NeutronNova: Folding everything that reduces to zero-check

Abhiram Kothapalli
University of California, Berkeley

Srinath Setty
Microsoft Research

Abstract.

We introduce NeutronNova, a new folding scheme for the *zero-check* relation: an instance-witness pair is in the zero-check relation if a corresponding multivariate polynomial evaluates to zero for all inputs over a suitable Boolean hypercube. The folding scheme is a two-round protocol, and it internally invokes a *single* round of the sum-check protocol. The folding scheme is more efficient than prior state-of-the-art schemes and directly benefits from recent improvements to the sum-check prover. The prover's work is the cost to commit to a witness and field operations in a single round of the sum-check protocol. So, if the witness contains "small" field elements, the prover only commits to "small" field elements. The verifier's work is a constant number of group scalar multiplications, field operations, and hash computations. Moreover, the folding scheme generalizes to fold multiple instances at once and requires only $\log n$ rounds of the sum-check protocol, where n is the number of instances folded.

As a corollary, we provide a folding scheme for any relation $\mathcal{R}$ for which there is a reduction of knowledge (RoK) from $\mathcal{R}$ to one or more instance-witness pairs in the zero-check relation. Such RoKs appear implicitly in prior lookup arguments (e.g., Lasso) and high-performance zkVMs for RISC-V (e.g., Jolt). We formalize these RoKs for several relations including indexed lookups, grand products, and CCS (which generalizes Plonkish, AIR, and R1CS). These are simple and constant round RoKs that leverage interaction to perform randomized checks on committed witness elements. Instead of *proving* these resulting zero-check instances as is done in prior proof systems such as Jolt, NeutronNova provides the more efficient option of continual folding of zero-check instances into a single running instance.

1 Introduction

A folding scheme [KST22] is a simple cryptographic protocol between a *prover* and a *verifier* that reduces the task of checking two NP instances into the task of checking a single instance. For example, suppose that a verifier holds two public inputs x_1 and x_2 and suppose that the prover wishes to prove that it knows two witnesses w_1 and w_2 such that they both satisfy a circuit C, that is, $C(w_1, x_1) = 1$ and $C(w_2, x_2) = 1$. Instead of proving both claims, the prover and the verifier can first invoke a folding scheme for circuit satisfiability to reduce both claims into a single claim of the same size about some witness w_3 and some public input x_3. Furthermore, Kothapalli et al. [KST22,KS24] show that folding schemes when used in a recursive manner provides incrementally verifiable computation (IVC) [Val08], a powerful cryptographic primitive that

allows producing a proof of correct execution of a "long running" computation in an incremental fashion—without having to prove the entire computation at once. Specifically, the prover to takes as input a proof π_i proving the correct execution of the first i steps of a computation, and update it to produce a proof π_{i+1} proving the first $i+1$ steps of the computation. Notably, the prover's per-step work to update a proof and the verifier's work to verify a proof are both independent of the number of steps executed.[1]

Prior to folding schemes, IVC was constructed using succinct non-interactive arguments of knowledge (SNARKs) [Val08,BCTV14], non-interactive arguments of knowledge (NARKs) with accumulation schemes [BGH19,BCMS20], or NARKs with split-accumulation schemes [BCL+21]. Folding schemes avoid SNARKs and NARKs entirely and merely employ a particular type of reduction of knowledge [KP23]. Furthermore, folding schemes not only provide a clean abstraction that is not tied to SNARKs or NARKs, they also provide a significantly more efficient prover than their predecessors. The efficiency stems from not having to produce a SNARK (or a NARK), but rather directly fold instances in some relation. In state-of-the-art folding schemes [KS24,BC23,EG23], the prover merely commits to its witness and performs some finite field operations.

Motivating application: zkVMs. zkVMs refer to succinct proof systems for machine executions: given an assembly program designed to run on a machine (e.g., RISC-V), the prover proves the correct execution of the program. zkVMs are attractive in practice because one does not need to express their computation with circuits. Rather, a programmer expresses their desired computation in a high-level language (e.g., Rust) and compiles it to an assembly program using existing compiler toolchains. Furthermore, zkVMs can, by design, support proving program executions with arbitrary control flow.

Technical challenges: time efficiency + space efficiency. A key challenge is to ensure that the zkVM prover runs efficiently, and, in particular, ensure that the prover's space complexity is independent of the length of the execution proven. As demonstrated by Ben-Sasson et al. [BCTV14] this can be achieved by utilizing (SNARK-based) IVC, which allows one to prove the correct execution of the supported machine cycle-by-cycle (or a fixed number of cycles at once). This ensures that the prover's space requirements are essentially the space requirements of the program whose execution is being proven.

Given that folding schemes offer a more efficient route to constructing IVC, at first blush, folding schemes appear to be a perfect fit for constructing time-efficient and space-efficient zkVMs. Indeed, this is the approach taken by the Nexus project [nex24b], which effectively replaces SNARKs with folding schemes in the

[1] A folding-based IVC can also be built in a tree fashion (e.g., [ZV23,NDC+24]): the prover could first produce proofs $\pi_{i\to j}$ and $\pi_{j\to k}$ proving the correct execution of the computation from steps i to j and j to k respectively. The prover can then combine them to produce a proof $\pi_{i\to k}$ that proves the correct execution of the computation from steps i to k. Crucially, the cost to produce the combined proof does not depend on the number of steps proven in the incoming proofs.

zkVM due to [BCTV14] and builds a zkVM for RISC-V. While this certainly achieves better efficiency than [BCTV14], Nexus's prover is still 3+ orders of magnitude slower than state-of-the-art zkVMs such as Jolt [AST24].[2]

Focusing on time efficiency, Jolt achieves a significant speedup through a carefully-designed set of reductions from the correctness of a RISC-V execution to a simpler set of problems in NP (a combination of lookups, grand products, and R1CS), which are then proven with a proof system derived from Spartan [Set20]. In contrast, Nexus encodes the entire RISC-V execution with a universal circuit expressed with R1CS. This means that even though folding ensures that the prover's space requirements do not exceed the size of (a circuit representation of) a single cycle (or a pre-defined number of cycles) and the per-constraint prover costs are low, the circuit encoding each cycle is substantial: 15,000 R1CS constraints [nex24a]. Specifically, the universal circuit pays for every instruction supported by the machine. Furthermore, it uses Merkle proofs to verify memory operations, which is up to three orders of magnitude less efficient than Jolt's approach, which is a collection of techniques referred to as offline memory checking [BEG$^+$91,CDD$^+$03,SAGL18].

Focusing on space efficiency, Jolt is a monolithic zkVM that can only execute a limited number of CPU cycles, where the limit is given by the amount of space available to the prover. In contrast, Nexus is a space-efficient zkVM that scales to any number of CPU cycles, thanks to the use of folding schemes.

Research questions. The above motivates the following question: *Can we design folding schemes that enable space-efficient zkVMs while still leveraging reductions from universal machine execution into more efficient problems?*

To answer this, we start with Jolt's design paradigm of using a carefully-curated set of relations that imply the correctness of universal machine execution as opposed to directly proving machine execution via constraints. We bring this approach to the folding-based zkVM setting by designing a suite of optimized folding schemes for such relations, which can then be used instead of directly folding machine execution constraints. As such, the folding schemes that we provide serve as a starting point toward the goal of building a zkVM that is simultaneously space and time efficient.

1.1 A technical overview our solution: NeutronNova

We introduce NeutronNova, a suite of folding schemes for relations that arise in the context of proving the correct execution of CPUs such as RISC-V.

We design NeutronNova in a modular fashion to enable a rapid development of folding schemes for new relations that may arise in the future. To enable such extensibility, we design a folding scheme for a highly-expressive core-relation, zero-check, to which we can naturally reduce a variety of complex relations such

[2] When we say Nexus, we refer to their folding-based zkVM. More recently, they have integrated Jolt, but that version suffers from the same space limitations as Jolt.

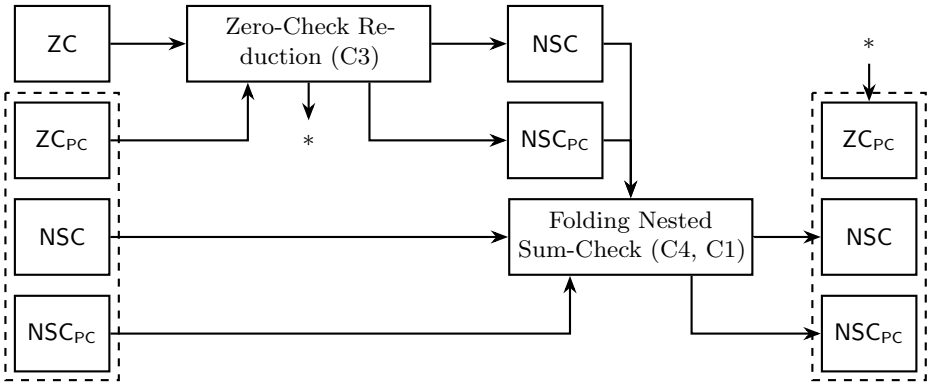

Fig. 1. An overview of ZeroFold, our folding scheme for the zero-check relation underlying NeutronNova. An instance in the zero-check relation, ZC, is folded into a running instance consisting of instances in the power-check relation, $\mathsf{ZC}_{\mathsf{PC}}$, that checks that a vector commits to the powers of a random challenge, a variant of the sum-check relation NSC, and a variant of the sum-check relation that enforces the power-check relation $\mathsf{NSC}_{\mathsf{PC}}$. In the diagram, $*$ indicates that one of the outputs of the zero-check reduction is a fresh $\mathsf{ZC}_{\mathsf{PC}}$ instance included in the output running instance.

as CCS [STW23], lookups, and grand-products. We believe that this modular framework also eases the formal verification of a zkVM based on NeutronNova.

In more detail, an instance-witness pair is in the zero-check relation if a prescribed multivariate polynomial, with coefficients that are a function of the instance and witness, evaluates to zero for all values over a suitable hypercube. As shown implicitly in prior work, a vast number of complex relations can be *interactively* reduced to the zero-check relation, often with minimal computation and communication [BFLS91,BFL92,Sha92,BTVW14,Set20,GWC19,GW20,SL20,CBBZ23,STW24,AST24]. For instance, the circuit satisfiability relation reduces to checking that a set of multiplication and addition gate constraints [BFLS91,Sha92,BFL92] evaluate to zero, the grand product relation reduces to checking a set of constraints between coefficients of polynomials at neighboring monomials evaluate to zero [SL20,GW20], and modern NP-complete relations, such as R1CS [GGPR13], CCS [STW23], and Plonkish [GWC19] reduce to checking that entry-wise constraints on a fixed set of vectors evaluate to zero. By formally modeling such interactive reductions as *reductions of knowledge* [KP23], a folding scheme for the zero-check relation would imply a folding scheme for all of these relations.

ZeroFold: A new folding scheme for the zero-check relation

We introduce ZeroFold, a two-round folding scheme for the zero-check relation. This folding scheme internally invokes a *single* round of the sum-check protocol [LFKN90]. The prover's work is in the folding scheme is the cost to commit to its witness and some finite field operations in the single round of

the sum-check protocol. Notably, if the witness contains "small" field elements, the prover only commits to "small" field elements. This makes these commitments very fast to compute (an order of magnitude faster than committing to arbitrary field elements), a property also leveraged in prior works like Spartan [Set20,STW23], HyperNova [KS24], Protostar [BC24b], Lasso [STW24], and Jolt [AST24]. Furthermore, our prover benefits from recent efficiency improvements to the sum-check prover [DT24]. The verifier's work is a constant number of group scalar multiplications, field operations, and hash computations. Our folding scheme naturally extends to folding an arbitrary number of instances at once (i.e, ZeroFold is a multi-folding scheme [KS24]), while ensuring that costs grow only linearly for the prover and the verifier in the number of instances. Note that the communication within the single round of the sum-check protocol and the verifier's work for sum-check messages do *not* depend on the number of instances folded. When folding n instances, ZeroFold requires only $\log n$ rounds of the sum-check protocol.

Theorem 1 (Folding zero-check (Informal)). *There exists a folding scheme for folding $n \geq 2$ instances in the zero-check relation, ZC, with $1 + \log n$ rounds, an $O(\log n)$ communication complexity, an $O(n)$ prover time complexity, and an $O(\log n)$ verifier time complexity.*

As a concrete example, suppose that the verifier holds two linearly homomorphic commitments $\overline{w}_0$ and $\overline{w}_1$, and would like to fold the task of checking that the prover knows openings $w_0 \in \mathbb{F}^n$ and $w_1 \in \mathbb{F}^n$ such that, say, all the elements are 0 or 1. This can be reduced to the task of folding two zero-check statements

$$0 = w_0(x) \cdot w_0(x) - w_0(x)$$
$$0 = w_1(x) \cdot w_1(x) - w_1(x)$$

for all $x \in \{0,1\}^\ell$, where $w_i(x)$ indicates evaluating the polynomial representation of w_i at location x and $\ell = \log n$.

