mathcal{V}(\mathsf{vk}, u, \pi) \to \{0, 1\}$: On input the verifier key vk, instance u, and a proof π, outputs 1 if the instance is accepting and 0 otherwise.

A non-interactive argument of knowledge satisfies *completeness* if for any PPT adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{c|l} \mathcal{V}(\mathsf{vk}, u, \pi) = 1 & \begin{array}{l}\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda),\\ (\mathsf{s}, (u, w)) \leftarrow \mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R},\\ (pk, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}),\\ \pi \leftarrow \mathcal{P}(pk, u, w)\end{array}\end{array}\right] = 1.$$

A non-interactive argument of knowledge satisfies *knowledge soundness* if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that for all randomness ρ

$$\Pr\left[\begin{array}{c|l} \begin{array}{l}\mathcal{V}(\mathsf{vk}, u, \pi) = 1,\\ (\mathsf{pp}, \mathsf{s}, u, w) \notin \mathcal{R}\end{array} & \begin{array}{l}\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda),\\ (\mathsf{s}, u, \pi) \leftarrow \mathcal{A}(\mathsf{pp}; \rho),\\ (pk, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}),\\ w \leftarrow \mathcal{E}(\mathsf{pp}, \rho)\end{array}\end{array}\right] = \mathrm{negl}(\lambda).$$

A non-interactive argument of knowledge is succinct if the verifier's time to check the proof π and the size of the proof π are at most polylogarithmic in the size of the statement proven.

Polynomial commitment scheme We adapt the definition from [BFS20]. A polynomial commitment scheme for multilinear polynomials is a tuple of four protocols $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$:

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell)$: takes as input ℓ (the number of variables in a multilinear polynomial); produces public parameters pp.
- $\mathcal{C} \leftarrow \mathsf{Commit}(\mathsf{pp}, g)$: takes as input a ℓ-variate multilinear polynomial over a finite field $g \in \mathbb{F}[\ell]$; produces a commitment $\mathcal{C}$.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g)$: verifies the opening of commitment $\mathcal{C}$ to the ℓ-variate multilinear polynomial $g \in \mathbb{F}[\ell]$; outputs $b \in \{0, 1\}$.
- $b \leftarrow \mathsf{Eval}(pp, \mathcal{C}, r, v, \ell, g)$ is a protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}$, the number of variables ℓ, a scalar $v \in \mathbb{F}$, and $r \in \mathbb{F}^\ell$. $\mathcal{P}$ additionally knows a *ell*-variate multilinear polynomial $g \in \mathbb{F}[\ell]$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $g(r) = v$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0, 1\}$.

Definition 3.2. A tuple of four protocols $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is an extractable polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$ if the following conditions hold.

- **Completeness.** For any ℓ-variate multilinear polynomial $g \in \mathbb{F}[\ell]$,

$$\Pr\left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell);\ \mathcal{C} \leftarrow \mathsf{Commit}(\mathsf{pp}, g): \\ \mathsf{Eval}(\mathsf{pp}, \mathcal{C}, r, v, \ell, g) = 1 \wedge v = g(r) \end{array} \right\} \geq 1 - \mathrm{negl}(\lambda)$$

- **Binding.** For any PPT adversary $\mathcal{A}$, size parameter $\ell \geq 1$,

$$\Pr\left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell);\ (\mathcal{C}, g_0, g_1) = \mathcal{A}(\mathsf{pp}); \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g_0);\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g_1): \\ b_0 = b_1 \neq 0 \wedge g_0 \neq g_1 \end{array} \right\} \leq \mathrm{negl}(\lambda)$$

- **Knowledge soundness.** Eval is a succinct argument of knowledge for the following NP relation given $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell)$.

$$\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}) = \{\langle (\mathcal{C}, r, v), (g) \rangle : g \in \mathbb{F}[\mu] \wedge g(r) = v \wedge \mathsf{Open}(pp, \mathcal{C}, g) = 1\}$$

3.2 Polynomial IOPs and polynomial commitments

Modern SNARKs are constructed by combining a type of interactive protocol called a *polynomial IOP* [BFS20] with a cryptographic primitive called a *polynomial commitment scheme* [KZG10]. The combination yields a succinct *interactive* argument, which can then be rendered non-interactive via the Fiat-Shamir transformation [FS86], yielding a SNARK.

Roughly, a polynomial IOP is an interactive protocol where, in one or more rounds, the prover may "send" to the verifier a very large polynomial g. Because g is so large, one does not wish for the verifier to read a complete description of g. Instead, in any efficient polynomial IOP, the verifier only "queries" g at one point (or a handful of points). This means that the only information the verifier needs about g to check that the prover is behaving honestly is one (or a few) evaluations of g.

In turn, a polynomial commitment scheme enables an untrusted prover to succinctly *commit* to a polynomial g, and later provide to the verifier any evaluation $g(r)$ for a point r chosen by the verifier, along with a proof that the returned value is indeed consistent with the committed polynomial. Essentially, a polynomial commitment scheme is exactly the cryptographic primitive that one needs to obtain a succinct argument from a polynomial IOP. Rather than having the prover send a large polynomial g to the verifier as in the polynomial IOP, the argument system prover instead cryptographically commits to g and later reveals any evaluations of g required by the verifier to perform its checks.

Whether or not a SNARK requires a trusted setup, as well as whether or not it is plausibly post-quantum secure, is determined by the polynomial commitment scheme used. If the polynomial commitment scheme does not require a trusted setup, neither does the resulting SNARK, and similarly if the polynomial commitment scheme is plausibly binding against quantum adversaries, then the SNARK is plausibly post-quantum sound.

SuperSpartan can make use of any commitment schemes for *multilinear* polynomials g.[5] Here an ℓ-variate multilinear polynomial $g \colon \mathbb{F}^\ell \to \mathbb{F}$ is a polynomial of degree at most one in each variable. A brief summary of the multilinear polynomial commitment schemes that are most relevant to this work is provided in Figure 3.1. All of the schemes in the figure, except for KZG-based scheme, are transparent; Brakedown-PC and Orion-PC are plausibly post-quantum secure.

4 SuperSpartan's polynomial IOP for CCS

This section describes SuperSpartan's polynomial IOP for CCS. It is a straightforward generalization of polynomial IOP for R1CS in Spartan [Set20]. Appendix C describes a natural extension of SuperSpartan's polynomial IOP to additionally handle CCS+ (i.e., CCS with lookup operations).

[5] Any univariate polynomial commitment scheme can be transformed into a multilinear one, though the transformations introduce some overhead (see, e.g., [CBBZ23, BCHO22, ZXZS20]).

As per Definition 2.2, suppose we are given a CCS structure-instance tuple with size bounds

$$((m, n, \ell, t, q, d), (M_0, \ldots, M_{t-1}, s_0, \ldots, s_{q-1}), x).$$

For simplicity, we describe our protocol assuming m and n are powers of two (if this is not the case, we can pad each matrix M_i and the witness vector z with zeros to extend m and n to a power of 2 without increasing the prover or verifier's time in the SNARKs resulting from the protocol below).

Interpret the matrices $M_0, \ldots, M_{t-1}$ as functions mapping domain $\{0,1\}^{\log m} \times \{0,1\}^{\log n}$ to $\mathbb{F}$ in the natural way. That is, an input in $\{0,1\}^{\log m} \times \{0,1\}^{\log n}$ is interpreted as the binary representation of an index $(i, j) \in \{0, 1, \ldots, m-1\} \times \{0, 1, \ldots, n-1\}$, and the function outputs the (i, j)'th entry of the matrix.

Theorem 1 (A generalization of [Set20] from R1CS to CCS)**.** *For any finite field $\mathbb{F}$, there exists a polynomial IOP for $\mathcal{R}_{CCS}$, with the following parameters, where $m \times n$ denotes the dimensionality of the CCS coefficient matrices, and N denotes the total number of non-zero entries across all of the matrices:*

- *soundness error is $O((t + d) \log m)/|\mathbb{F}|$*
- *round complexity is $O(\log m + \log n)$;*
- *communication complexity is $O(d \log m + \log n)$ elements of $\mathbb{F}$;*
- *at the start of the protocol, the prover sends a single $(\log m - 1)$-variate multilinear polynomial $\widetilde{W}$, and the verifier has a query access to t additional $2 \log m$-variate multilinear polynomials $\widetilde{M_0}, \ldots, \widetilde{M_{t-1}}$;*
- *the verifier makes a single evaluation query to each of the polynomials $\widetilde{W}$, $\widetilde{M_0}, \ldots, \widetilde{M_{t-1}}$, and otherwise performs $O(dq + d \log m + \log n)$ operations over $\mathbb{F}$;*
- *the prescribed prover performs $O(N + tm + qmd \log^2 d)$ operations over $\mathbb{F}$ to compute its messages over the course of the polynomial IOP (and to compute answers to the verifier's four queries to $\widetilde{W}, \widetilde{M_0}, \ldots, \widetilde{M_{t-1}}$).*

Remark 8. The $O(qmd \log^2 d)$ field operations term in the prover's runtime from Theorem 1 involves using FFTs to multiply together d different degree-1 polynomials. FFTs are only practical over some finite fields. Moreover, in CCS/AIR/Plonkish instances that arise in practice, d rarely, if ever, exceeds 100 and is often as low as 5 or even 2 [GPR21, BGtRZt23]. Hence, these $O(d \log^2 d)$-time FFT algorithms will typically be much slower than the naive $O(d^2)$-time algorithm for multiplying together d degree-1 polynomials. Using Karatsuba's algorithm in place of FFTs would also yield a sub-quadratic-in-d time algorithm that works over any field.