A prior approach [BFL92,BTVW14,Set20,CBBZ23], notably from [BTVW14], for encoding zero-check is to embed each of the constraints into coefficients of a Lagrange polynomial, and then check that this polynomial is zero when evaluated at a random point. In particular the verifier first samples a challenge τ and instead checks that

$$0 = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot (w_0(x) \cdot w_0(x) - w_0(x)) \tag{1}$$
$$0 = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot (w_1(x) \cdot w_1(x) - w_1(x)), \tag{2}$$

where X is the corresponding decimal representation if x is treated as the bit representation i.e., $X = \sum_{i=0}^{\ell-1} 2^i \cdot x_i$. This essentially amounts to folding the sum-check problem, which checks the sum of evaluations of a polynomial over a

suitable hypercube. While sum-check is decidedly an easier problem to work with, it is still not clear how to fold the above instances: the verifier cannot simply output a folded instance as a random linear combination of the input instances due to the inherent non-linearity of the summand polynomial.

To solve the non-linearity issue, we devise a new folding technique that utilizes a *single* round of the sum-check protocol, which we refer to as $\mathsf{SumFold}$. The essential idea in $\mathsf{SumFold}$ is that we can devise new polynomials $f(b, x)$ such that $f(0, x) = w_0(x)$ and $f(1, x) = w_1(x)$. Then, checking Equations (1), and (2) is equivalent to checking that

$$Q(B) = \sum_{b \in \{0,1\}} \mathsf{eq}(b, B) \cdot \sum_{x \in \{0,1\}^\ell} \tau^X \cdot (f(B, x) \cdot f(B, x) - f(B, x)) \tag{3}$$

is the zero polynomial, where eq is a Lagrange polynomial such that $\mathsf{eq}(b, b') = 1$ if $b = b'$ and 0 otherwise for $b, b' \in \{0, 1\}$. Then, the verifier can check Equation (3) with overwhelming probability by sampling a random challenge $\beta \in \mathbb{F}$ and checking that $0 = Q(\beta)$.

Then, by running the sum-check protocol [LFKN90] on the outer sum for a *single* round, the verifier can reduce the task of checking Equation (3) to the task of checking

$$T = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot f(\beta, x) \cdot f(\beta, x) - f(\beta, x),$$

for some value T. By the construction of f, for $w \leftarrow w_1 + \beta \cdot w_2$, this amounts to checking

$$T = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot (w(x) \cdot w(x) - w(x)).$$

The verifier can accordingly output a folded instance $\overline{w} \leftarrow \overline{w}_1 + \beta \cdot \overline{w}_2$.

Hence, the verifier has folded two zero-check instances into a single sum-check instance. The verifier can then fold in new zero-check instances by first reducing to a sum-check instance (the zero-check reduction in Figure 1), and then repeating the above procedure (the nested sum-check reduction in Figure 1).

However, several challenges remain. First, with each additional zero-check instance, a new challenge τ must be sampled, and polynomials encoding the powers of τ (and the corresponding commitments) must also be folded. As it is too expensive for the verifier to compute these commitments in each folding step, this task must be outsourced to the prover. This places an additional burden on the verifier in each step to check that the claimed commitment to the powers of τ indeed agrees with τ. In applications such as IVC, where the verifier is represented as an arithmetic circuit, this is overly expensive and infeasible. Our idea here is to encode this check itself as *another* zero-check instance, which we refer to as $\mathsf{ZC}_{\mathsf{PC}}$, and bootstrap the existing folding scheme to fold these checks

alongside. Moreover, even on the prover's end, committing to the full vector of the powers of τ (which scales with 2^ℓ) is too expensive. We show how this can be circumvented by having the prover commit to two $\sqrt{n}$-sized vectors, one with the powers of τ and the other with a strided powers of τ (a similar pattern appears in tensor polynomial commitment schemes [GLS+23]). The prover and the verifier can then consider a variant of the sum-check relation, which we refer to as the *nested sum-check relation* (NSC), that computes the tensor product of these two vectors to produce the full powers of τ on the fly. Figure 1 summarizes the resulting folding scheme.

NeutronNova: Folding schemes for more complex relations

NeutronNova is the resulting suite of folding schemes that result from sequentially composing a reduction of knowledge (RoK) from a relation $\mathcal{R}$ to any number of zero-check instances with the folding scheme for zero-check.

This immediately provides a new folding scheme with attractive efficiency characteristics for customizable constraint systems (CCS), an NP-complete relation that generalizes R1CS [GGPR13], Plonkish [GWC19], and AIR [BSBHR19] and naturally reduces to zero-check. We provide a comparison between NeutronNova's folding scheme for CCS and prior work in Section 1.2.

Lemma 1 (Folding CCS (Informal)). *There exists a folding scheme for the CCS relation,* CCS*, with the same round complexity, communication complexity, prover time complexity, and verifier time complexity as ZeroFold.*

Moreover, by the results of Kothapalli and Setty [KS24, Lemma 4], we have that a folding scheme for CCS induces a corresponding non-uniform IVC (NIVC) scheme (i.e., IVC that supports different functions in each step of execution) over arbitrary degree constraints with a matching cost profile.

Beyond CCS, we formally describe a reduction of knowledge (RoK) from the grand-product relation, where a witness is a vector, and an instance is a commitment to a vector and a claimed product of all entries in the witness, to the zero-check relation. This RoK is based on the grand-product argument of Setty and Lee [SL20] and it is a non-interactive RoK.

Lemma 2 (Folding grand-product (Informal)). *There exists a folding scheme for folding n instances in the grand-product relation,* GP*, with $2 + \log n$ rounds and the same communication complexity, prover time complexity, and verifier time complexity as ZeroFold.*

We additionally describe a RoK from an indexed lookup relation to four instance-witness pairs in the grand product relation. This RoK is based on Lasso [STW24] and consists of a single round of interaction. By sequentially composing the prior two RoKs, we get a RoK from indexed lookups to zero-checks and hence a folding scheme for indexed lookups. We compare this with prior lookup arguments for folding schemes in Section 1.2.

Lemma 3 (Folding lookup (Informal)). *There exists a folding scheme for folding n instances in the lookup relation,* LKP*, with $3 + \log 4n$ rounds, and the same communication complexity, prover time complexity, and verifier time complexity as ZeroFold.*

1.2 Related work

HyperNova. HyperNova [KS24] is a folding scheme for CCS, where the prover runs the sum-check protocol [LFKN90] to reduce an instance-witness pair in CCS [STW23] into an instance-witness pair in linearized CCS, which admits a simple folding scheme based on random linear combination without any additional help from the prover. Beyond committing to the witness, HyperNova's prover's work is merely finite field operations in the sum-check protocol. Furthermore, HyperNova is a multi-folding scheme that can fold $k \geq 2$ instances at once. Leveraging this, [ZZD23] show that HyperNova naturally extends to provide a generalization of IVC called PCD [BCCT13]. One can apply their transformation to NeutronNova's folding scheme for CCS to obtain PCD.

NeutronNova improves upon HyperNova: NeutronNova avoids running the sum-check protocol in entirety, so NeutronNova's prover's work in the sum-check protocol is lower by $\approx 2\times$. More importantly, NeutronNova features a dramatically smaller verifier circuit. HyperNova's verifier circuit performs $k - 1$ group scalar multiplications and NeutronNova's verifier circuit performs $k + 1$ group scalar multiplications. However, HyperNova's verifier circuit verifies the sum-check protocol messages from $\log m$ rounds, where m is the number of constraints in CCS. HyperNova's verifier circuit incurs $O(d \cdot \log m)$ hash computations and field operations, where d is the degree of CCS constraints. In contrast, NeutronNova's verifier circuit incurs $O(d)$ hash computations and field operations. Overall, NeutronNova's verifier circuit is smaller.

Protostar. Protostar [BC23] focuses on folding special-sound protocols (with a algebraic verifiers) and it provides special-sound protocols for various relations including Plonkish, CCS, and lookups. To fold special-sound protocols, Protostar folds the verifier of a non-interactive special sound protocol expressed as a relation. This relation can be naturally reduced to the zero-check relation.

Focusing on their folding scheme for CCS, the prover's work and the verifier circuit size under NeutronNova's folding scheme for CCS are similar to that of Protostar's. Unfortunately, Protostar cannot efficiently fold more than two instances at once. To fold $k > 2$ instances, Protostar has to fold them one-by-one increasing the verifier circuit size by a factor of $O(k)$ compared to folding two instances. Alternatively, one can attempt to fold all k instances at once, but the costs grow exponentially in k [EG23, §1.2]. Furthermore, unlike Protostar, NeutronNova employs the sum-check protocol as a black box, so it can leverage recent optimizations [Gru24,DT24] to the sum-check protocol.

Protostar also describes a lookup argument within folding schemes. Their folding scheme is based on logarithmic derivatives [Hab22] (which relies on grand sums of

ratios of field elements) whereas the RoK that we describe for lookups is based on Lasso [STW24] (which relies on grand products). As described, they have similar asymptotic efficiency. However, by using the hybrid reduction of Quarks (§5.2), our approach can avoid commitments to arbitrary field elements. Regardless, we could devise a RoK from lookups to zero-checks relying on logarithmic derivatives.

Recently, Bünz and Chen [BC24b] extend Protostar with access to a global read-write memory. One can replace Protostar with NeutronNova in their construction to achieve better efficiency, especially due to NeutronNova's support for folding multiple instances at once.

Protogalaxy. Protogalaxy [EG23] extends Protostar [BC23] to support folding $k > 2$ instances at once. In Protogalaxy, an instance is logarithmic in the number of constraints folded, whereas in NeutronNova, the instances are constant-sized. In Protogalaxy, the verifier circuit performs $O(d + \log m)$ finite field operations and hash computations when folding two instances. Whereas, with NeutronNova, this is only $O(d)$. Their prover time scales super-linearly with the number of instances folded i.e., $O(k \log k)$, whereas in NeutronNova, the prover time scales linearly with k. Protogalaxy describes alternate schemes that scale better for larger values of k and handle the more general case of multiple running instances, but remarks that it may have worse constants than their main protocol. In any case, the instances are still logarithmic in the number of constraints folded. Recent work [EGS+24] applies Protogalaxy to build a relaxed version of IVC, where they apply Protogalaxy to a relation that appears closely related to our zero-check relation. One can replace Protogalaxy with NeutronNova in their construction and achieve better efficiency due to our improved folding scheme.

LatticeFold. LatticeFold [BC24a] describes a HyperNova-like folding scheme in the lattice setting. Unlike most folding schemes including NeutronNova, LatticeFold provides plausible post-quantum security. However, LatticeFold, like HyperNova, runs the full sum-check protocol, so it incurs higher recursion overheads than NeutronNova. Furthermore, their prover must commit to low-norm versions of witness vectors and prove range checks to ensure that each entry in the committed vector is a value in the range $[b]$. The degree of the sum-check is $\min(2b, d)$, so choosing a small value for b (e.g., $b = 2$) makes the prover commit to an excessive number of witness vectors. On the other hand, choosing a large value of b increases the degree of the sum-check protocol, which in turn increases the prover's work in the sum-check protocol and the verifier circuit sizes.

Folding schemes for non-homomorphic commitments. Recently, there is work to generalize Protostar-type folding schemes to non-homomorphic commitments (e.g., Merkle commitments to codewords) [BMNW24]. Compared to NeutronNova, they provide plausible post-quantum security. Unfortunately, they still require excessive amounts of hashing to verify Merkle membership proofs. Furthermore, the resulting IVC schemes are limited to a bounded number of steps (i.e., concrete attacks exist for non-constant recursion depth).

Mova and Ova. Mova [DGMV24] is a recent folding scheme that improves on Nova [KST22] by avoiding a commitment to the cross-term. NeutronNova (like

its predecessors HyperNova and Protostar) naturally achieves this property. Furthermore, unlike NeutronNova, Mova is limited to folding two instances at once. Moreover, when folding two instances, Mova requires a logarithmic number of hashes and finite field operations in the verifier circuit, whereas NeutronNova requires only a constant number of hashes and finite field operations.

Ova [ova] reduces the number of group scalar multiplications in Nova by 1. This is an improvement atop Nova in terms of verifier circuit sizes (by about 10–13%). Unfortunately, the prover still needs to compute and commit to a cross-term, which can contain arbitrary field elements even when witness elements are "small", a problem solved by prior works [KS24,BC23,EG23] including this work.

Nebula. In a companion work, Nebula [AS24] provides an efficient read-write memory primitive in folding schemes. The read-write memory is globally accessible across different steps of NIVC and beyond. To achieve this, Nebula provides a natural generalization of NIVC, which they refer to as commitment-carrying NIVC (CC-NIVC): a proof contains an incremental commitment to non-deterministic witness provided at each step of NIVC. To construct CC-IVC, they adapt HyperNova's compiler to provide a compiler from multi-folding schemes to CC-NIVC. Using CC-NIVC, Nebula retrofits offline memory checking [BEG$^+$91,SAGL18,STW24,AST24] within NIVC. Since they use a folding scheme as a black box, one can adapt their techniques to our setting and directly reduce read-write memory checking to zero-check and use NeutronNova's folding scheme for zero-check. We leave this to the future work.

Jolt+Nova. A naive approach [ST24] to make Jolt space efficient is to recursively compose it with Nova [KST22]: Jolt produces a proof of some number of CPU cycles at once, which is verified using a step of Nova. Unfortunately, it requires writing optimized circuits that check Jolt's proofs. Furthermore, this approach *requires* producing a Jolt proof. In contract, rather than proofs, NeutronNova generates zero-check instances (roughly, witness commitments), which are then folded with a single round of the sum-check protocol. Furthermore, NeutronNova's single-round sum-check protocol makes it more friendly to GPUs and ASICs than $O(\log^2 n)$ rounds of the sum-check protocol used in Jolt (where n is the number of CPU cycles).[3] Overall, NeutronNova provides a more efficient and a more direct route to build zkVMs that are both space and time efficient.

2 Preliminaries

In this section, we fix our notation and recall reductions of knowledge, which we use throughout our development. In Appendix A, we formally present multilinear polynomials and relevant properties, commitment schemes, arguments of knowledge, and IVC.

[3] Jolt can make the number of rounds closer to $O(\log n)$ by using the hybrid grand product protocol of [SL20], rather their current approach [GKR08,Tha13]. However, concretely, NeutronNova's round complexity will still be an order of magnitude better.

2.1 Notation

We let λ to denote the security parameter. We let $\mathsf{negl}(\lambda)$ to denote a negligible function in λ. We write $\Pr[X] \approx \epsilon$ to mean that $|\Pr[X] - \epsilon| = \mathsf{negl}(\lambda)$. Throughout the paper, the depicted asymptotics depend on λ, but we elide this for brevity. We let PPT denote probabilistic polynomial time and let EPT denote expected probabilistic polynomial time. We let $[n]$ denote the set $\{1, \ldots, n\}$. We let $\{u_i\}_{i \in [n]}$ denote the set $\{u_1, \ldots, u_n\}$.

We let $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p) and let $\mathbb{F}^n$ denote vectors of length n over elements in $\mathbb{F}$. We write $\mathbb{F}^d[X_1, \ldots, X_n]$ to denote multivariate polynomials over field $\mathbb{F}$ in the variables $(X_1, \ldots, X_n)$ with degree bound d for each variable. We omit the superscript if there is no degree bound. We denote vectors as $\vec{v} = (v_1, \ldots, v_n)$. Given a vector of polynomials $\vec{g}$, we let $\vec{g}(x) = (g_1(x), \ldots, g_n(x))$. We let $\mathsf{eq}(x, y) \in \mathbb{F}^1[X_1, \ldots, X_\ell, Y_1, \ldots, Y_\ell]$ denote the polynomial that outputs 1 if $x = y$ and 0 otherwise for $x, y \in \{0,1\}^\ell$. For vector $v \in \mathbb{F}^n$ we let $\widetilde{v} \in \mathbb{F}^1[X_1, \ldots, X_{\log n}]$ denote the multilinear polynomial extension of v (i.e., $\widetilde{v}(i) = \sum_j \mathsf{eq}(i, j) \cdot v_j$).

2.2 Reductions of Knowledge

We now recall the reductions of knowledge framework, introduced by Kothapalli and Parno [KP23]. Reductions of knowledge are a generalization of arguments of knowledge, in which a verifier interactively *reduces* checking a prover's knowledge of a witness in a relation $\mathcal{R}_1$ to checking the prover's knowledge of a witness in another (simpler) relation $\mathcal{R}_2$. In particular, both parties take as input a claimed instance u_1 to be checked, and the prover additionally takes as input a corresponding witness w_1 such that $(u_1, w_1) \in \mathcal{R}_1$. After interaction, the prover and verifier together output a new statement u_2 to be checked in place of the original statement, and the prover additionally outputs a corresponding witness w_2 such that $(u_2, w_2) \in \mathcal{R}_2$.

We modify the original definition to account for the preprocessed setting, in which a deterministic *encoder* algorithm preprocesses a portion of the statement called the structure (typically encoding a circuit or set of constraints) *once* and outputs a prover and verifier key which can be used to verify any number of witnesses (e.g., variable assignments) against this structure. We enable the encoder to additionally output a new structure to check the output instance-witness pair against (this can be preprocessed by a subsequent encoder).

Definition 1 (Reduction of Knowledge [KP23]). *Consider relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance, and witness tuples. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic algorithm $\mathcal{K}$, called the generator, the prover, the verifier and the encoder respectively with the following interface.*

- $\mathcal{G}(\lambda, n) \to \mathsf{pp}$*: Takes as input security parameter λ and size parameters n. Outputs public parameters* pp.

- $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$*: Takes as input public parameters* pp *and structure* s_1*. Outputs prover key* pk*, verifier key* vk*, and updated structure* s_2*.*
- $\mathcal{P}(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)$*: Takes as input public parameters* pp*, and an instance-witness pair* (u_1, w_1)*. Interactively reduces the task of checking* $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ *to the task of checking* $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$*.*
- $\mathcal{V}(\mathsf{pk}, u_1) \to u_2$*: Takes as input public parameters* pp*, and an instance* u_1 *in* $\mathcal{R}_1$*. Interactively reduces the task of checking instance* u_1 *to the task of checking a new instance* u_2 *in* $\mathcal{R}_2$*.*

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ *denote the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$*. We treat* $\langle \mathcal{P}, \mathcal{V} \rangle$ *as a function that takes as input* $((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$ *and runs the interaction on the prover's input* (pk, u_1, w_1) *and the verifier's input* (pp, u_1)*. At the end of the interaction,* $\langle \mathcal{P}, \mathcal{V} \rangle$ *outputs the verifier's instance* u_2 *and the prover's witness* w_2*. A reduction of knowledge* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *satisfies the following conditions.*

(i) Completeness: For any PPT adversary $\mathcal{A}$*, given* $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$*,* $(\mathsf{s}_1, u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$ *such that* $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ *and* $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ *we have that the prover's output instance is equal to the verifier's output instance* u_2*, and that*

$$(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1)) \in \mathcal{R}_2.$$

(ii) Knowledge soundness: For any expected polynomial-time adversaries $\mathcal{A}$ *and* $\mathcal{P}^*$*, there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that given* $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$*,* $(\mathsf{s}_1, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$*, and* $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$*, we have that*

$$\Pr[(\mathsf{pp}, \mathsf{s}_1, u_1, \mathcal{E}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st})) \in \mathcal{R}_1] \approx \Pr[(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})) \in \mathcal{R}_2].$$

(iii) Public reducibility: There exists a deterministic polynomial-time function φ *such that for any PPT adversary* $\mathcal{A}$ *and expected polynomial-time adversary* $\mathcal{P}^*$*, given*

$$\begin{aligned} \mathsf{pp} &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathsf{s}_1, u_1, \mathsf{st}) &\leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) &\leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1), \end{aligned}$$

and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ *with the interaction transcript* tr*, we have that* $\varphi(\mathsf{pp}, \mathsf{s}_1, u_1, \mathsf{tr}) = u_2$*.*

As with arguments of knowledge, we can define various additional properties for reductions of knowledge such as succinctness (the communication is sublinear in the witness size), non-interactivity (the interaction consists of a single message from the prover), public-coin (the verifier only sends random challenges), and tree-extractability (there exists an extractor that can produce a satisfying witness given a tree of accepting transcripts). We define these properties in Appendix A.3.

Typically, we are interested in reducing several relations at once. We can interpret several relations as a single relation using the following product operator.

Definition 2 (Relation product). *For relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameter, structure, instance, and witness pairs we define the relation product as follows.*

$$\mathcal{R}_1 \times \mathcal{R}_2 = \left\{ (\mathsf{pp}, \mathsf{s}, (u_1, u_2), (w_1, w_2)) \,\middle|\, (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1, (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2 \right\}.$$

We let $\mathcal{R}^n$ denote $\mathcal{R} \times \ldots \times \mathcal{R}$ for n times.

A motivating property of reductions of knowledge is that they are composable, allowing us to build complex reductions by stitching together simpler ones. In particular, given reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ we have that $\Pi_2 \circ \Pi_1$ (that is, running Π_1 first and then running Π_2 on the outputs) is a reduction of knowledge from R_1 to $\mathcal{R}_3$. Similarly, given reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4$ we have that $\Pi_1 \times \Pi_2$ (that is, independently running Π_1 and Π_2 on pairs of inputs) is a reduction of knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$. We formally define sequential and parallel composition in Appendix A.3.

In this work we are chiefly interested in building folding schemes, a particular type of reduction of knowledge that reduces the task of checking several instances in some relation $\mathcal{R}_2$ into a running instance in a relation $\mathcal{R}_1$.

Definition 3 (Folding scheme). *A folding scheme for $\mathcal{R}_1^m$ and $\mathcal{R}_2^n$ is a reduction of knowledge of type $\mathcal{R}_1^m \times \mathcal{R}_2^n \to \mathcal{R}_1$ where the encoder outputs its input structure. We call a reduction of type $\mathcal{R}^n \to \mathcal{R}$ simply as a folding scheme for $\mathcal{R}$.*

2.3 The sum-check protocol

Recall that the standard sum-check relation checks that the sum of evaluations of an ℓ-variate polynomial Q (under a commitment) on the Boolean hypercube results in some value T. Formally, the sum-check relation is defined as follows.

Definition 4 (Unstructured sum-check relation). *Let $(\mathsf{Gen}, \mathsf{Commit})$ denote an additively homomorphic commitment scheme. Consider a size bound $\ell \in \mathbb{N}$. The unstructured sum-check relation USC over public parameter, instance, witness pairs is defined as follows.*

$$\mathsf{USC} = \left\{ (\mathsf{pp}, (\overline{Q}, T), Q) \,\middle|\, \begin{array}{l} Q \in \mathbb{F}[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Commit}(\mathsf{pp}, Q), \\ T = \sum_{x \in \{0,1\}^\ell} Q(x) \end{array} \right\}$$

Central to our development is the sum-check protocol [LFKN90], which, when recast as a reduction of knowledge [BCS21], reduces from the sum-check relation to the polynomial evaluation relation, which we define below.

Definition 5 (Polynomial evaluation relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively homomorphic commitment scheme. Consider size bound* $\ell \in \mathbb{N}$*. We define the polynomial evaluation relation,* PE*, as follows.*

$$\mathsf{PE} = \left\{ (\mathsf{pp}, (\overline{Q}, x, y), g) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Commit}(\mathsf{pp}, Q), \\ y = Q(x) \end{array} \right\}.$$

Lemma 4 (The sum-check protocol). *There exists a succinct, public-coin, tree-extractable reduction of knowledge (Lemma 15) from* USC *to* PE *compatible with all encoder, generator, and commitment algorithms where the output commitment is the same as the input commitment. For polynomials in* $\mathbb{F}^d[X_1, \ldots, X_\ell]$ *the communication complexity is* $O(d \cdot \ell)$ *elements in* $\mathbb{F}$*.*

3 SumFold: A folding scheme for the sum-check relation

In this section, we design a folding scheme for instance-witness pairs in the sum-check relation by using the sum-check protocol itself as a core building block. In the next section, we show that this enables a folding scheme for zero-check.