Proof of Theorem 1. For a CCS structure and instance, $\mathcal{I} = (M_0, \ldots, M_{t-1}, s_0, \ldots, s_{q-1}, x)$ and a purported witness W, let $Z = (W, 1, x)$. As explained prior to the theorem statement, we can interpret $M_0, \ldots, M_{t-1}$ as functions mapping $\{0,1\}^{\log m} \times \{0,1\}^{\log n}$ to $\mathbb{F}$, and similarly we interpret Z and $(1, x)$ as functions with the following respective signatures in the same manner: $\{0,1\}^{\log n} \to \mathbb{F}$ and $\{0,1\}^{\log(n)-1} \to \mathbb{F}$.

Having the verifier evaluate $\widetilde{Z}$ efficiently. Let us first assume that the CCS witness W and $(1, x)$ both have the same length $n/2$. Then it is easy to check that the MLE $\widetilde{Z}$ of Z satisfies

$$\widetilde{Z}(X_0, \ldots, X_{\log n-1}) = (1 - X_0) \cdot \widetilde{W}(X_1, \ldots, X_{\log n-1}) + X_0 \cdot \widetilde{(1, x)}(X_1, \ldots, X_{\log n-1}) \tag{13}$$

Indeed, the right hand side of Equation (13) is a multilinear polynomial, and it is easily checked that $\widetilde{Z}(x_0, \ldots, x_{\log n-1}) = Z(x_0, \ldots, x_{\log n-1})$ for all $x_0, \ldots, x_{\log n-1} \in \{0,1\}^{\log n}$ (since the first $n/2$ of the evaluations of Z are given by W and the rest are given by the vector $(1, x)$). Hence, the right hand side of Equation (13) must be the unique multilinear extension of Z.

If the length of $(1, x)$ is less than that of W (as is the case in the CCS instances resulting from the AIR $\to$ CCS transformation of Lemma 3), we replace x with a padded vector v that appends zeros to x until it has the same length as W (and we henceforth use n to denote twice the length of this padded vector). Replacing $(1, x)$ with the padded vector does increase the time required for the verifier to compute $\widetilde{(1, x)}(X_1, \ldots, X_{\log n-1})$ by

more than an *additive* $O(\log n)$ field operations. For example, if the unpadded vector $(1, x)$ has length 2^ℓ, it is easy to check that the padded vector of length $n/2$ has multilinear extension equal to

$$(X_1, \ldots, X_{\log(n)-1}) \mapsto (1 - X_{\ell+1}) \cdot (1 - X_{\ell+2}) \cdot \cdots \cdot (1 - X_{\log(n)-1}) \cdot \widetilde{(1,x)}(X_1, \ldots, X_\ell).$$

Indeed, this is a multilinear polynomial that agrees with the padded vector $(1, x)$ at all inputs $(x_1, \ldots, x_{\log(n)-1}) \in \{0,1\}^{\log(n)-1}$, and hence must equal the unique multilinear polynomial extending the padded vector.

The protocol. Have the verifier pick $\tau \in \mathbb{F}^{\log m}$ at random. Similar to [Set20, Theorem 4.1], checking if $(\mathcal{I}, W) \in \mathcal{R}_{\text{CCS}}$ is equivalent, except for a soundness error of $\log m/|\mathbb{F}|$ over the choice of $\tau \in \mathbb{F}^{\log m}$, to checking if the following identity holds:

$$0 \stackrel{?}{=} \left(\sum_{a \in \{0,1\}^{\log m}} \widetilde{eq}(\tau, a) \cdot \sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log n}} \widetilde{M}_j(a, y) \cdot \widetilde{Z}(y) \right) \right) \tag{14}$$

where recall from Section 3.1 that $\widetilde{eq}$ is the MLE of $eq : \{0,1\}^{\log m} \times \{0,1\}^{\log m} \to \mathbb{F}$:

$$eq(a, e) = \begin{cases} 1 & \text{if } a = e \\ 0 & \text{otherwise.} \end{cases}$$

That is, if $(\mathcal{I}, W) \in \mathcal{R}_{\text{CCS}}$, then Equation (14) holds with probability 1 over the choice of τ, and if $(\mathcal{I}, W) \notin \mathcal{R}_{\text{CCS}}$, then Equation (14) holds with probability at most $O(\log m/|\mathbb{F}|)$ over the random choice of τ.

Consider computing the right hand side of Equation (14) by applying the sum-check protocol to the polynomial

$$g(a) \coloneqq \widetilde{eq}(\tau, a) \cdot \sum_{i=0}^{q-1} c_i \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{M}_j(a, y) \cdot \widetilde{Z}(y) \right).$$

From the verifier's perspective, this reduces the task of computing the right hand side of Equation (14) to the task of evaluating g at a random input $r_a \in \mathbb{F}^{\log m}$. Note that the verifier can evaluate $\widetilde{eq}(\tau, r_a)$ unassisted in $O(\log m)$ field operations, as it is easily checked (see Equation (9)) that

$$\widetilde{eq}(\tau, r_a) = \prod_{i=1}^{\log m} \left(\tau_i r_{a,i} + (1 - \tau_i)(1 - r_{a,i}) \right).$$

With $\widetilde{eq}(\tau, r_a)$ in hand, $g(r_a)$ can be computed in $O(dq)$ time given the following t quantities for $i \in \{0, 1, \ldots, t-1\}$:

$$\sum_{y \in \{0,1\}^{\log n}} \widetilde{M}_i(r_a, y) \cdot \widetilde{Z}(y).$$

These t quantities can be computed by applying the sum-check protocol t more times in parallel, once to each of the following polynomials, where $i \in \{0, 1, \ldots, t-1\}$ (to reduce communication costs by a factor of t, pick a random $\gamma \in \mathbb{F}$ and take linear combination with weights given by $[\gamma^0, \ldots, \gamma^{t-1}]$):[6]

$$\widetilde{M}_i(r_a, y) \cdot \widetilde{Z}(y).$$

To perform the verifier's final check in this invocation of the sum-check protocol, it suffices for the verifier to evaluate each of the above t polynomials at the random vector r_y, which means it suffices for the verifier to evaluate $\widetilde{M}_i(r_a, r_y)$. These evaluations can be obtained via the verifier's assumed query access to $\widetilde{M}_0, \ldots, \widetilde{M}_{t-1}$. $\widetilde{Z}(r_y)$ can be obtained from one query to $\widetilde{W}$ and one query to $\widetilde{(1,x)}$ via Equation (13).

In summary, we have the following polynomial IOP:

[6] As with Footnote 2, using powers-of-γ as the coefficients of the random linear combination adds soundness error $(t-1)/|\mathbb{F}|$ to the polynomial IOP. One can instead choose t independent random coefficients from $\mathbb{F}$, which would add soundness error just $1/|\mathbb{F}|$.

1. $\mathcal{P} \to \mathcal{V}$: a $(\log(n)-1)$-variate multilinear polynomial $\widetilde{W}$ as an oracle.
2. $\mathcal{V} \to \mathcal{P}$: $\tau \in_R \mathbb{F}^{\log m}$
3. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check reduction described in the prose above. This entails:
   (a) $\mathcal{V} \leftrightarrow \mathcal{P}$: Apply the sum-check protocol to

$$g(a) \coloneqq \widetilde{eq}(\tau, a) \cdot \sum_{i=0}^{q-1} c_i \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log n}} \widetilde{M_j}(a, y) \cdot \widetilde{Z}(y) \right) \tag{15}$$

   to compute the following quantity, and confirm it equals 0:

$$\sum_{b \in \{0,1\}^{\log m}} g(b).$$

   This reduces checking the task of computing the right hand side of Equation (14) to the task of computing $g(r_a)$ for a random input $r_a \in \mathbb{F}^s$.
   (b) $\mathcal{V} \to \mathcal{P}$: choose a random $\gamma \in \mathbb{F}$ and send γ to $\mathcal{P}$.
   (c) $\mathcal{V} \leftrightarrow \mathcal{P}$ Apply the sum-check protocol a second time, to compute

$$\sum_{i=0}^{t-1} \sum_{y \in \{0,1\}^{\log n}} \gamma^i \cdot \widetilde{M_i}(r_a, y) \cdot \widetilde{Z}(y),$$

   and confirm that the result is consistent with the claimed value of $g(r_a)$.