A strawman folding scheme for the unstructured sum-check relation (Definition 4) is quite natural. By linearity, to interactively fold two unstructured sum-check instances $(T_1, \overline{Q}_1)$ and $(T_2, \overline{Q}_2)$ the verifier can send a random challenge ρ and check instead that the prover knows a polynomial Q to the folded instance $(T_1 + \rho \cdot T_2, \overline{Q}_1 + \rho \cdot \overline{Q}_2)$, which indeed the prover can compute so long as it knows polynomials Q_1 and Q_2 that satisfy $(T_1, \overline{Q}_1)$ and $(T_2, \overline{Q}_2)$.

Sum-check over structured polynomials. Challenges arise in contexts of interest, where the verifier does not explicitly hold commitments to Q_1 and Q_2. In particular, when encoding instances in NP-complete relations, where the polynomial Q is more concisely represented as a set of multilinear polynomials $(g_0, \ldots, g_{t-1})$ which we denote as $\vec{g}$ and a polynomial $F \in \mathbb{F}[Y_1, \ldots, Y_t]$ such that $Q(x) = F(g_0(x), \ldots, g_{t-1}(x))$ for all x in $\{0,1\}^\ell$. The multilinear polynomials $\vec{g}$ themselves are typically a linear function G of some (smaller) set of underlying vectors $\vec{w} = (w_0, \ldots, w_{s-1})$ and sometimes a public vector x. For example, when encoding circuit satisfiability, polynomials g_1, g_2, and g_3 could represent left, right, and output gate values selected from a single vector (w, x) encoding the full list of intermediate wire values and inputs.

Hence, in practice, the commitment to the polynomial Q is much more concisely represented as a commitments to polynomials $\vec{w}$ alongside public vector x and functions F and G. With this the following more accurately captures the relation we aim to fold.

Definition 6 (Sum-check relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size*

bounds $n, m, \ell, d, s, t \in \mathbb{N}$. We define the sum-check relation, SC, as follows

$$\mathsf{SC} = \left\{ (\mathsf{pp}, (F, G), (T, \vec{u}, \mathsf{x}), \vec{w}) \;\middle|\; \begin{array}{l} T \in \mathbb{F}, F \in \mathbb{F}^d[Y_1, \ldots, Y_t], \vec{w} \in (\mathbb{F}^n)^s, \mathsf{x} \in \mathbb{F}^m, \\ \mathsf{Commit}(\mathsf{pp}, w_i) = u_i, \\ \vec{g} \in (\mathbb{F}^1[X_1, \ldots, X_\ell])^t \leftarrow G(\vec{w}, \mathsf{x}) \\ T = \sum_{x \in \{0,1\}^\ell} F(\vec{g}(x)) \end{array} \right\}$$

where G is linear over the input vector $(\vec{w}, \mathsf{x})$.

Challenges with folding sum-check. The simple folding scheme that worked for USC no longer works for SC due to the potential non-linearity of F. To demonstrate this, consider the following. Concretely, suppose that G simply outputs multilinear extensions of its inputs (e.g., the multilinear extension of the witness and the public input concatenated) and F multiplies all its inputs. Suppose then that the verifier holds two sets of commitments $(\overline{g}_0, \ldots, \overline{g}_{t-1})$ and $(\overline{h}_0, \ldots, \overline{h}_{t-1})$ and would like to check that the prover knows two sets of ℓ-variate multilinear polynomial openings $(g_0, \ldots, g_{t-1})$ and $(h_0, \ldots, h_{t-1})$ such that

$$T_0 = \sum_{x \in \{0,1\}^\ell} g_0(x) \cdot g_1(x) \cdot \ldots \cdot g_{t-1}(x) \tag{4}$$

$$T_1 = \sum_{x \in \{0,1\}^\ell} h_0(x) \cdot h_1(x) \cdot \ldots \cdot h_{t-1}(x). \tag{5}$$

Suppose that the verifier samples a challenge ρ (as before), and computes the folded instance:

$$(T_0 + \rho \cdot T_1, (\overline{g}_0 + \rho \cdot \overline{h}_0), \ldots, (\overline{g}_{t-1} + \rho \cdot \overline{g}_{t-1})).$$

Then, while the prover can compute the corresponding openings $(g_0 + \rho \cdot h_0), \ldots, (g_{t-1} + \rho \cdot g_{t-1})$, they no longer satisfy the sum-check relation because

$$\begin{aligned} T_0 + \rho \cdot T_1 &= \sum_{x \in \{0,1\}^\ell} g_0(x) \cdot \ldots \cdot g_{t-1}(x) + \rho \cdot h_0(x) \cdot \ldots \cdot h_{t-1}(x) \\ &\neq \sum_{x \in \{0,1\}^\ell} (g_0(x) + \rho \cdot h_0(x)) \cdot \ldots \cdot (g_{t-1}(x) + \rho \cdot h_{t-1}(x)). \end{aligned}$$

Our solution: SumFold. We are now ready to describe NeutronNova's folding scheme for the sum-check relation, which we refer to as $\mathsf{SumFold}$. Recall that the crux of $\mathsf{SumFold}$ is that we embed polynomials g_i and h_i as Lagrange coefficients in a single-variable linear polynomial f_i (and likewise for T_0 and T_1). The prover and the verifier then run the sum-check protocol [LFKN90] for a *single* round to bind this new variable to a random value.

Indeed, for all $j \in [t]$, define f_i as a multilinear polynomial in $\ell + 1$ variables such that for all $x \in \{0,1\}^\ell$, $f_j(0, x) = g_j(x)$ and $f_j(1, x) = h_j(x)$. Specifically, we define

$$f_j(b, x) = (1 - b) \cdot g_j(x) + b \cdot h_j(x).$$

Then, checking Claims (4) and (5) is equivalent to checking

$$T_0 = \sum_{x \in \{0,1\}^\ell} f_0(0,x) \cdot f_1(0,x) \cdot \ldots \cdot f_{t-1}(0,x)$$
$$T_1 = \sum_{x \in \{0,1\}^\ell} f_0(1,x) \cdot f_1(1,x) \cdot \ldots \cdot f_{t-1}(1,x)$$

Embedding the above polynomials as Lagrange coefficients, the verifier can equivalently check

$$\sum_{b \in \{0,1\}} \mathsf{eq}(X,b) \cdot T_b = \sum_{b \in \{0,1\}} \mathsf{eq}(X,b) \cdot \sum_{x \in \{0,1\}^\ell} f_0(b,x) \cdot f_1(b,x) \cdot \ldots \cdot f_{t-1}(b,x).$$

Then, the verifier picks a random challenge $\beta \in \mathbb{F}$, and by the Schwartz-Zippel Lemma (Lemma 12), it reduces to checking

$$\sum_{b \in \{0,1\}} \mathsf{eq}(\beta,b) \cdot T_b = \sum_{b \in \{0,1\}} \mathsf{eq}(\beta,b) \cdot \sum_{x \in \{0,1\}^\ell} f_0(b,x) \cdot f_1(b,x) \cdot \ldots \cdot f_{t-1}(b,x).$$

The prover and verifier then apply the sum-check protocol [LFKN90] for a *single* round to the outer sum to bind the variable b to a random challenge $r_b \in \mathbb{F}$. Then, for some $T' \in \mathbb{F}$, the verifier is left to check the following claim about the inner sum

$$T' = \sum_{x \in \{0,1\}^\ell} f_0(r_b,x) \cdot f_1(r_b,x) \cdot \ldots \cdot f_{t-1}(r_b,x).$$

At this point, for $j \in [t]$, the verifier computes commitments to multilinear polynomials $f_j(r_b)$ (i.e., f_j with the first variable set to r_b), homomorphically as

$$\begin{aligned}\mathsf{Commit}(\mathsf{pp}, f_j(r_b)) &= (1-r_b) \cdot \mathsf{Commit}(\mathsf{pp}, f_j(0)) + r_b \cdot \mathsf{Commit}(\mathsf{pp}, f_j(1)) \\ &= (1-r_b) \cdot \overline{g}_j + r_b \cdot \overline{h}_j\end{aligned}$$

Finally, the verifier produces the folded instance

$$(T', ((1-r_b) \cdot \overline{g}_0 + r_b \cdot \overline{h}_0), \ldots, ((1-r_b) \cdot \overline{g}_{t-1} + r_b \cdot \overline{h}_{t-1})).$$

Furthermore, the prover can produce the satisfying folded witness $((1-r_b) \cdot g_0 + r_b \cdot h_0), \ldots, ((1-r_b) \cdot g_{t-1} + r_b \cdot h_{t-1})$.

SumFold generalizes naturally to fold *any* number of sum-check instances. If there are n sum-check instances, then the verifier samples $r_b \in \mathbb{F}^{\log n}$ and run the sum-check protocol for $\log n$ rounds. At the end of the protocol, the commitments and witnesses are folded using the evaluations of $\mathsf{eq}(r_b, i)$ for all $i \in \{0,1\}^{\log n}$ as weights. In the case of $n = 2$, these weights are simply $(1-r_b)$ and r_b as described above. We now formally describe our generalized construction.

Construction 1 (SumFold). We construct a folding scheme for SC. Consider size bounds $\ell, d, s, t \in \mathbb{N}$ and let $\nu = \log n$. Let $(\mathcal{P}_{\mathsf{sc}}, \mathcal{V}_{\mathsf{sc}})$ denote the sum-check protocol (Lemma 4). Consider an arbitrary generator algorithm and an underlying additively homomorphic commitment scheme. We define the encoder, prover, and verifier as follows.

$\underline{\mathcal{K}(\mathsf{pp}, (F, G))}$: Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, G), \perp)$ and structure (F, G).

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \{(T_i, \vec{u}_i, \mathsf{x}_i)\}_{i \in [n]}, \{\vec{w}_i\}_{i \in [n]})}$:

1. $\mathcal{V}$: Sample and send $\rho \xleftarrow{\$} \mathbb{F}^{\nu}$.
2. $\mathcal{P}, \mathcal{V}$: Compute $(c, r_b) \leftarrow \langle \mathcal{P}_{\mathsf{sc}}, \mathcal{V}_{\mathsf{sc}} \rangle((\mathsf{pk}, \mathsf{vk}), (\overline{Q}, T), Q)$, where

$$
\begin{aligned}
T &\leftarrow \sum_{i \in \{0,1\}^{\nu}} \mathsf{eq}(\rho, i) \cdot T_i \\
f_j(b, x) &\leftarrow \sum_{i \in \{0,1\}^{\nu}} \mathsf{eq}(b, i) \cdot g_{i,j}(x) \text{ where } \vec{g}_i \leftarrow G(\vec{w}_i, \mathsf{x}_i) \\
Q(b) &\leftarrow \mathsf{eq}(\rho, b) \cdot \left(\sum_{x \in \{0,1\}^{\ell}} F(f_1(b, x), \ldots, f_t(b, x)) \right) \\
\overline{Q} &\leftarrow ((F, G), \rho, (\vec{u}_i, \mathsf{x}_i)_{i \in [n]}).
\end{aligned}
$$

3. $\mathcal{P}, \mathcal{V}$: Output the folded instance-witness pair (the verifier only outputs the instance) $((T', \vec{u}, \mathsf{x}), \vec{w})$, where

$$
\begin{aligned}
T' &\leftarrow c \cdot \mathsf{eq}(\rho, r_b)^{-1} \\
u_j &\leftarrow \sum_{i \in \{0,1\}^{\nu}} \mathsf{eq}(r_b, i) \cdot u_{i,j} \\
\mathsf{x} &\leftarrow \sum_{i \in \{0,1\}^{\nu}} \mathsf{eq}(r_b, i) \cdot \mathsf{x}_i \\
w_j &\leftarrow \sum_{i \in \{0,1\}^{\nu}} \mathsf{eq}(r_b, i) \cdot w_{i,j}
\end{aligned}
$$

We formally prove the following theorem in Appendix B.1.

Theorem 2 (SumFold). *Construction 1 is a folding scheme for* SC *with* $1 + \log n$ *rounds, a communication complexity of* $O(d \log n)$ *field elements, a prover time complexity of* $O(ntd \cdot 2^{\ell})$ *field operations, and a verifier time complexity of* $O(d \log n)$ *field operations.*

Proof (Intuition). We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct PPT extractor χ that outputs a satisfying input witness with

probability $1 - \mathsf{negl}(\lambda)$ given a tree of accepting transcripts and the corresponding output instance-witness pairs in SC. We do so by leveraging the tree-extractor χ_{sc} guaranteed by the tree extractability of the sum-check protocol (Lemma 4).