   The above sum-check reduction reduces the check in Equation (14) to checking if the following hold, where r_a, r_y are respectively vectors in $\mathbb{F}^{\log m}$ and $\mathbb{F}^{\log n}$ chosen at random by the verifier over the course of the two invocations of the sum-check protocol above:
   - $\forall i \in \{0, 1, \ldots, t-1\}$, $\widetilde{M_i}(r_a, r_y) \stackrel{?}{=} v_i$, and
   - $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$.
4. $\mathcal{V}$:
   - check if $\forall i \in \{0, 1, \ldots, t-1\}$, $\widetilde{M_i}(r_x, r_y) \stackrel{?}{=} v_i$, with one query to $\widetilde{M_i}$;
   - check if $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$ by checking if: $v_Z = (1 - r_y[1]) \cdot v_W + r_y[1] \cdot \widetilde{(x, 1)}(r_y[2..])$, where $r_y[2..]$ refers to a slice of r_y without the first element of r_y, and $v_W \leftarrow \widetilde{W}(r_y[2..])$ via an oracle query (see Equation (13)).

Figure 2: Pseudocode for our polynomial IPCP for CCS (Definition 2.2).

Completeness. Perfect completeness follows from perfect completeness of the sum-check protocol and the fact that Equation (14) holds with probability 1 over the choice of τ if $(\mathcal{I}, W) \in \mathcal{R}_{\text{CCS}}$.

Soundness. Applying a standard union bound to the soundness error introduced by probabilistic check in Equation (14) with the soundness error of the sum-check protocol [LFKN90], we conclude that the soundness error for the depicted polynomial IOP as at most $O\left((d \cdot \log m + t + \log n)\right)/|\mathbb{F}|$.[7]

Round and communication complexity. The sum-check protocol is applied twice. In the first invocation of sum-check (Line 3a of Figure 2), the polynomial to which the sum-check protocol is applied has degree at most d in each of its $\log m$ variables. In the remaining invocation of sum-check, the polynomial to which sum-check is applied has degree at most 2 in each of its $\log n$ variables. Hence, the round complexity of the polynomial IOP is $\log m + \log n$ and the total communication is $O(d \log m + \log n)$ field elements.[8]

Verifier time. The asserted bounds on the verifier's runtime are immediate from the verifier's runtime in the sum-check protocol, and the fact that $\widetilde{eq}$ can be evaluated at any input $(\tau, r_a) \in \mathbb{F}^{2\log m}$ in $O(\log m)$ field operations.

Prover Time. Straightforward adaption of the linear-time Spartan prover [Set20] (which uses prior techniques for linear-time sum-checks [Tha13], see also [Tha20, Section 7.5.2] for an exposition), establishes that the honest prover in the polynomial IOP for CCS can be implemented in $O(N + tm + qmd \log^2 d)$ $\mathbb{F}$-ops . Here, recall that N denotes the number of non-zero entries in total across the constraint matrices $M_0, \ldots, M_{t-1}$. This includes the time required to compute $\widetilde{M}_i(r_a, r_y)$ for all $i \in \{0, 1, \ldots, t-1\}$ (i.e., to compute answers to the verifier's queries to the polynomials $\widetilde{M}_i$, and $\widetilde{Z}$).

These costs can be summarized as follows. In the first invocation of sum-check, the prover devotes work to the following tasks.

- $O(N)$ time is devoted to the field operations required to compute $M_i \cdot Z$ for $i = 0, 1, \ldots, t-1$. Denote the jth such vector by u_j, and recall that the first invocation of the sum-check protocol is applied to the polynomial

$$g(a) \coloneqq \widetilde{eq}(\tau, a) \cdot \sum_{i=0}^{q-1} \prod_{s \in S_i} \widetilde{u}_s(a).$$

  Per Equation (12), the prescribed prover message in round j of the sum-check protocol is the degree-(d+1) univariate polynomial h_j where

$$h_j(c) = \sum_{(a_{j+1}, a_{j+2}, \ldots, a_{\log(m)-1}) \in \{0,1\}^{\log(m)-j-1}} \widetilde{\mathsf{eq}}(\tau, r_0, \ldots, r_{j-1}, c, a_{j+1}, \ldots, a_{\log m}) \sum_{i=0}^{q-1} \prod_{s \in S_i} \widetilde{u}_s(r_0, \ldots, r_{j-1}, c, a_{j+1}, \ldots, a_{\log(m)-1}).$$

  Here, $r_1, \ldots, r_{j-1}$ are random field elements chosen by the verifier in rounds $1, 2, \ldots, j-1$.

  Given the t vectors $u_0, \ldots, u_{t-1}$, each of length m, linear-time sum-check techniques [CTY12, Tha13] can implement the prover in the first invocation of sum-check in time $O(tm + qd \log m)$ as follows.

- $O(tm)$ total time (following standard techniques [CTY12, Tha13]) suffices to compute the following set of evaluations across all rounds j of this first invocation of the sum-check protocol:

$$\left\{\widetilde{u}_s(r_0, \ldots, r_{j-1}, b) \colon s \in \{0, \ldots, t-1\}, b \in \{0,1\}^{\log(m)-j}\right\}.$$

[7] The additive term $t/|\mathbb{F}|$ can be removed from the soundness error by replacing the t powers of γ in Step (c) of the polynomial IOP with t random field elements, see Footnote 6. This increases the communication cost from the verifier to the prover by $t-1$ field elements.

[8] When compiled into a SNARK, this communication can be independent of d by using a polynomial commitment scheme.

- $O(m)$ total time (following standard techniques [CTY12, Tha13]) suffices to compute the following set of evaluations across all rounds j of this first invocation of the sum-check protocol:

$$\left\{\widetilde{eq}(\tau, r_0, \ldots, r_{j-1}, b) \colon b \in \{0,1\}^{\log(m)-j}\right\}.$$

- Given the evaluations computed in the two bullet points above, the task of computing h_j amounts to the task of repeatedly computing the product of at most $(d+1)$ degree-1 univariate polynomials over $\mathbb{F}$, and summing the results. Here, the relevant univariate polynomials in a single variable X are

$$\widetilde{u}_s(r_0, \ldots, r_{j-1}, X, a_{j+1}, \ldots, a_{\log(m)-1})$$

and

$$\widetilde{eq}(\tau, r_0, \ldots, r_{j-1}, X, b')$$

as $(a_{j+1}, \ldots, a_{\log(m)-1})$ and b' range over $\{0,1\}^{\log(m)-j-1}$.

This is because, in order to specify a degree-1 univariate polynomial, it suffices to evaluate that polynomial at the inputs $\{0,1\}$ (or any two other elements from $\mathbb{F}$ for that matter), and for each of the relevant degree-1 polynomials, the evaluations at inputs $\{0,1\}$ can be computed in $O(1)$ time from the quantities in the two bullet points above.

The number of such products to be computed is

$$O\left(\sum_{j=1}^{\log m} (m/2^j) \cdot q\right) = O(mq),$$

as in round j there is one such product for each tuple

$$(a_{j+1}, \ldots, a_{\log(m)-1}) \in \{0,1\}^{\log(m)-j-1}$$

and multiset S_i (i.e., there are $(m/2^j) \cdot q$ such products to compute in total in round j). The task of multiplying d degree-1 polynomials is reducible to the well-known task in the computer algebra literature of computing so-called *sub-product trees*, which can be done in $O(d \log^2 d)$ field operations if the field supports FFTs of length up to d.[9] The claimed runtime bound for the prover in the first sum-check instance follows.

The second sum-check invocation applies sum-check to the $(\log n)$-variate polynomial

$$\sum_{i=0}^{t-1} \gamma^i \cdot \widetilde{M_i}(r_a, y) \cdot \widetilde{Z}(y),$$

which has degree at most 2 in each variable. Standard linear-time-sum-check techniques [CTY12, Tha13] can implement the prover in this protocol in $O(N+t)$ time.

□

5 Avoiding pre-processing for uniform IRs

This section shows how to avoid preprocessing of circuit structure (to achieve succinct verification costs) for CCS instances with a "uniform" structure (e.g., a circuit with many copies of the same sub-circuit). In particular, we show that, for uniform CCS instances, the verifier can evaluate the multilinear extension polynomials $\widetilde{M_0}, \ldots, \widetilde{M_{t-1}}$ at any desired point in time logarithmic in the number of rows and columns of these matrices. We describe this for CCS instances arising from the AIR-to-CCS transformation of Lemma 3. We also discuss the case of "SIMD CCS", a natural extension of "SIMD R1CS" [TKPS22] that captures data-parallel computations.

[9] See, for example, `https://cstheory.stackexchange.com/questions/681/multiplying-n-polynomials-of-degree-1`.

5.1 The multilinear extension of the "adding 1 in binary" function.

Recall that we can index integers in $\{0, 1, \ldots, D-1\}$ via their binary representation in the natural way, with the integer $k-1$ represented by the all-1s vector and the integer 0 indexed by the all-0s vector. In this indexing, let us think of the right-most bit as the low-order bit of the binary representation.

For a bit-vector $i \in \{0,1\}^{\log D}$, let $\mathsf{to\text{-}int}(i) = \sum_{j=0}^{\log D - 1} i_j \cdot 2^j$ denote the integer that i represents. Similarly, for an integer $\kappa \in \{0, 1, \ldots, t-1\}$, let $\mathsf{bin}(\kappa)$ denote the natural binary representation of j. The number of bits in $\mathsf{bin}(\kappa)$ will always be clear from context.

Define the function

$$\mathsf{next}(i,j)\colon \{0,1\}^{\log D} \times \{0,1\}^{\log D} \to \{0,1\}$$

which takes as input two bit-vectors i and j and outputs 1 if and only if i and j represent integers $I = \mathsf{to\text{-}int}(i)$ and $J = \mathsf{to\text{-}int}(j)$ such that $J = I + 1$. One can think of this function as "adding 1 in binary": it takes as input two bit-vectors and indicates whether the second is obtained by adding 1 to (the integer represented) by the first. Note that if i is the all-1s vector, then $(i,j) = 0$ for all $j \in \{0,1\}^{\log D}$.

Theorem 2. *For any point $(r_x, r_y) \in \mathbb{F}^{\log D} \times \mathbb{F}^{\log D}$, $\widetilde{\mathsf{next}}(r_x, r_y)$ can be evaluated in logarithmic time.*

Proof. We give an explicit expression for $\widetilde{\mathsf{next}}$. Specifically:

$$\widetilde{\mathsf{next}}(x_1, \ldots, x_{\log D}, y_1, \ldots, y_{\log D}) = h(x_1, \ldots, x_{\log D}, y_1, \ldots, y_{\log D}) + g(x_1, \ldots, x_{\log D}, y_1, \ldots, y_{\log D}) \quad (16)$$

where

$$h(x_1, \ldots, x_{\log D}, y_1, \ldots, y_{\log D}) = (1 - x_{\log D}) \cdot y_{\log D} \cdot \widetilde{\mathsf{eq}}(x_1, \ldots, x_{\log D - 1}, y_1, \ldots, y_{\log D - 1})$$

and

$$g(x_1, \ldots, x_{\log D}, y_1, \ldots, y_{\log D})$$

equals

$$\sum_{k=1}^{\log D - 1} \left(\prod_{i=0}^{k-1} x_{\log D - i}(1 - y_{\log D - i}) \right) \cdot (1 - x_{\log D - k}) \cdot y_{\log D - k} \cdot \widetilde{\mathsf{eq}}\left(x_1, \ldots, x_{\log D - k - 1}, y_1, \ldots, y_{\log D - k - 1}\right).$$

Indeed, both h and g are multilinear polynomials, so to confirm that $h + g$ equals the unique multilinear extension of next, it suffices to show that $h(x,y) + g(x,y) = \mathsf{next}(x,y)$ for all $(x,y) \in \{0,1\}^{\log D} \times \{0,1\}^{\log D}$. We break the analysis into three cases.

- If $x_{\log D} = 0$ then $g(x,y) = 0$ because of the vector $x_{\log D}$ appearing in the first term of the product in the definition of g. Meanwhile, $h(x,y) = 1$ if and only if $y_{\log D} = 1$ and the first $\log D - 1$ bits of x and y match. In other words, if $x_{\log D} = 0$ then $g(x,y) + h(x,y)$ equals 1 if and only if y is the binary representation of $\mathsf{to\text{-}int}(x) + 1$.
- If $x_{\log D} = 1$, then let $k \geq 1$ be the smallest value such that $x_{\log D - k} = 0$ assuming such a k exists. Then $h(x,y) = 0$ because of the factor $(1 - x_{\log D})$ appearing in its definition. Moreover, in this case y is the binary representation of $\mathsf{to\text{-}int}(x) + 1$ if and only if $y_{\log D} = y_{\log D - 1} = \cdots = y_{\log D - (k-1)} = 0$, $y_{\log D - k} = 1$ and
$$(y_1, \ldots, y_{\log D - (k+1)}) = (x_1, \ldots, x_{\log D - (k+1)}).$$
For this y, $g(x,y) = 1$, because the k'th term of the sum in the definition of g evaluates to 1 while all other terms evaluate to 0. Meanwhile, for all other $y' \in \{0,1\}^{\log D}$, *all* terms of the sum defining g evaluate to 0.

  In other words, for all $(x,y) \in \{0,1\}^{\log D} \times \{0,1\}^{\log D}$ such that $x_{\log D} = 1$,

$$g(x,y) + h(x,y) = \begin{cases} 0 + 1 \text{ if } y \text{ is the binary representation of } \mathsf{to\text{-}int}(x) + 1 \\ 0 + 0 \text{ otherwise.} \end{cases}$$

- The final case is that $x_{\log D}$ is the all ones vector. In this case, it is easily checked that, regardless of the value of y, $h(x,y) = 0$ and every term in the sum defining $g(x,y)$ is also 0.

In each of the three cases above, we have shown that $h(x,y) + g(x,y)$ equals 1 if y is the binary representation of $\mathsf{to\text{-}int}(x) + 1$, and equals 0 otherwise. Hence, $h(x,y) + g(x,y) = \mathsf{next}(x,y)$.

We now explain that Expression (16) can be evaluated at any point $(r_x, r_y) \in \mathbb{F}^{\log D} \times \mathbb{F}^{\log D}$ in $O(\log D)$ time. It is easy to see that $h(r_x, r_y)$ can be evaluated in $O(\log D)$ time. It is also clear that each term of the sum defining g can be evaluated at (r_x, r_y) in $O(\log D)$ time. Since there are $O(\log D)$ terms of the sum, that immediately implies an upper bound of $O(\log D^2)$ time in total.

However, using the fact that adjacent terms of the sum involve almost identical factors, the runtime can be reduced to $O(\log m)$. For example, if $v(k)$ denotes the value of the k'th term of the sum, then

$$v(2) = v(1) \cdot x_{\log D - 1}(1 - y_{\log D - 1}) \left((1 - x_{\log D - 1}) y_{\log D - 1}\right)^{-1} (1 - x_{\log D - 2}) y_{\log D - 2} \widetilde{\mathsf{eq}}(x_{\log D - 2}, y_{\log D - 2})^{-1}.$$

This ensures that $v(i)$ can be computed in constant amortized time given $v(i-1)$ for each $i \in \{2, 3, \ldots, \log D\}$. (A batch inversion algorithm is used to compute all necessary field inversions across all terms of the sum with a single field inversion and $O(\log D)$ multiplications). This means $O(\log D)$ total total time suffices to evaluate all terms. □

5.2 CCS instances arising from AIR

Notation for this section. Throughout this section, m denotes the corresponding AIR parameter, so that the AIR witness vector w is in $\mathbb{F}^{(m-1)\cdot t/2}$ and the AIR vector z (Equation (6)) is in $\mathbb{F}^{(m+1)\cdot t/2}$. This means that m also denotes the number of rows in the CCS matrices $M_0, \ldots, M_{t-1}$ arising from the AIR $\rightarrow$ CCS transformation of Lemma 3. However, ws discussed shortly, in this section we assume $m - 1$ is a power of 2, and we will be padding $M_0, \ldots, M_{t-1}$ with all-zero-rows to ensure their number of rows is a power of 2 (this padding does not increase prover or verifier time in SuperSpartan). This means that the number of rows in the padded matrix will *not* be m, but rather $2(m-1)$.

Let $v = |w_{\text{AIR}}| = (m-1) \cdot t/2$. Recall that for CCS instances arising from Lemma 3 (after padding $(1, x)$ to have length v, so that $z = (w_{\text{AIR}}, 1, x)$ has length $2v = n$, see Remark 7), the length of the witness vector z is $n = (m-1) \cdot t$. Hence, we can think of z as itself a matrix, with $2(m-1)$ rows and $t/2$ columns. We consider $m - 1$ and t to be powers of 2 throughout this section. We also pad the matrices $M_0, \ldots, M_{t-1}$ with $m - 1$ all0-zeros rows to ensure that they have $2(m-1)$ rows, which is a power of 2. Hence, the multilinear extensions $\widetilde{M}_0, \ldots, \widetilde{M}_{t-1}$ are functions mapping the domain

$$\mathbb{F}^{1+\log(m-1)} \times \mathbb{F}^{\log(n)} = \mathbb{F}^{1+\log(m-1)} \times \mathbb{F}^{\log(m-1)+\log(t)-1} = \mathbb{F}^{\log(m-1)+\log(t)}$$

to $\mathbb{F}$.