In particular, χ can produce a sufficient number of accepting sub-trees for the underlying sum-check protocol by using the provided instance-witness pairs in SC to solve for satisfying instance-witness pairs output by the sum-check protocol in the polynomial evaluation relation PE. Then, the sum-check tree extractor χ_{sc} can produce a satisfying input witness in the unstructured sum-check relation USC for each sub-tree. By the binding property of the commitment scheme, these witnesses must all be identical. But, by interpolating over the verifier's initial challenge, this means that the witness produced by χ_{sc} must also be a satisfying witness for the input relation SC^n. □

4 ZeroFold: A folding scheme for the zero-check relation

In this section, we design $\mathsf{ZeroFold}$, a folding scheme for the zero-check relation, which asserts that a polynomial is zero over a prescribed set of points. We use $\mathsf{SumFold}$ (Construction 1), which is a folding scheme for the sum-check relation, as the central engine underlying our construction. In the next section, we show that this enables folding schemes for a variety of relations that reduce to zero-check.

As with folding unstructured sum-check instances, unstructured zero-check instances can be folded with a standard random linear combination. However, as with the sum-check relation, in practice, commitments to the involved polynomials are typically represented as commitments to a set of (smaller) witness vectors that generate the aforementioned polynomials. Hence, we are more accurately interested in folding the following relation.

Definition 7 (The zero-check relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size bounds* $n, m, \ell, d, s, t \in \mathbb{N}$*. We define the zero-check relation,* ZC*, as follows*

$$\mathsf{ZC} = \left\{ (\mathsf{pp}, (F, G), (\vec{u}, \mathsf{x}), \vec{w}) \;\middle|\; \begin{array}{l} F \in \mathbb{F}^d[Y_1, \ldots, Y_t], \vec{w} \in (\mathbb{F}^n)^s, \mathsf{x} \in \mathbb{F}^m, \\ \mathsf{Commit}(\mathsf{pp}, w_i) = u_i, \\ \vec{g} \in (\mathbb{F}^1[X_1, \ldots, X_\ell])^t \leftarrow G(\vec{w}, \mathsf{x}), \\ \forall x \in \{0,1\}^\ell.\ 0 = F(\vec{g}(x)) \end{array} \right\}$$

where G *is linear over the input vector* $(\vec{w}, \mathsf{x})$*.*

Once again, we can no longer use a standard random linear combination due to non-linearity. Our goal then is to *reduce* zero-check to the sum-check relation and then invoke $\mathsf{SumFold}$ from Section 3. This introduces various difficulties in terms of efficiency, which we methodically address.

4.1 From zero-check to (nested) sum-check

To illustrate core ideas, as in the previous section, suppose that the linear map G simply outputs multilinear extensions of its inputs and the polynomial

F multiplies all its inputs. Suppose then that the verifier holds two sets of commitments $(\overline{g}_0, \ldots, \overline{g}_{t-1})$ and $(\overline{h}_0, \ldots, \overline{h}_{t-1})$, and would like to check that the prover knows two sets of ℓ-variate multilinear polynomial openings $(g_0, \ldots, g_{t-1})$ and $(h_0, \ldots, h_{t-1})$ such that

$$0 = g_0(x) \cdot \ldots \cdot g_{t-1}(x)$$
$$0 = h_0(x) \cdot \ldots \cdot h_{t-1}(x)$$

for all $x \in \{0,1\}^\ell$. To fold these instances, our goal is to reduce them to a corresponding set of sum-check instances.

Recall that, from prior work [BTVW14], this can be done by having the verifier sample a challenge τ and instead check for all $x \in \{0,1\}^\ell$ that

$$0 = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot g_0(x) \cdot \ldots \cdot g_{t-1}(x)$$
$$0 = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot h_0(x) \cdot \ldots \cdot h_{t-1}(x)$$

where $X = \sum_{i=0}^{\ell} 2^i \cdot x_i$ (i.e., X is the corresponding decimal representation if x is treated as the bit representation).[4]

To match the interface of the sum-check folding scheme, we define the Lagrange polynomial $e(x) = \sum_{z \in \{0,1\}^\ell} \mathsf{eq}(z, x) \cdot \tau^Z$ encoding the powers of τ (i.e., e is the multilinear extension of the vector $[\tau^0, \ldots, \tau^{2^\ell}]$), and have the verifier check for all $x \in \{0,1\}^\ell$ that

$$0 = \sum_{x \in \{0,1\}^\ell} e(x) \cdot g_0(x) \cdot \ldots \cdot g_{t-1}(x)$$
$$0 = \sum_{x \in \{0,1\}^\ell} e(x) \cdot h_0(x) \cdot \ldots \cdot h_{t-1}(x)$$

Recall that in $\mathsf{SumFold}$, the verifier samples a challenge ρ and the prover computes the folded witness as $g_j + \rho \cdot h_j$ for $j \in [t]$ for which the verifier homomorphically computes the corresponding folded commitment $\overline{g}_j + \rho \cdot \overline{h}_j$. Then, by extension the prover will similarly need to fold e and the verifier will need to fold a corresponding commitment to e.

[4] We use the reduction from zero-check to sum-check from Clover [BTVW14], rather than from Spartan [Set20] although the latter is more widely adopted [SL20,CBBZ23,STW23,STW24,AST24,DP23,DT24]. Clover's variant contributes a soundness error of $2^\ell/|\mathbb{F}|$ whereas Spartan's variant contributes soundness error of at most $\ell/|\mathbb{F}|$. Both are acceptable as $|\mathbb{F}| \approx 2^{256}$ in our concrete instantiation. Furthermore, Clover's variant requires sampling a single challenge $\tau \in \mathbb{F}$ whereas Spartan's variant requires sampling a challenge from $\mathbb{F}^\ell$. We adopt the variant from Clover to keep NeutronNova's verifier circuit size independent of ℓ.

A key question is the following: how do the verifier and the prover represent a commitment to e, which is highly structured. Naturally, the prover can directly send a commitment to a vector of powers of τ, which for now we will assume is trusted. However, this requires committing to a vector of size equal to the number of constraints 2^ℓ. Furthermore, these vector entries are completely random. If the witness for the zero-check instance contains "small" field elements (e.g., witness elements are from the set $\{0, \ldots, 2^b - 1\}$, for $b = 32$), the cost to commit to e is at least an order of magnitude more expensive than the cost to commit to the witness, which is highly undesirable.

As a first attempt, τ itself could serve as a commitment to e. However, because e loses its tensor structure after folding, we will no longer have that $\tau + \rho \cdot \tau$ represents a commitment to $e + \rho \cdot e$. In particular, we have that

$$\begin{aligned} e + \rho \cdot e &= \sum_{z \in \{0,1\}} (1 + \rho) \cdot \mathsf{eq}(z, x) \cdot \tau^Z \\ &\neq \sum_{z \in \{0,1\}} \mathsf{eq}(z, x) \cdot (\tau + \rho \cdot \tau)^Z \end{aligned}$$

Our solution starts with the observation that

$$\begin{aligned} e(x) &= \sum_{z \in \{0,1\}^\ell} \mathsf{eq}(z, x) \cdot \tau^Z \\ &= \Big(\sum_{z_1 \in \{0,1\}^\ell} \mathsf{eq}(z_1, x_1) \cdot \tau^{Z_1} \Big) \cdot \Big(\sum_{z_2 \in \{0,1\}^\ell} \mathsf{eq}(z_2, x_2) \cdot \tau^{\sqrt{2^\ell} \cdot Z_2} \Big) \end{aligned}$$

where z_1 and z_2 represent the first and second half of z and Z_i is the decimal representation of z_i (for $i \in \{1, 2\}$).[5] Following this, we define

$$\begin{aligned} e_1(x_1) &= \sum_{z_1 \in \{0,1\}^\ell} \mathsf{eq}(z_1, x_1) \cdot \tau^{Z_1} \\ e_2(x_2) &= \sum_{z_2 \in \{0,1\}^\ell} \mathsf{eq}(z_2, x_2) \cdot \tau^{\sqrt{2^\ell} \cdot Z_2}. \end{aligned}$$

Now, we have that $e(x) = e_1(x_1) \cdot e_2(x_2)$. Hence, the verifier can instead check

$$\begin{aligned} 0 &= \sum_{x \in \{0,1\}^\ell} e_1(x_1) \cdot e_2(x_2) \cdot g_0(x) \cdot \ldots \cdot g_{t-1}(x) \\ 0 &= \sum_{x \in \{0,1\}^\ell} e_1(x_1) \cdot e_2(x_2) \cdot h_0(x) \cdot \ldots \cdot h_{t-1}(x) \end{aligned}$$

[5] A very recent work by Dao and Thaler [DT24] leverages a similar tensor decomposition applied to the multilinear extension of the equality function. Their focus is to speed up the task of *proving* a zero-check instance rather than folding multiple instance together. Although our purpose for decomposition is different, our prover also reaps benefits from lower field operation counts observed in [DT24] for the single round of the sum-check protocol in NeutronNova's folding scheme.

Now, the prover can directly provide commitments to e_1 and e_2, or, for efficiency, a single commitment $\overline{e}$ to the vector $e = (e_1, e_2)$, which only commits to a vector of size $2 \cdot \sqrt{2^\ell}$. By linearity, the verifier can compute the folded commitment $\overline{e} + \rho \cdot \overline{e}$ for which the prover can compute the folded witness $(e_1 + \rho \cdot e_1, e_2 + \rho \cdot e_2)$. Formally, the prover and the verifier have reduced to folding the following variant of sum-check. Note that the verifier still needs to check the correctness of the provided commitments (which we address below).

Definition 8 (Nested sum-check relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size bounds* $n, m, \ell, d, s, t \in \mathbb{N}$*. We define the nested sum-check relation,* NSC*, as follows*

$$\mathsf{NSC} = \left\{ \begin{pmatrix} (\mathsf{pp}, \\ (F, G), \\ (T, \vec{u}, \mathsf{x}, \overline{e}), \\ (\vec{w}, e)) \end{pmatrix} \middle| \begin{array}{l} T \in \mathbb{F}, F \in \mathbb{F}^d[Y_1, \ldots, Y_t], \vec{w} \in (\mathbb{F}^n)^s, \mathsf{x} \in \mathbb{F}^m, \\ \mathsf{Commit}(\mathsf{pp}, w_i) = u_i, \mathsf{Commit}(\mathsf{pp}, e) = \overline{e}, \\ \vec{g} \in \mathbb{F}^1[X_1, \ldots, X_\ell] \leftarrow G((\vec{w}, \mathsf{x})) \\ T = \sum_{x \in \{0,1\}^\ell} e_1(x_1) \cdot e_2(x_2) \cdot F(\vec{g}(x)) \end{array} \right\}$$

where G *is linear over the input vector* (w, x) *and* x_1 *and* x_2 *(likewise,* e_1 *and* e_2*) represent the first and second half of* x *(likewise,* e*).*

It appears that we can now apply $\mathsf{SumFold}$, but two problems remain. First, unlike the sum-check relation, the nested sum-check relation enforces an additional stipulation that e_1 is only evaluated on the first half of x and e_2 is only evaluated on the second half of x. As we will see in the next section, this is only a minor technicality which we can fix with an appropriate padding to e_1 and e_2. Second, when reducing from zero-check to the nested sum-check relation, the verifier must still ensure that e_1 and e_2 indeed contain appropriate powers of τ. Formally, the verifier is tasked with checking the following relation.

Definition 9 (Power-check relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size bound* m*. We define the power-check relation* PC *over public parameter, instance, witness tuples as follows*

$$\mathsf{PC} = \left\{ (\mathsf{pp}, (\overline{e}, \tau), e) \middle| \begin{array}{l} \mathsf{Commit}(\mathsf{pp}, e) = \overline{e}, \\ e_1 = (\tau^0, \tau^1, \tau^2, \ldots, \tau^{\sqrt{m}-1}) \\ e_2 = (\tau^0, \tau^{\sqrt{m}}, \tau^{2\sqrt{m}}, \ldots, \tau^{(\sqrt{m}-1)\cdot\sqrt{m}}) \end{array} \right\}$$

where e_1 *and* e_2 *represent the first and second half of* e*.*

From power-check to zero-check. Recall that for most applications it is too expensive for the verifier to check a power-check instance in each folding step. To circumvent this, our solution is for the prover and the verifier to also fold the power-check instances alongside the zero-check instances. Instead of designing a new folding scheme for the power-check relation, we reinterpret the power-check

relation as a particular type of zero-check relation. This allows us the bootstrap our eventual folding scheme for zero-check for the purpose of power-check.