Recall from the proof of Lemma 3 that for each $j \in \{0, 1, 2, \ldots, t-1\}$ and $i \in \{0, 1, \ldots, m-1\}$, the following holds:

- First, that
$$M_j[i][k] = 1 \text{ if } k = i \cdot t/2 + j, \tag{17}$$
unless $i = 0$ and $j < t/2$ or $i = m - 1$ and $j \geq t/2$. These last two cases are as follows:
- If $i = 0$ and $j < t/2$, then
$$M_j[i][j + |w_{\text{AIR}}|] = 1. \tag{18}$$
- If $i = m - 1$ and $j \geq t/2$, then
$$M_j[i][j + m \cdot t/2] = 1. \tag{19}$$

All entries of M_j not considered in any of the cases above are 0.

For $k \in \{0,1\}^{\log n}$, write $k = (k_0, k_1, \ldots, k_{\log(n)-1})$. We can break the bits of k into three pieces, say (k_r, k_h, k_c) with $k_r \in \{0,1\}^{\log(m-1)}$, $k_h \in \{0,1\}$, and $k_c \in \{0,1\}^{\log(t)-1}$. We think of k_h as specifying the

high-order bit of a"row" of z (essentially indicating whether or not the row is one of the first $m-1$ rows of z) and k_c as specifying the column of z (of which there are $t/2$).

Let $(j_0, \ldots, j_{\log(t)-1})$ denote the binary representation $\mathsf{bin}(j)$ of $j \in \{0, 1, \ldots, t-1\}$ with $j_{\log(t)-1}$ equal to the high-order bit. Let $j' = \mathsf{to\text{-}int}(j_0, \ldots, j_{\log(t)-2})$ denote the integer represented by j with its high-order bit removed.

A clean expression for $\widetilde{M}_j$. We claim that $\widetilde{M}_j(i, (k_r, k_h, k_c))$ equals the sum of the following four terms:

$$j_{\log(t)-1}(1-k_h) \cdot \widetilde{\mathsf{next}}(i, (k_r, 0))) \cdot \widetilde{\mathsf{eq}}((j_0, \ldots, j_{\log(t)-2}), k_c) \tag{20}$$

$$(1-j_{\log(t)-1})(1-k_h) \cdot \widetilde{\mathsf{eq}}(i, (k_r, 0)) \cdot \widetilde{\mathsf{eq}}((j_0, \ldots, j_{\log(t)-2}), k_c) \tag{21}$$

$$\begin{aligned}(1-j_{\log(t)-1}) \cdot k_h \cdot \widetilde{\mathsf{eq}}(i, \mathsf{bin}(m-1)) \cdot \widetilde{\mathsf{eq}}(k_r, \mathsf{bin}(m-1)) \cdot \widetilde{\mathsf{eq}}(j_0, \ldots, j_{\log(t)-2}, k_c) \\ +j_{\log(t)-1} \cdot k_h \cdot \widetilde{\mathsf{eq}}(i, \mathsf{bin}(m-1)) \cdot \widetilde{\mathsf{eq}}(k_r, \mathsf{bin}(1)) \cdot \widetilde{\mathsf{eq}}(j_0, \ldots, j_{\log(t)-2}, k_c)\end{aligned} \tag{22}$$

$$\begin{aligned}-(1-j_{\log(t)-1}) \cdot (1-k_h) \cdot \widetilde{\mathsf{eq}}(i, \mathsf{bin}(0)) \cdot \widetilde{\mathsf{eq}}(k_r, \mathsf{bin}(0)) \cdot \widetilde{\mathsf{eq}}((j_0, \ldots, j_{\log(t)-2}), k_c) \\ +(1-j_{\log(t)-1}) \cdot k_h \cdot \widetilde{\mathsf{eq}}(i, \mathsf{bin}(0)) \cdot \widetilde{\mathsf{eq}}(k_r, \mathsf{bin}(0)) \cdot \widetilde{\mathsf{eq}}((j_0, \ldots, j_{\log(t)-2}), k_c)\end{aligned} \tag{23}$$

In Equation (23) above, $\mathsf{bin}(0)$ refers to the all-zeros vector of length $1 + \log(m-1)$ or $\log(m-1)$ as appropriate.

Analyzing the sum of Expressions (20)-(23). To show that the sum of the above four expressions equals $\widetilde{M}_j(i, (k_r, k_h, k_c))$, we must establish that the sum of Expressions (20)-(23) is a multilinear polynomial in (i, k) that agrees with M_j at all inputs in

$$\{0,1\}^{\log(m-1)} \times \{0,1\} \times \{0,1\}^{\log n} \to \{0,1\}.$$

This will establish that it equals the unique multilinear extension of $M_j(i, k)$. Clearly, Expressions (20)-(23) are multilinear in the variables i and k, so we turn to showing that $M_j(i, k)$ equals the sum of the above four expressions when $(i, k) \in \{0,1\}^{\log m} \times \{0,1\}^{\log n}$.

For $i \in \{0,1\}^{\log(m-1)} \times \{0\}$, Expression (20) equals 1 if and only if the high-order bit of j is 1 (i.e., $j \geq t/2$), $\mathsf{to\text{-}int}(k_r) = \mathsf{to\text{-}int}(i) + 1$ and $j' = \mathsf{to\text{-}int}(k_c)$. This means that

$$\mathsf{to\text{-}int}(k_r, k_c) = (\mathsf{to\text{-}int}(i) + 1) \cdot t/2 + j' = i \cdot (t/2) + j.$$

This precisely matches Equation (17) in this case.

Meanwhile, Expression (20) equals 1 for $i \in \{0,1\}^{\log(m-1)} \times \{0\}$ if and only if the high-order bit of j is 0 (i.e., $j \leq t/2$), $k_r = i$ and $j' = \mathsf{to\text{-}int}(k_c)$. This means that

$$\mathsf{to\text{-}int}(k_r, k_c) = \mathsf{to\text{-}int}(i) \cdot t/2 + j' = \mathsf{to\text{-}int}(i) \cdot t/2 + j.$$

This precisely matches Equation (17), unless $i = \mathbf{0}$ and $j < t/2$, which is a special case.

Expressions (22) and (23) "correct" the behavior of (the sum of) Equations (20) and (21) when $\mathsf{to\text{-}int}(i) = m-1$ and when $\mathsf{to\text{-}int}(i) = 0$.

Specifically, Equation (22) maps $(i, k) \in \{0,1\}^{1+\log(m-1)} \times \{0,1\}^{\log n}$ to 1 in the following two cases (otherwise it maps (i, k) to 0):

- If $\mathsf{to\text{-}int}(i) = m-1$, $j < t/2$, and $\mathsf{to\text{-}int}(k) = (m-1)t/2 + j$.
- If $\mathsf{to\text{-}int}(i) = m-1$, $j \geq t/2$, and $\mathsf{to\text{-}int}(k) = mt/2 + j$.

This precisely matches the behavior of Equation (17) and (19) in these two cases.

Equation (23) maps $(i,k) \in \{0,1\}^{1+\log(m-1)} \times \{0,1\}^{\log n}$ to a non-zero value only in the following two cases:

- It maps (i,k) to -1 if $\mathsf{to\text{-}int}(i) = 0$, $j < t/2$, and $\mathsf{to\text{-}int}(k) = j$,
- It maps (i,k) to 1 if $\mathsf{to\text{-}int}(i) = 0$, $j < t/2$, and $\mathsf{to\text{-}int}(k) = (m-1)t/2 + j$.

The first case "cancels" the $+1$ evaluation that Expression (21) takes when $\mathsf{to\text{-}int}(i) = 0$, $j < t/2$ and $\mathsf{to\text{-}int}(k) = j$. The second case matches the behavior of Expression (18).

Evaluating Expressions (20)-(23) in logarithmic time. By Theorem 2 and Equation (9), the right hand side of Equations (20)-(23) can be computed in time $O(\log(n) + \log(m))$ for any $(i, (k_r, k_h, k_c)) \in \mathbb{F}^{\log(m)} \times \mathbb{F}^{\log n}$.

Remark 9 (Handling periodic constraints). We sketch how SuperSpartan can support periodic AIR constraints if desired. Suppose that an AIR constraint should be applied to rows i and $i+1$ only if i is an integer multiple of some designated power of 2, say 2^η. One can still apply the AIR $\rightarrow$ CCS transformation of Lemma 3. In the protocol for CCS of Figure 2, one can change Equation (15) to:

$$g(a) \coloneqq \widetilde{eq}(\tau, a) \cdot \left(\prod_{\ell=0}^{\eta-1} (1 - a_\ell) \right) \sum_{i=0}^{q-1} \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log n}} \widetilde{M_j}(a, y) \cdot \widetilde{Z}(y) \right) \tag{24}$$

Here, we have inserted the product $\prod_{\ell=0}^{\eta-1}(1-a_\ell)$, which ensures that $g(a) = 0$ whenever a is the binary representation of an integer that is *not* a multiple of 2^η. The insertion of this product increases the proof size and verifier time by at most η field elements and $O(\eta)$ field operations.

Recall that the AIR $\rightarrow$ CCS transformation of Lemma 3 assumed there was only one AIR constraint, but that any number of AIR constraints can be handled via the standard "random linear combination of constraints" technique captured in Equation (8), which reduces k constraints $g_1, \ldots, g_k$ to the following single constraint:

$$\sum_{i=0}^{k-1} r^i \cdot g_i.$$

If one or more AIR constraints are periodic, then we think of each AIR constraint g_i as taking as input $\log m$ extra variables $(a_1, \ldots, a_{\log m})$ corresponding to the "low-order" η bits of the "CCS row index", and we replace any periodic AIR constraint g_i with period 2^η with

$$\left(\prod_{\ell=0}^{\eta-1} (1 - a_\ell) \right) \cdot g_i.$$

The method described in Equation (24) above corresponds to this technique when there is just a single AIR constraint (so no random linear combination of constraints is taken). An equivalent way to describe the technique is to imagine that we apply the protocol of Figure 2 separately to each constraint $g_1, \ldots, g_k$ (with Equation (15) changed to (24) for any constraint with period η) but rather than applying the sum-check protocol k separate times to k different polynomials, we apply the sum-check protocol just once, to a random linear combination of the polynomials.

5.3 Avoiding preprocessing for SIMD CCS

Tzialla et al. [TKPS22] describe a natural extension of R1CS where the same circuit is applied over β different witnesses and public inputs. SIMD R1CS enforces input/output consistency checks across neighboring instances (e.g., the input to instance i must be the output of instance $i-1$). The computational model is therefore closely analogous to AIR, except that each sequential step is specified with R1CS. Note that in the absence of such input/output consistency checks are omitted, SIMD R1CS captures data-parallel circuits [Tha13, WJB+17, WTS+18]. Additionally, Tzialla et al. [TKPS22] describe an extension of Spartan [Set20], called Phalanx, to prove SIMD R1CS.

Costs of Phalanx. For β copies of R1CS with $m \times n$ matrices with at most N non-zero entries and ℓ public inputs and outputs, the verifier's work is $O(N + \log \beta)$ field operations plus the time to verify an evaluation proof for a $\log(\beta \cdot (n - \ell - 1))$-variate multilinear polynomial encoding β witnesses. If preprocessing is allowed, which incurs $O(N)$ field and cryptographic operations, the verifier's work drops to the cost of verifying polynomial evaluation proofs for a constant number of $O(\log N)$-variate committed multilinear polynomials.

For proving input/output consistency checks, they describe an approach that either requires $O(\beta \cdot \ell)$ field operations for the verifier, or the verifier incurs the cost of verifying a polynomial evaluation proof for a committed $O(\log(\beta \cdot \ell))$-variate multilinear polynomial (the latter option requires a preprocessing work of $O(\beta \cdot \ell)$ field and cryptographic operations).

SIMD CCS and SuperSpartan for SIMD CCS. There is a natural definition of SIMD CCS analogous to SIMD R1CS. Furthermore, SuperSpartan can be extended to prove those instances. In addition, to prove the input/output consistency checks, by using the fact that $\widetilde{\mathsf{next}}$ can be evaluated in logarithmic time (Theorem 2), the verifier's work is $O(\log(\beta \cdot \ell))$ without requiring any preprocessing—an exponential improvement over Phalanx's cost to verify input/output consistency without any preprocessing. We leave a full description of these details to the near-term future work.

6 Compiling the polynomial IOP for CCS to SNARKs for CCS

This section compiles our polynomial IOP for CCS into a SNARK using various polynomial commitment schemes for multilinear polynomials. Unless the CCS instance is structured in the sense of Section 5, this requires a polynomial commitment scheme that can efficiently handle sparse multilinear polynomials $\widetilde{M}_i$, meaning that the time to commit to the polynomial and compute an evaluation proof is proportional to the number of inputs $x \in \{0,1\}^\ell$ for the appropriate value of ℓ, such that $\widetilde{M}_i(x) \neq 0$. Here, we use the compiler from Spartan [Set20], which transforms any of the multilinear polynomial commitment schemes listed in Figure 3.1 into ones that meet this requirement.

The following theorem captures our result.

Theorem 3. *Assuming that $|\mathbb{F}| = 2^{\Theta(\lambda)}$ there exists a family of preprocessing SNARKs for CCS with efficiency characteristics depicted in Figures 3 and 4.*

Proof. From applying [BFS20, Theorem 8] to the polynomial IOP for CCS in Theorem 1 using a sparse polynomial commitment obtained by applying [GLS+21, Theorem 3] to polynomial commitment schemes listed in Figure 3.1, there exists a public-coin interactive argument for $\mathcal{R}_{\text{R1CS}}$ with witness-extended emulation. Applying the Fiat-Shamir transform [FS86] to the public-coin interactive argument results in the claimed SNARKs for CCS.

If the CCS instance is not uniform (Section 5) the verifier[10], in a preprocessing step, commits to t

[10]The party that commits to $\widetilde{M}_0, \ldots, \widetilde{M}_t$ in pre-processing is often referred to as the *indexer* in the literature. The important point is that these commitments must be computed honestly for the resulting pre-processing SNARK to be sound. The pre-processing commitment procedure is transparent if the polynomial commitment scheme used is transparent.

commitment scheme	preprocessing time	prover time	proof size	verifier time
Brakedown-PC	$O(N)$ $\mathbb{F}$ $O(N)$ $\mathbb{H}$	$O(N)$ $\mathbb{F}$ $O(N)$ $\mathbb{H}$	$O(t\lambda)$ $\mathbb{H}$ $O(\sqrt{N \cdot \lambda} + d \log m)$ $\mathbb{F}$	$O(t\sqrt{N \cdot \lambda})$ $\mathbb{H}$ $O(dq + d \log m)$ $\mathbb{F}$
Orion-PC	$O(N)$ $\mathbb{F}$ $O(N)$ $\mathbb{H}$	$O(N)$ $\mathbb{F}$ $O(N)$ $\mathbb{H}$	$O(t\lambda \log^2 N)$ $\mathbb{H}$ $O(d \log m)$ $\mathbb{F}$	$O(t\lambda \log^2 N)$ $\mathbb{H}$ $O(dq + d \log m)$ $\mathbb{F}$
Dory-PC	$O(N)$ $\mathbb{G}_1$	$O(N)$ $\mathbb{G}_1$	$O(\log N)$ $\mathbb{G}_T$ $O(d \log m)$ $\mathbb{F}$	$O(\log N)$ $\mathbb{G}_T$ $O(dq + d \log m)$ $\mathbb{F}$
KZG + Gemini	$O(N)$ $\mathbb{G}_1$	$O(N)$ $\mathbb{G}_1$	$O(\log N)$ $\mathbb{G}_1$ $O(d \log m)$ $\mathbb{F}$	$O(\log N)$ $\mathbb{G}_1$ $O(dq + d \log m)$ $\mathbb{F}$

Figure 3: SuperSpartan family of SNARKs for *non-uniform* instances of CCS. We obtain a separate SNARK for each choice of the polynomial commitment scheme. The prover time listed is in addition to the $O(N + tm + qmd \log^2 d)$ field operations required to implement the two invocations of the sum-check protocol in SuperSpartan. The KZG+Gemini commitment scheme also involves a trusted setup to produce an SRS of size $O(N)$. Recall that N denotes the number of non-zero entries across all CCS matrices $M_0, \ldots, M_{t-1}$, m denotes the number of rows and n the number of columns in each matrix, q denotes the number of multi-sets $S_0, \ldots, S_{q-1}$, and d denotes the maximum size of any multiset S_i.

$(\log(m) + \log(n))$-variate polynomials $\widetilde{M}_0, \ldots, \widetilde{M}_{t-1}$ that evaluate to a non-zero value at at most N locations over the Boolean hypercube $\{0,1\}^{\log m + \log(n)}$.

The prover: (1) commits to a $O(\log n)$-variate polynomial $\widetilde{Z}$; (2) participates in the sum-check protocol in the polynomial IOP in Theorem 1 (which costs $O(N + d \cdot t \cdot m + qtd \log^2 d)$ $\mathbb{F}$-ops); (3) proves evaluations of the $(\log n)$-variate multilinear polynomial $\widetilde{Z}$; and (4) If the CCS instance is not uniform (Section 5) proves one evaluation each of the t different $(\log m + \log n)$-variate multilinear polynomials $\widetilde{M}_0, \ldots, \widetilde{M}_{t-1}$ committed during the preprocessing step (each of these polynomials is evaluated at the same point $(r_a, r_y) \in \mathbb{F}^{\log m} \times \mathbb{F}^{\log n}$ (see Figure 2). Note that many polynomial commitment schemes have effective batching procedures that enable these t evaluations to be proven and verified with the same cost as a single evaluation to a single committed polynomial.