Let $m = 2^\ell$. We observe that the power-check relation can be enforced with $2 \cdot \sqrt{m}$ quadratic constraints. The first half of e can be checked by checking if $e_0 = 1$ and that each subsequent entry is the result of multiplying the previous entry by τ. The second half of e can be checked by checking that $e_{\sqrt{m}} = 1$, checking that $e_{\sqrt{m}+1}$ (presumably $\tau^{\sqrt{m}}$) is the result of multiplying the last entry of the first half of e (presumably $\tau^{\sqrt{m}-1}$) by τ, checking that the subsequent entry $e_{\sqrt{m}+2}$ (presumably $\tau^{2\cdot\sqrt{m}}$) is the square of the previous entry, and then check that all subsequent entries result from multiplying the previous entry by $e_{\sqrt{m}+1}$.

To reduce to power-check to zero-check, we observe that each entry of e is checked to be the result of multiplying two other entries in (e, τ). Then, given e and τ we can fix the structure polynomial G_{PC} to set $g_1 \leftarrow \widetilde{e}$ and set g_2 and g_3 to contain the left and right inputs respectively to the multiplication operation. Then, F_{PC} can compute $g_1(x) - g_2(x) \cdot g_3(x)$, for all $x \in \{0,1\}^\ell$ thereby enforcing each multiplication constraint in the zero-check. Formally, we encode a power-check instance as a zero-check instance as follows.

Construction 2 (From power-check to zero-check). We have that $(\mathsf{pp}, (\overline{e}, \tau), e) \in \mathsf{PC}$ if and only if $(\mathsf{pp}, (F_{\mathsf{PC}}, G_{\mathsf{PC}}), (\overline{e}, \tau), e) \in \mathsf{ZC}$ where $G_{\mathsf{PC}}(e, \tau)$ produces $g_1 \leftarrow \widetilde{e}$, g_2, and g_3 such that

$$g_2(i) = \begin{cases} 1 & i = 0 \\ e_{i-1} & 1 \le i < \sqrt{m} \\ 1 & i = \sqrt{m} \\ e_{i-2} & i = \sqrt{m}+1 \\ e_{i-1} & i = \sqrt{m}+2 \\ e_{\sqrt{m}+1} & \sqrt{m}+2 < i < 2\sqrt{m} \end{cases} \qquad g_3(i) = \begin{cases} 1 & i = 0 \\ \tau & 1 \le i < \sqrt{m} \\ 1 & i = \sqrt{m} \\ \tau & i = \sqrt{m}+1 \\ e_{i-1} & i = \sqrt{m}+2 \\ e_{i-1} & \sqrt{m}+2 < i < 2\sqrt{m} \end{cases}$$

and $F_{\mathsf{PC}}(y_1, y_2, y_3) = y_1 - y_2 \cdot y_3$. Note that g_1, g_2, and g_3 can be padded with evaluations to zero to any desired length. We let $\mathsf{ZC}_{\mathsf{PC}}$ denote the zero-check relation with $(F_{\mathsf{PC}}, G_{\mathsf{PC}})$ fixed for the structure. Similarly, we let $\mathsf{NSC}_{\mathsf{PC}}$ denote the nested sum-check relation with $(F_{\mathsf{PC}}, G_{\mathsf{PC}})$ fixed for the structure.

Putting everything together, given any number of zero-check instances, and any number of power-check instances (represented as zero-check instances), the verifier first samples a challenge τ for which the prover responds with an (unverified) commitment $\overline{e}$ to the powers of τ. Then, the zero-check instances, and similarly the power-check instances, can be reduced to the corresponding nested sum-check instances over the corresponding structures with respect to the challenge τ. Alongside, the verifier outputs a new power-check instance $(\overline{e}, \tau)$ to be checked (in the future). The following construction formally captures this aggregate reduction.

Construction 3 (Zero-check reduction). We construct a reduction of knowledge of type

$$\mathsf{ZC}^n \times \mathsf{ZC}^m_{\mathsf{PC}} \to \mathsf{NSC}^n \times \mathsf{NSC}^m_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}}.$$

Consider size bounds $\ell, d, t \in \mathbb{N}$. Consider an arbitrary generator algorithm, an arbitrary encoder algorithm that outputs its input structure and an arbitrary underlying additively homomorphic commitment scheme. We define the prover and the verifier as follows.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}}, \vec{\mathsf{u}}_{\mathsf{pc}}), (\vec{\mathsf{w}}, \vec{\mathsf{w}}_{\mathsf{pc}}))}$:

1. $\mathcal{V}$: Sample and send $\tau \xleftarrow{\$} \mathbb{F}$.
2. $\mathcal{P}$: Compute $((\overline{e}, \tau), e) \in \mathsf{ZC}_{\mathsf{PC}}$ with size parameter $m = 2^{\ell/2}$ and send $\overline{e}$.
3. $\mathcal{P}, \mathcal{V}$: Output the following instance-witness pairs (the verifier only outputs the instances).

$$\Big(\{(0, \mathsf{u}_i, \overline{e})\}_{i\in[n]}, \{(\mathsf{w}_i, e)\}_{i\in[n]}\Big) \in \mathsf{NSC}^n$$
$$\Big(\{(0, \mathsf{u}_{\mathsf{pc},j}, \overline{e})\}_{j\in[m]}, \{(\mathsf{w}_{\mathsf{pc},j}, e)\}_{j\in[m]}\Big) \in \mathsf{NSC}^m_{\mathsf{PC}}$$
$$\Big((\overline{e}, \tau), e\Big) \in \mathsf{ZC}_{\mathsf{PC}}$$

We formally prove the following lemma in Appendix B.2.

Lemma 5 (Zero-check reduction). *Construction 3 is a reduction of knowledge of type* $\mathsf{ZC}^n \times \mathsf{ZC}^m_{\mathsf{PC}} \to \mathsf{NSC}^n \times \mathsf{NSC}^m_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}}$ *with* 1 *round, a communication complexity of* 1 *field element and* 1 *element in the message space of the commitment scheme over size* $2^{\ell/2+1}$ *vectors, an* $O(2^{\ell/2+1})$ *prover time complexity, and an* $O(1)$ *verifier time complexity.*

Proof (Intuition). We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct PPT extractor χ that outputs a satisfying input witness given a tree of accepting transcripts and corresponding output instance-witness pairs in $\mathsf{NSC}^n \times \mathsf{NSC}^m_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}}$.

Indeed, in such a tree, consider an output satisfying NSC instance-witness pair

$$((0, \mathsf{u}_i, \overline{e}), (\mathsf{w}_i, e)) \in \mathsf{NSC}.$$

By the satisfiability condition of NSC, for $(\vec{u}_i, \mathsf{x}_i) \leftarrow \mathsf{u}_i$ and $\vec{w}_i \leftarrow \mathsf{w}_i$ we have that

$$0 = \sum_{x\in\{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(G(\vec{w}_i, \mathsf{x}_i)).$$

But from the fact that e is a satisfying $\mathsf{ZC}_{\mathsf{PC}}$ instance, it must contain powers of some $\tau \in \mathbb{F}$. Substituting, we have that

$$0 = \sum_{x \in \{0,1\}^\ell} \tau^X \cdot F(G(\vec{w}_i, \mathsf{x}_i)(x)),$$

where X is the corresponding decimal representation if x is treated as the bit representation.

By the binding property of the commitment scheme, and the verifier's construction, we must have that $\vec{w}_i$ and x_i remain identical across all transcripts. Then, with enough such transcripts, by interpolation we have that

$$0 = F(G(\vec{w}_i, \mathsf{x}_i)(x))$$

for all $x \in \{0,1\}^\ell$. But, this means that

$$(\mathsf{u}_i, \mathsf{w}_i) \in \mathsf{ZC}.$$

By an identical line of reasoning χ can produce satisfying witnesses for the input $\mathsf{ZC}_{\mathsf{PC}}$ instances as well. □

4.2 A folding scheme for the nested sum-check relation

Given the reductions in the prior subsection, our goal is to fold a set of nested sum-check instances over the original structure (F, G), and a set of nested sum-check instances over the power-check structure $(F_{\mathsf{PC}}, F_{\mathsf{PC}})$. This involves recasting nested sum-check instances as sum-check instances.

Indeed, as stated earlier, one technicality is that unlike the sum-check relation, which performs a check of the form $T = \sum_x F(G(\vec{w}, \mathsf{x})(x))$, the nested sum-check relation performs a more structured check of the form:

$$T = \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(G(\vec{w}, \mathsf{x})(x)).$$

The nested sum-check instance can be reinterpreted as a sum-check instance by first defining polynomials h_1 and h_2, which extend the domain of e_1 and e_2 to $\{0,1\}^\ell$ as follows

$$h_1(x) = e_1(x_1) \cdot \widetilde{(1, 1, \ldots, 1)}(x_2) = e_1(x_1)$$
$$h_2(x) = \widetilde{(1, 1, \ldots, 1)}(x_1) \cdot e_2(x_2) = e_2(x_2)$$

Then, we can define the structure G' to additionally produce h_1 and h_2 given an additional vector e in the nested sum-check witness

$$G'((\vec{w}, e), \mathsf{x}) = (G(\vec{w}, \mathsf{x}), h_1, h_2)$$

Crucially, we have that h_1 and h_2 are produced linearly from e, which is stipulated by the sum-check relation. Then, we can define the polynomial F' to account for these additionally produced polynomials

$$F'(\vec{g}(x), h_1(x), h_2(x)) = h_1(x) \cdot h_2(x) \cdot F(\vec{g}(x))$$

Then, we have that

$$\begin{aligned} T &= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(G(\vec{w}, \mathsf{x})(x)) \\ &= \sum_{x \in \{0,1\}^\ell} F'(G'((\vec{w}, e), \mathsf{x})(x)) \end{aligned}$$

Hence, the prover and the verifier can fold a set of nested sum-check instances over structure (F, G) by using $\mathsf{SumFold}$ to fold equivalent sum-check instances with respect to structure (F', G').

We can similarly define the structure $(F'_{\mathsf{PC}}, G'_{\mathsf{PC}})$ to fold the nested sum-check instances over the power-check structure. At this point the prover and the verifier can run two independent invocations of $\mathsf{SumFold}$, once to fold over structure (F', G') and once to fold over structure $(F'_{\mathsf{PC}}, G'_{\mathsf{PC}})$. Alternatively, for concrete efficiency, the prover and the verifier can run a single invocation of $\mathsf{SumFold}$ by taking a random linear combination of the original structures. In particular, consider two sets of nested sum-check instance-witness pairs

$$\begin{aligned} \{(T_i, (\vec{u}_i, \overline{e}_i, \mathsf{x}_i), (\vec{w}_i, e_i))\}_{i \in [n]} &\in \mathsf{NSC}^n \\ \{(T_{\mathsf{pc},i}, (\vec{u}_{\mathsf{pc},i}, \overline{e}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}), (\vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}))\}_{i \in [n]} &\in \mathsf{NSC}^n_{\mathsf{PC}} \end{aligned}$$

to be checked over structures (F, G) and $(F_{\mathsf{PC}}, G_{\mathsf{PC}})$ respectively. The verifier begins by sending a challenge $\gamma \in \mathbb{F}$. Then, for each $i \in [n]$, the prover and the verifier can reduce to checking that

$$T_i + \gamma \cdot T_{\mathsf{pc},i} = \sum_{x \in \{0,1\}^\ell} F'(G'((\vec{w}_i, e_i), \mathsf{x}_i)(x)) + \gamma \cdot F'_{\mathsf{PC}}(G'_{\mathsf{PC}}((\vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}), \mathsf{x}_{\mathsf{pc},i})(x))$$

Once again, we can recast the above instance to a sum-check instance by defining the following structure:

$$\begin{aligned} F''(\vec{g}, \vec{g}_{\mathsf{pc}}) &= F(\vec{g}) + \gamma \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}) \\ G''(\vec{w}, e, \vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}, (\mathsf{x}, \mathsf{x}_{\mathsf{pc}})) &= (G'((\vec{w}, e), \mathsf{x}), G'_{\mathsf{PC}}((\vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}), \mathsf{x}_{\mathsf{pc}})). \end{aligned}$$

Then, for the aggregated instance-witness pair

$$(T_i + \gamma \cdot T_{\mathsf{pc},i}, (\vec{u}_i, \overline{e}_i, \vec{u}_{\mathsf{pc},i}, \overline{e}_{\mathsf{pc},i}, (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i})), (\vec{w}_i, e_i, \vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i})),$$

the verifier can equivalently check

$$T_i + \gamma \cdot T_{\mathsf{pc},i} = \sum_{x \in \{0,1\}^\ell} F''(G''((\vec{w}_i, e_i, \vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i}))(x)) \tag{6}$$

for all $i \in [n]$. Note that because we are pairing off NSC and $\mathsf{NSC}_{\mathsf{PC}}$ instances, the above optimization specifically requires the same number of instances from each relation. This is not a problem in practice as we can always pad either side with trivially satisfying instance-witness pairs.