The verifier to verify a proof: (1) participates in the sum-check protocol in the polynomial IOP in Theorem 1 (which costs $O(d \cdot q + d \log m + \log n)$ $\mathbb{F}$-ops); (2) verifies the proofs of evaluations of one $(\log(n))$-variate committed multilinear polynomial; (3) If the CCS instance is uniform in the sense of Section 5, the verifier evaluates $\widetilde{M}_0, \ldots, \widetilde{M}_{t-1}$ on its on, each at a random point (r_a, r_y), which by Section 5 can be done in $O(t \cdot (\log n + \log m))$ time. If the CCS instance is non-uniform, then the verifier checks the evaluation proofs provided by the prover for each of $\widetilde{M}_0(r_a, r_y), \ldots, \widetilde{M}_{t-1}(r_a, r_y)$ provided by the prover. Again, many polynomial commitment schemes have effective batching procedures that enable these t evaluations to be verified with the same cost as a single evaluation to a single committed polynomial.

Finally, the proof size is the sum of the proof sizes from the sum-check protocol in the polynomial IOP from Theorem 1 and the evaluation proof sizes from polynomial commitment schemes. □

7 Extensions and conclusions

We note that SNARKs based on polynomial IOPs and polynomial commitments can be easily extended to handle customizable constraints, where gates compute high-degree multivariate polynomials. (Unfortunately, it is not clear how to extend linear-PCP-based SNARKs [GGPR13, Gro16] to support higher degree constraints due to the way they use bilinear pairings, which restrict the verifier's check to be a degree-2 polynomial evaluation.) Furthermore, polynomial IOPs that already handle R1CS can easily be extended to support CCS: Appendix A describes how to extend Marlin [CHM+20] to obtain SuperMarlin.

However, the resulting SNARKs (including Plonk applied to Plonkish) have cryptographic costs that grow with the size d of the CCS multi-sets $S_0, \ldots, S_{t-1}$ (in the language of Plonkish, with the degree d of the polynomial constraints). This is because SNARKs such as Marlin and Plonk have the prover commit to a "quotient polynomial" whose degree grows with d, making it unattractive to use Plonk or SuperMarlin to prove constraint systems with larger values of d (e.g., $d > 5$). Furthermore, the cryptographic costs for the

commitment scheme	preprocessing time (circuit-independent)	prover time	proof size	verifier time
Brakedown-PC		$O(n)$ $\mathbb{F}$ $O(n)$ $\mathbb{H}$	$O(\lambda)$ $\mathbb{H}$ $O(\sqrt{n \cdot \lambda} + d \log m)$ $\mathbb{F}$	$O(\sqrt{n \cdot \lambda})$ $\mathbb{H}$ $O(dq + d \log m)$ $\mathbb{F}$
Orion-PC	$O(\sqrt{n})$ $\mathbb{F}$ $O(\sqrt{n})$ $\mathbb{H}$	$O(n)$ $\mathbb{F}$	$O(\lambda \log^2 n)$ $\mathbb{H}$ $O(d \log m)$ $\mathbb{F}$	$O(\lambda \log^2 n)$ $\mathbb{H}$ $O(dq + d \log m)$ $\mathbb{F}$
Dory-PC	$O(\sqrt{n})$ $\mathbb{G}_1$	$O(n)$ $\mathbb{G}_1$	$O(\log n)$ $\mathbb{G}_T$ $O(d \log m)$ $\mathbb{F}$	$O(\log n)$ $\mathbb{G}_T$ $O(dq + d \log m)$ $\mathbb{F}$
KZG + Gemini		$O(n)$ $\mathbb{G}_1$	$O(\log n)$ $\mathbb{G}_1$ $O(d \log m)$ $\mathbb{F}$	$O(\log n)$ $\mathbb{G}_1$ $O(dq + d \log m)$ $\mathbb{F}$

Figure 4: SuperSpartan family of SNARKs for *uniform* instances of CCS (e.g., those arising from the AIR $\rightarrow$ CCS transformation of Lemma 3). We obtain a separate SNARK, one for each choice of the polynomial commitment scheme. The prover time listed is in addition to the $O(N + tm + qmd \log^2 d)$ field operations required to implement the two invocations of the sum-check protocol in SuperSpartan. The KZG+Gemini commitment scheme also involves a trusted setup to produce an SRS of size $O(n)$. Recall that N denotes the number of non-zero entries across all CCS matrices $M_0, \ldots, M_{t-1}$, m denotes the number of rows and n the number of columns in each matrix, q denotes the number of multi-sets $S_0, \ldots, S_{q-1}$. and d denotes the maximum size of any multiset S_i.

prover (and for pre-processing) in these SNARKs also grows with the number t of CCS constraint matrices $M_0, \ldots, M_{t-1}$ and the number of rows in these matrices.

The above prover cost profiles are in contrast to SuperSpartan (and Hyperplonk), where only the prover's finite field operations, and not the cryptographic operations, grows with the degree d. Moreover, in the case of uniform CCS instances, SuperSpartan's prover's cryptographic work does not grow with the number of constraint matrices t or the number of rows m, only with the number of columns n.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[BBHR19a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2019.

[BBHR19b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732. Springer, 2019.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, 2012.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orru. Gemini: Elastic snarks for diverse environments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2022.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019.

[bel] bellman. `https://crates.io/crates/bellman`.

[BFL92] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. 2(4), December 1992.

[BFR+13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2013.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[BGtRZt23] Jeremy Bruestle, Paul Gafni, and the RISC Zero team. RISC Zero zkVM: Scalable, Transparent Arguments of RISC-V. 2023. Available at `https://www.risczero.com/proof-system-in-detail.pdf`.

[BSCKL23] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves: (ECFFT Part II). In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I*, pages 467–496. Springer, 2023.

[BSS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, May 2008.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[CTY12] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proceedings of the VLDB Endowment*, 5(1):25, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 186–194, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum snarks for r1cs, 2021.

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. *Cryptology ePrint Archive*, 2021.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 99–108, 2011.

[GW20] Ariel Gabizon and Zachary J Williamson. plookup: A simplified polynomial protocol for lookup tables. 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953, 2019.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 536–547, 2008.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2022.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 177–194, 2010.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, October 1990.

[LNS20] Jonathan Lee, Kirill Nikitin, and Srinath Setty. Replicated state machines without replicated execution. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[Mic94] Silvio Micali. CS proofs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

[OWB20] Alex Ozdemir, Riad S. Wahby, and Dan Boneh. Scaling verifiable computation using efficient set accumulators, 2020.

[PGHR13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2013.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference (TCC)*, 2013.

[SAGL18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2018.

[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 ACM European Conference on Computer Systems (EuroSys)*, April 2013.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.

[Sta21] StarkWare. ethSTARK Documentation. Cryptology ePrint Archive, Paper 2021/582, 2021. `https://eprint.iacr.org/2021/582`.

[SVP+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the USENIX Security Symposium*, August 2012.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2013.

[Tha20] Justin Thaler. Proofs, arguments, and zero-knowledge. `http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html`, 2020.

[TKPS22] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2022.

[WJB+17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[WSR+15] Riad S. Wahby, Srinath Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2015.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2022.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

A SuperMarlin

In this section, we sketch a variant of Marlin [CHM+20] that applies to CCS. Our presentation borrows substantially from the presentation of Marlin for R1CS in [Tha20, Chapter 10].

For simplicity, we assume that the number of rows m and columns n in the CCS matrices $M_0, \ldots, M_{t-1}$ are equal. We also assume that there is a multiplicative subgroup H of $\mathbb{F}$ of size exactly n (say, H is the set of all n'th roots of unity, so that the so-called *vanishing polynomial* for H is $\mathbb{Z}_H(x) = x^n - 1$. This polynomial maps all entries of H to 0 and no other entries of $\mathbb{F}$ to 0.

For $M_0, \ldots, M_t$ let z_{M_i} denote $M_i \cdot z$, which is a vector in $\mathbb{F}^n$, and let $\hat{z}_{M_i}$ be the associated univariate low-degree extension polynomial. That is, indexing the n entries of $\hat{z}_{M_i}$ by the n elements of H using any natural bijection, $\hat{z}_{M_i}$ is the unique polynomial of degree at most $n-1$ such that $\hat{z}_{M_i}(h) = (z_{M_i})_h$ for all $h \in H$.

We describe the protocol as a polynomial IOP, whereby each message from prover to verifier specifies a univariate polynomial of degree at most $O(n)$. In the associated SNARK, the prover will *commit* to each such polynomial with a polynomial commitment scheme, and each time the verifier needs to evaluate a committed polynomial at a point, the prover will provide the evaluation along with a proof that the returned evaluation is consistent with the committed polynomial.

The polynomial IOP. To check that indeed

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = \mathbf{0}, \tag{25}$$

the verifier must confirm two properties. First:

$$\text{for all } h \in H, \sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} \hat{z}_{M_i}(h) = \mathbf{0}. \tag{26}$$

Second:

$$\text{for all } h \in H, \text{ and } i = 0, \ldots, t-1, \ \hat{z}_{M_i}(h) = \sum_{j \in H} M_{h,j} \cdot \hat{z}(j). \tag{27}$$

Equation (27) ensures that for each $i = 0, \ldots, t-1$, $z_{M_i} = M_i \cdot z$. Assuming this to be so, Equation (26) confirms that Equation (25) holds.