Then, the prover and the verifier can run $\mathsf{SumFold}$ over all the instance-witness pairs in Equation (6) to reduce to checking a single instance

$$\begin{aligned} T_\gamma &= \sum_{x \in \{0,1\}^\ell} F''(G''((\vec{w}, e, \vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc}}))(x)) \\ &= \sum_{x \in \{0,1\}^\ell} F'(G'((\vec{w}, e), \mathsf{x})(x)) + \gamma \cdot F'_{\mathsf{PC}}(G'_{\mathsf{PC}}((\vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}), \mathsf{x}_{\mathsf{pc}})(x)) \end{aligned}$$

At this point, the prover can send

$$T = \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(G(\vec{w}, \mathsf{x})(x)) \tag{7}$$

$$T_{\mathsf{pc}} = \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{\mathsf{pc},1}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2}(x_2) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}})(x)). \tag{8}$$

The verifier checks that indeed $T_\gamma = T + \gamma \cdot T_{\mathsf{pc}}$, and reduces to checking Equations (7) and (8). But, by definition, this is precisely equivalent to checking an instance in NSC and an instance in $\mathsf{NSC}_{\mathsf{PC}}$. Hence, the above interaction gives us a reduction of knowledge from $\mathsf{NSC}^n \times \mathsf{NSC}^n_{\mathsf{PC}}$ to $\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}$. We formally describe this reduction below.

Construction 4 (Folding nested sum-check). We construct a reduction of knowledge of type

$$\mathsf{NSC}^n \times \mathsf{NSC}^n_{\mathsf{PC}} \to \mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}.$$

Indeed, let $(\mathcal{G}_{\mathsf{SF}}, \mathcal{K}_{\mathsf{SF}}, \mathcal{P}_{\mathsf{SF}}, \mathcal{V}_{\mathsf{SF}})$ be $\mathsf{SumFold}$ (Construction 1). Consider size bounds $\ell, d, t \in \mathbb{N}$. Consider an arbitrary generator algorithm and an underlying additively homomorphic commitment scheme. We define the encoder, the prover, and the verifier as follows.

$\underline{\mathcal{K}(\mathsf{pp}, (F, G))}$: Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, G), \perp)$ and structure (F, G).

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \{((T_i, \mathsf{u}_i, \mathsf{x}_i), (T_{\mathsf{pc},i}, \mathsf{u}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}))\}_{i \in [n]}, \{(\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i})\}_{i \in [n]})}$:

1. $\mathcal{V}$: Sample and send $\gamma \xleftarrow{\$} \mathbb{F}$.
2. $\mathcal{P}$: Compute the updated prover key $\mathsf{pk}' \leftarrow (F', G')$ where

$$\begin{aligned} F'(\vec{g}, h_1, h_2, \vec{g}_{\mathsf{pc}}, h_{\mathsf{pc},1}, h_{\mathsf{pc},2}) &= h_1 \cdot h_2 \cdot F(\vec{g}) + \gamma \cdot h_{\mathsf{pc},1} \cdot h_{\mathsf{pc},2} \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}) \\ G'(\vec{w}, e, \vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}, (\mathsf{x}, \mathsf{x}_{\mathsf{pc}})) &= (G(\vec{w}, \mathsf{x}), h_1, h_2, G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}}), h_{\mathsf{pc},1}, h_{\mathsf{pc},2}) \end{aligned}$$

   where $(F_{\mathsf{PC}}, G_{\mathsf{PC}})$ are defined as in Construction 2 with an appropriate padding, and for $j \in \{1, 2\}$, $h_j(x) = \widetilde{e}_j(x_j)$ and $h_{\mathsf{pc},j}(x) = \widetilde{e}_{\mathsf{pc},j}(x_j)$, where

(e_1, e_2), $(e_{\mathsf{pc},1}, e_{\mathsf{pc},2})$, and (x_1, x_2) represent the first and second half of the original vector.

3. $\mathcal{P}, \mathcal{V}$: Run the sum-check folding reduction $\langle \mathcal{P}_{\mathsf{SF}}, \mathcal{V}_{\mathsf{SF}} \rangle$ on the updated prover and verifier keys $(\mathsf{pk}', \mathsf{vk})$, and the updated instance-witness pairs for $i \in [n]$

$$\left((T_i + \gamma \cdot T_{\mathsf{pc},i}, (\mathsf{u}_i, \mathsf{u}_{\mathsf{pc},i}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i})\right), (\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i}))$$

to produce a folded instance-witness pair $(T_\gamma, (\mathsf{u}, \mathsf{u}_{\mathsf{pc}}), (\mathsf{x}, \mathsf{x}_{\mathsf{pc}})), (\mathsf{w}, \mathsf{w}_{\mathsf{pc}})$.

4. $\mathcal{P}$: Parse $(\vec{w}, e) \leftarrow \mathsf{w}$, $(\vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}) \leftarrow \mathsf{w}_{\mathsf{pc}}$, compute $\vec{g} \leftarrow G(\vec{w}, \mathsf{x})$, $\vec{g}_{\mathsf{pc}} \leftarrow G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}})$, and send to $\mathcal{V}$

$$T \leftarrow \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(\vec{g}(x))$$
$$T_{\mathsf{pc}} \leftarrow \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{\mathsf{pc},1}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2}(x_2) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}(x)).$$

5. $\mathcal{V}$: Check that $T_\gamma = T + \gamma \cdot T_{\mathsf{pc}}$.

6. $\mathcal{P}, \mathcal{V}$: Output the following instance-witness pairs (the verifier only outputs the instances).

$$((T, \mathsf{u}, \mathsf{x}), \mathsf{w}) \in \mathsf{NSC}$$
$$((T_{\mathsf{pc}}, \mathsf{u}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}}), \mathsf{w}_{\mathsf{pc}}) \in \mathsf{NSC}_{\mathsf{PC}}$$

We formally prove the following lemma in Appendix B.3.

Lemma 6 (Folding nested sum-check). *Construction 4 is a reduction of knowledge of type* $\mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n \to \mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}$ *with* $2 + \log n$ *rounds, a communication complexity of* $O(d \log n)$ *field elements, a prover time complexity of* $O(ntd \cdot 2^\ell)$ *field operations, and a verifier time complexity of* $O(d \log n)$ *field operations.*

Proof (Intuition). We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct PPT extractor χ that outputs a satisfying input witness given a tree of accepting transcripts and corresponding output instance-witness pairs in $\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}$. We do so by leveraging the tree-extractor χ_{FSC} guaranteed by the tree extractability of $\mathsf{SumFold}$ (Lemma 17).

In particular, χ can produce a sufficient number of accepting sub-trees for the underlying $\mathsf{SumFold}$ by using the provided instance witness pairs in $\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}$ to solve for corresponding satisfying output instance-witness pairs in SC. Then, $\mathsf{SumFold}$'s tree extractor χ_{FSC} can produce a satisfying input witness in SC^n for each sub-tree. By the binding property of the commitment scheme, these witnesses must all be identical. Then, by interpolating over the verifier's initial challenge, this means that the witness produced by χ_{FSC} must also be a satisfying witness for the input relation $\mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n$. □

4.3 ZeroFold: Putting everything together

So far, we have constructed the zero-check reduction $\Pi_{\mathsf{ZCR}} : \mathsf{ZC}^n \times \mathsf{ZC}_{\mathsf{PC}}^n \to \mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n \times \mathsf{ZC}_{\mathsf{PC}}$ (Construction 3) and a folding scheme for the nested sum-check relation $\Pi_{\mathsf{FNSC}} : \mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n \to \mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}}$ (Construction 4). We can use these two reductions to produce a folding scheme for ZC with a running instance type of $\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}}$

Indeed, suppose that a verifier would like to check n zero-check instances $\vec{\mathsf{u}} = (\mathsf{u}_1, \ldots, \mathsf{u}_n)$ and n running instances $\vec{\mathsf{U}} = (\mathsf{U}_1, \ldots, \mathsf{U}_n)$ in $\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}}$. The verifier first collects n instances in $\mathsf{ZC}_{\mathsf{PC}}$ from $\vec{\mathsf{U}}$ and reduces these instances along with $\vec{\mathsf{u}}$ via the zero-check reduction Π_{ZCR} into n instances in NSC, n instances $\mathsf{NSC}_{\mathsf{PC}}$, and a fresh instance in $\mathsf{ZC}_{\mathsf{PC}}$. Now the verifier is left with $2n$ instances in NSC and $2n$ instances in $\mathsf{NSC}_{\mathsf{PC}}$ (half from the original n running instances and half freshly generated by Π_{ZCR}). The verifier reduces all these instances via the folding scheme for the nested sum-check relation Π_{FNSC} into a single instance in NSC and a single instance in $\mathsf{NSC}_{\mathsf{PC}}$. The verifier concludes by outputting these two instances alongside the fresh instance in $\mathsf{ZC}_{\mathsf{PC}}$ generated by Π_{ZCR} as the folded instance.

We formally capture this construction in the following theorem. We let $1_{\mathcal{R}}$ denote the identity reduction (i.e. the prover and the verifier output their inputs) for the relation $\mathcal{R}$. In Appendix 4.4, we discuss how to reduce the number of rounds to $1 + \log n$.

Theorem 3 (ZeroFold). *Given reductions of knowledge*

$$\begin{aligned} \Pi_{\mathsf{ZCR}} &: \mathsf{ZC}^n \times \mathsf{ZC}_{\mathsf{PC}}^n \to \mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n \times \mathsf{ZC}_{\mathsf{PC}} && \textit{(Construction 3)}, \\ \Pi_{\mathsf{FNSC}} &: \mathsf{NSC}^{2n} \times \mathsf{NSC}_{\mathsf{PC}}^{2n} \to \mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} && \textit{(Construction 4)} \end{aligned}$$

we have that

$$(\Pi_{\mathsf{FNSC}} \times 1_{\mathsf{ZC}_{\mathsf{PC}}}) \circ (1_{\mathsf{NSC}^n \times \mathsf{NSC}_{\mathsf{PC}}^n} \times \Pi_{\mathsf{ZCR}})$$

is a reduction of knowledge of type

$$(\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}})^n \times \mathsf{ZC}^n \to (\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}})$$

with $3+\log n$ *rounds, a communication complexity of* $O(d \log n)$ *field elements and* 1 *element in the message space of the commitment scheme over size* $2^{\ell/2+1}$ *vectors, a prover time complexity of* $O(ntd \cdot 2^\ell)$ *field operations and a commitment over a size* $2^{\ell/2+1}$ *vector, and a verifier time complexity of* $O(d \log n)$ *field operations.*

4.4 Optimizing the round complexity of Theorem 3

We observe that the zero-check folding scheme can be optimized down to $1 + \log n$ rounds of communication: First, as the commitment $\overline{e}$ to the powers of τ in Π_{ZCR} is checked independently of the interaction at a later point, the prover is free to send it as part of its first message in Π_{FNSC}. This allows the verifier to send the randomness τ in Π_{ZCR}, the randomness γ in Π_{NSC}, and the randomness ρ in the

first round of SumFold all in one message, removing a round of communication. Second, in Π_{NSC}, instead of sending T_γ at the end of the underlying sum-check protocol and then again T and T_{pc}. the prover can directly send T and T_{pc} (in place of T_γ) in the final round of the underlying sum-check protocol. This, once again, removes a round of communication. In the case of folding a single zero-check instance with a single running instance, this brings the overall number of rounds to 2.

5 Folding relations targeting zero-check

In this section, we design reductions of knowledge [KP23] from various relations to zero-check, thereby demonstrating that these relations can also be efficiently folded. Concretely, we reduce an NP-complete relation, CCS [STW23], the grand-product relation, and the lookup relation to zero-check. In all these cases, the reductions are simple, constant-round, and involve minimal verifier work.

To tie together the results of this section, we start with a natural corollary of Theorem 3, which formally states that if there exists a reduction of knowledge from a relation $\mathcal{R}$ to any number of zero-check instances, then we can fold instances in $\mathcal{R}$ by first reducing them to instances in the zero-check relation and then applying ZeroFold, NeutronNova's folding scheme for zero-check.