The prover sends t degree-n polynomials $\hat{z}$, $\hat{z}_{M_0}, \ldots, \hat{z}_{M_{t-1}}$, the (univariate) low-degree extensions of z and $z_{M_0}, \ldots, z_{M_{t-1}}$.

Checking Equation (26)**.** A standard fact (see [BSS08], or [Tha20, Lemma 9.3] for an exposition) implies that the first check is equivalent to the existence of a polynomial h^* of degree at most $d \cdot n$ such that

$$\sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} \hat{z}_{M_i}(X) = h^*(X) \cdot \mathbb{Z}_H(X). \tag{28}$$

The prover sends the quotient polynomial h^*. The verifier probabilistically checks that Equation (28) holds by choosing a random $r \in \mathbb{F}$ and confirming that

$$\sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} \hat{z}_{M_i}(r) = h^*(r) \cdot \mathbb{Z}_H(r). \tag{29}$$

This requires querying the committed polynomials $\hat{z}_{M_0}, \ldots, \hat{z}_{M_{t-1}}$ and h^* at r; the verifier can evaluate $\mathbb{Z}_H(r)$ on its own in logarithmic time because $\mathbb{Z}_H(r)$ is sparse. Since all of the messages sent by the prover are polynomials of degree at most $n-1$, up to soundness error $2n/|\mathbb{F}|$ over the choice of r, if Equation (29) holds at r then it is safe for the verifier to believe that Equation (28) holds, and hence also Equation (26).

Checking Equation (27)**.** The check that Expression (27) holds leverages interaction. Fix M_i for some $i = 0, \ldots, t = 1$ for the remainder of the paragraph. Let $\hat{M}_i(X, Y)$ denote the bivariate low-degree extension of the matrix M, interpreted in the natural manner as a function $M_i(x, y) \colon H \times H \to \mathbb{F}$ via $M_i(x, y) = M_{x,y}$. That is, $\hat{M}_i(x, y)$ is the unique bivariate polynomial of degree at most n in each variable that extends M_i. Since $\hat{z}_{M_i}$ is the *unique* extension of z_{M_i} of degree less than n, it is easily seen that Equation (27) holds for all $h \in H$ if and only if the following equality holds as formal polynomials:

$$\hat{z}_{M_i}(X) = \sum_{j \in H} \hat{M}_i(X, j)\hat{z}(j). \tag{30}$$

Since any two distinct polynomials of degree at most n can agree on at most n inputs, if the verifier chooses r' at random from $\mathbb{F}$, then up to soundness error $n/|\mathbb{F}|$ over the choice of r', Equation (27) holds if and only if

$$\hat{z}_{M_i}(r') = \sum_{j \in H} \hat{M}(r', j)\hat{z}(j). \tag{31}$$

The verifier checks Equation (31) by sending r' to the prover and proceeding as follows. Let

$$q(Y) = \hat{M}_i(r', Y)\hat{z}(Y) - \hat{z}_{M_i}(r') \cdot |H|^{-1},$$

so that the validity of Equation (31) is equivalent to $\sum_{j \in H} q(Y) = 0$. The verifier requests that the prover establish that $\sum_{j \in H} q(Y) = 0$ by applying the so-called univariate sum-check protocol [BCR+19] (see [Tha20, Chapter 10] for an exposition).

At the end of the univariate sum-check protocol applied to q, the verifier needs to evaluate q at a randomly chosen point r''. Clearly this can be done in a constant number of field operations if the verifier is given $\hat{z}(r'')$, $\hat{z}_{M_i}(r')$, and $\hat{M}_i(r', r'')$. The first two evaluations, $\hat{z}(r'')$ and $\hat{z}_{M_i}(r')$, can be obtained with one query each to the polynomials $\hat{z}$ and $\hat{z}_{M_i}$.

Our description above omits two details: how the univariate sum-check protocol works, and how the bivariate polynomials $\hat{M}_i(X, Y)$ can be committed by an honest party in pre-processing, and evaluation proofs computed, in time proportional to the number of non-zero entries rather than the total number of entries (i.e., a sparse polynomial commitment scheme for bivariate polynomials under the assumption that the commitment is computed honestly). We direct the interested reader to Marlin [CHM+20] or [Tha20, Chapter 10] for details of both. Various concrete optimizations are also possible, using, e.g., efficient batching properties of evaluation queries to polynomials committed with homomorphic polynomial commitment schemes such as KZG commitments [KZG10].

B CCS+: CCS with lookup operations

This section describes an extension to CCS that additionally supports "lookup gates". In a nutshell, we allow a lookup table (a set of values from $\mathbb{F}$) to specified as part of the CCS+ structure and a set lookup operations (indexes into the satisfying assignment z), with the constraint that the values in the satisfying assignment at those specified locations are contained in the lookup table.

As a concrete example, suppose that the lookup table is $T = \{0, 1, \ldots, 2^{32} - 1\}$, and suppose that certain variables in the constraint system must be range constrained to be in the range $[0, 2^{32} - 1]$. The lookup operations in CCS+ can be used to specify which variables must be contained table and thereby enforce the desired range constraints.

Remark 10. In the definition below, for simplicity, we allow a single table, but it easily extends to the case of multiple tables. In particular, the lookup operation now becomes a tuple where the first entry in the tuple specifies the table and the second entry specifies the index into the satisfying assignment.

Definition B.1 (CCS+)**.** A CCS+ structure $\mathcal{S}$ consists of:

- size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$;
- a sequence of matrices $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times n}$ with at most $N = \Omega(\max(m, n))$ non-zero entries in total;
- a sequence of q multisets $[S_0, \ldots, S_{q-1}]$, where an element in each multiset is from the domain

$$\{0, \ldots, t-1\}$$

  and the cardinality of each multiset is at most d.
- a sequence of q constants $[c_0, \ldots, c_{q-1}]$, where each constant is from $\mathbb{F}$.
- a lookup table T, where T is a set of values from $\mathbb{F}$.
- a sequence of lookup operations L, where each entry in L is in the range $[n]$.

A CCS+ instance consists of public input $x \in \mathbb{F}^\ell$. A CCS+ witness consists of a vector $w \in \mathbb{F}^{n-\ell-1}$. A CCS+ structure-instance tuple $(\mathcal{S}, x)$ is satisfied by a CCS+ witness w if, for each lookup operation o in L, $z[o] \in T$, and moreover

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = \mathbf{0}.$$

Here, $z = (w, 1, x) \in \mathbb{F}^n$, $M_j \cdot z$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard product between vectors, and $\mathbf{0}$ is an m-sized vector with entries equal to the the additive identity in $\mathbb{F}$.

C SuperSpartan+: Extending SuperSpartan to handle CCS+

This section sketches how to extend SuperSpartan to handle CCS+. We leave it to the near-term future work to formalize this construction and provide a security analysis.

Suppose there is a polynomial IOP for lookup operations, which means the following. The verifier has oracle access to two polynomials: (1) a polynomial $\widetilde{T}$ that encodes the table T; and (2) a polynomial that encodes values claimed to be in the table $\widetilde{a}$. The polynomial IOP for lookup operations convinces the verifier that values in the second polynomial are contained in the table encoded by the first polynomial. There are many examples of such polynomial IOPs in the literature (e.g., polynomial IOPs that underlie lookup arguments such as plookup [GW20]). Furthermore, SuperSpartan's polynomial IOP for CCS (or any polynomial IOP for R1CS) can also provide a polynomial IOP for lookup operations: create a CCS structure-instance takes as non-deterministic input the two polynomials noted above and checks the desired lookup relation on the provided inputs (e.g., with randomized fingerprinting techniques).

Given a polynomial IOP for lookup operations, we can then easily extend the SuperSpartan's polynomial IOP for CCS to handle CCS+.

Given lookup operations L in the CCS+ structure, there exists a sparse matrix $M_L \in \mathbb{F}^{|L| \times n}$, where $\forall i \in [|L|]$, $M_L[i][L[i]] = 1$ (all other entries of M_L are set to 0). Suppose that the verifier has oracle access to $\widetilde{M_L}$. In the context of a SNARK, $\widetilde{M_L}$ is committed in the preprocessing phase in addition to other sparse matrices in the CCS structure. If $\widetilde{M_L}$ is structured, the commitment can be omitted, as the verifier will be able, on its own, to evaluate $\widetilde{M_L}$ at any desired point in logarithmic time.

We now discuss modifications to SuperSpartan's polynomial IOP. In the first message sent by the prover in SuperSpartan's polynomial IOP, in addition to the witness polynomial $\widetilde{w}$, the prover sends a polynomial $\widetilde{a}$ claimed to be the MLE of the vector $M_L \cdot z$, where $z = (w, 1, x)$. The prover and the verifier use an invocation of the sum-check protocol to check that the claimed relationship $M_L \cdot z = a$ indeed holds. SuperSpartan+'s polynomial IOP then invokes the polynomial IOP for lookup operations to confirm that entries in a are contained in the table T. Completeness and soundness follow from the completeness and soundness of the sum-check protocol and of the polynomial IOP for lookup operations.