Corollary 1 (Folding relations targeting zero-check). *Let Π_{FZC} represent the zero-check folding scheme from Theorem 3. Given a reduction of knowledge $\Pi : \mathcal{R} \to \mathsf{ZC}^m$ we have that*

$$\Pi_{\mathsf{FZC}} \circ (1_{(\mathsf{NSC}\times\mathsf{NSC}_{\mathsf{PC}}\times\mathsf{ZC}_{\mathsf{PC}})^{nm}} \times \Pi^n)$$

is a reduction of knowledge of type

$$(\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}})^{nm} \times \mathcal{R}^n \to (\mathsf{NSC} \times \mathsf{NSC}_{\mathsf{PC}} \times \mathsf{ZC}_{\mathsf{PC}})$$

where the round complexity, communication complexity, prover time complexity and verifier time complexity is the combined total of running Π for n times and Π_{FZC} over nm zero-check instances.

5.1 Reducing CCS to zero-check

We first reduce the customizable constraint system (CCS) [STW23], to zero-check without any interaction or work by the verifier. Since CCS is an NP-complete language that simultaneously generalizes R1CS [GGPR13], AIR [BSBHR19], and Plonkish [GWC19], we obtain a folding scheme for all of these relations whose efficiency matches the efficiency of our folding scheme for zero-check.

Recall that CCS checks arbitrary degree constraints represented using selector matrices $M_1, \ldots, M_t$ over z (which is the concatenation of a witness vector w, a public vector x, and a constant of 1) by first computing the matrix-vector products $M_j \cdot w$ for all $j \in [t]$ and then checking a sum of Hadamard products over the resulting vectors. Formally, CCS is defined as follows.

Definition 10 (CCS [STW23]). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size bounds* $m, n, N, \ell, t, q, d \in \mathbb{N}$ *where* $n > \ell$*. We define the customizable constraint system (CCS) relation,* CCS*, over structure, instance, witness tuples as follows.*

$$\mathsf{CCS} = \left\{ (\mathsf{pp}, (\vec{M}, \vec{S}, \vec{c}), (\overline{w}, \mathsf{x}), w) \;\middle|\; \begin{array}{l} M \in (\mathbb{F}^{m \times n})^t, S_i \subseteq [t] \text{ for } i \in [q], \vec{c} \in \mathbb{F}^q \\ \mathsf{x} \in \mathbb{F}^\ell, w \in \mathbb{F}^{m-\ell-1}, \\ \mathsf{Commit}(\mathsf{pp}, w) = \overline{w}, \\ \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} M_j z = 0 \end{array} \right\}$$

where $z = (w, \mathsf{x}, 1)$ *and each matrix* M_i *contains at most* $\Omega(m)$ *non-zero entries.*

We reduce CCS to zero-check as follows: we design an inner zero-check structure polynomial $G_{\vec{M}}$ that encodes the CCS selector matrices $M_1, \ldots, M_t$. Specifically, on input witness vector w and a public vector x, $G_{\vec{M}}$ outputs the multilinear extensions of $M_1 z, \ldots, M_t z$, where $z = (w, \mathsf{x}, 1)$. Then, for each row of entries in these vectors, we design an outer zero-check structure polynomial $F_{\vec{S},\vec{c}}$ that takes a sum of products exactly as the original CCS structure dictates. Formally we have the following reduction from CCS to zero-check.

Lemma 7 (From CCS to zero-check [STW23]). *Consider size bounds* m*,* n*,* N*,* ℓ*,* t*,* q*, and* d *where* $n > \ell$*. We have that* $(\mathsf{pp}, (\vec{M}, \vec{S}, \vec{c}), (\overline{w}, \mathsf{x}), w) \in \mathsf{CCS}$ *if and only if* $(\mathsf{pp}, (F, G), (\overline{w}, \mathsf{x}), w) \in \mathsf{ZC}$ *where for* $z = (w, \mathsf{x}, 1)$

$$F_{\vec{S},\vec{c}}(m_1, \ldots, m_t) = \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} m_j$$

$$G_{\vec{M}}(w, \mathsf{x}) = (\widetilde{(M_1 z)}, \ldots, \widetilde{(M_t z)}).$$

This naturally induces an equivalent reduction *of knowledge* (with no interaction) by defining the encoder, prover, and verifier to map the structure, instance, and witness identically. Then, by Corollary 1, we have a folding scheme for CCS.

5.2 Reducing grand-product to zero-check

In this section, we reduce the grand-product relation to the zero-check relation, thereby providing a folding scheme for grand-products. We rely on results from Lasso [STW24], which itself builds on its predecessors [Set20,SL20].

Recall that the grand-product relation checks that taking the product of elements of a committed vector v results in some prescribed value p.

Definition 11 (Grand-product relation). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size bound* $n, m \in \mathbb{N}$*. We define the grand-product relation* GP *over public parameter, instance, witness tuples as follows*

$$\mathsf{GP} = \left\{ (\mathsf{pp}, (p, \overline{v}), v) \;\middle|\; \begin{array}{l} v \in \mathbb{F}^n \\ \mathsf{Commit}(\mathsf{pp}, v) = \overline{v}, \\ v = \prod_{i \in [m]} v_i \end{array} \right\}.$$

Lasso provides two options for *proving* grand products: (1) the approach presented in Section 5.2, which in their setting is then proven with the sum-check protocol; and (2) the protocol of Thaler [Tha13], a specialization of GKR proof system [GKR08]. The main trade-off is that the first option requires committing to an additional vector of size N, where N us the number of entries in v whereas the second protocol requires logarithmic number of invocations of the sum-check protocol, resulting in a proof length of $O(\log^2 N)$. To mitigate this trade-off, Setty and Lee [SL20, §6] also describe a hybrid solution where their grand product argument is applied on an instance of size $N/2^k$ (for some chosen parameter k) and Thaler's protocol is applied to reduce a claim about that instance to a multilinear evaluation claim about $\widetilde{v}$. For simplicity, we focus on option (1). However, both option (2) and the hybrid solution induce alternative compatible reductions from grand-product to zero-check.

At a high level, Setty and Lee consider a new polynomial f, which contains the original vector v as well as all the intermediate products computed in a tree-like fashion. Then, the relationship between v and the final claimed product p can be checked using quadratic constraints.

Lemma 8 (Grand-product arithmetization [SL20]). *Consider $p \in \mathbb{F}$ and $v \in \mathbb{F}^n$. There exists an efficiently computable $f \in \mathbb{F}[X_1, \ldots, X_{\log n+1}]$ such that $p = \prod_{i\in[n]} v_i$ if and only if $f(0, x) = \widetilde{v}(x)$, $f(1, \ldots, 1, 0) = p$ and*

$$f(1, x) = f(x, 0) \cdot f(x, 1) \tag{9}$$

for all $x \in \{0, 1\}^{\log n}$.

In light of this arithmetization, we get a natural reduction from grand-product to zero-check: The prover begins by committing to an intermediate product vector v' which is the portion of f not including v. Together v and v' are treated as the new zero-check witness and a vector with a single entry of p is treated as the public vector. Then, we design an inner zero-check structure polynomial G_{GP}, which produces polynomials $g_1(x) = f(1, x)$, $g_2(x) = f(x, 0)$, and $g_3(x) = f(x, 1)$ by selecting elements from v, v', and p. For each evaluation of these polynomials we design an outer zero-check structure polynomial F_{GP} that computes Equation 9. Formally, we have the following reduction.

Construction 5 (From grand-product to zero-check). We construct a reduction of knowledge of type

$$\mathsf{GP} \to \mathsf{ZC}.$$

Consider size bound $n \in \mathbb{N}$. Let $(\mathsf{Gen}, \mathsf{Commit})$ denote an arbitrary additively homomorphic vector commitment scheme. We define the generator, encoder, prover, and verifier as follows.

$\underline{\mathcal{G}(\lambda, n)}$: Output $\mathsf{pp} \xleftarrow{\$} \mathsf{Gen}(\lambda, n)$.

$\underline{\mathcal{K}(\mathsf{pp})}$:

1. Define $F_{\mathsf{GP}}(g_1, g_2, g_3) = g_1 - g_2 \cdot g_3$ and define G_{GP} on input $((v, v'), p)$ to output

$$g_1(x) \leftarrow f(1, x)$$
$$g_2(x) \leftarrow f(x, 0)$$
$$g_3(x) \leftarrow f(x, 1)$$

   where f is the multilinear extension of $(v, (v'_1, \ldots, v'_{n-2}, p, v'_n))$.

2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow (\mathsf{pp}, \perp)$ and the updated zero-check structure $(F_{\mathsf{GP}}, G_{\mathsf{GP}})$.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (p, \overline{v}), v)}$:

1. $\mathcal{P}$: Compute a multilinear polynomial $\widetilde{f}$ as guaranteed by Lemma 8. Define $v' \in \mathbb{F}^n$ such that $\widetilde{v}' = \widetilde{f}(1, x)$ and send a commitment $\overline{v}' \leftarrow \mathsf{Commit}(\mathsf{pp}, v')$.
2. $\mathcal{P}, \mathcal{V}$: Produce the following zero-check instance-witness pair (the verifier only outputs the instance)

$$(((\overline{v}, \overline{v}'), p), (v, v')) \in \mathsf{ZC}.$$

We formally prove the following lemma in Appendix B.5, which, by Corollary 1 implies a folding scheme for GP.

Lemma 9 (From grand-product to zero-check). *Construction 5 is an RoK of type $\mathsf{GP} \to \mathsf{ZC}$ with a single round, a communication complexity of one element in the message space of the commitment scheme for vectors of size n. The prover's time complexity is $O(n)$ and the verifier's time complexity is $O(1)$.*

5.3 Reducing lookups to grand-product

In this section, we reduce a lookup instance to several grand product instances, which, from the previous subsection, can be reduced to zero-check instances. Hence, we get a folding scheme for the lookup relation.

Our starting point is Lasso [STW24], a state-of-the-art lookup argument. Lasso supports unstructured tables (for which the verifier holds a commitment) as well as very large Spark-only structured (SOS) tables that do not need to be materialized either by the prover nor the verifier. Crucially, Lasso's lookup argument for very large structured tables of size N is obtained by essentially running c copies of the lookup argument for unstructured tables on tables of size $N^{1/c}$ and then assembling results of sub-table lookups (e.g., with CCS). A similar observation appears in prior work [DP23]. For this reason, the rest of the section focuses on lookups over unstructured tables, but our results naturally generalize to Lasso's SOS tables.

Recall that the lookup relation checks that a committed vector v only consists of elements found in a committed table t. Lasso considers a more general form, which they refer to as indexed lookups, where an additional committed vector

a indicates which index of t each entry of v can be found. All lookups used in Jolt [AST24] are indexed lookups, so we focus on this general version.

Definition 12 (Lookup relation [STW24]). *Let* $(\mathsf{Gen}, \mathsf{Commit})$ *denote an additively-homomorphic commitment scheme for vectors over finite field* $\mathbb{F}$*. Consider size parameters* $n, m \in \mathbb{N}$*. We define the lookup relation* LKP *over public parameter, instance, witness tuples as follows*

$$\mathsf{LKP} = \left\{ (\mathsf{pp}, (\overline{t}, \overline{a}, \overline{v}), (t, a, v)) \;\middle|\; \begin{array}{l} v, a \in \mathbb{F}^m, t \in \mathbb{F}^n, \\ \forall i \in [m].\ v_i = t_{a_i}, \\ \mathsf{Commit}(\mathsf{pp}, t) = \overline{t}, \\ \mathsf{Commit}(\mathsf{pp}, a) = \overline{a}, \\ \mathsf{Commit}(\mathsf{pp}, v) = \overline{v}, \end{array} \right\}.$$

Instead of directly proving the above relation, Lasso reformulates the lookup relation as the following bivariate polynomial identity test. This bivariate identity is implicit in [STW24, Claim 3].

Lemma 10 (Lookup arithmetization [Set20,STW24]). *Consider vectors* $v, a \in \mathbb{F}^m$ *and* $t \in \mathbb{F}^n$*. There exist efficiently computable vectors* $c \in \mathbb{F}^m$ *and* $f \in \mathbb{F}^n$ *such that* $v_i = t_{a_i}$ *for all* $i \in [m]$ *if and only if*

$$\begin{aligned} &\left(\prod_{i\in[n]} (i + t_i \cdot X - Y)\right) \cdot \left(\prod_{i\in[m]} (a_i + v_i \cdot X + (c_i + 1) \cdot X^2 - Y)\right) = \\ &\left(\prod_{i\in[m]} (a_i + v_i \cdot X + c_i \cdot X^2 - Y)\right) \cdot \left(\prod_{i\in[n]} (i + t_i \cdot X + f_i \cdot X^2 - Y)\right). \end{aligned} \tag{10}$$

Checking Equation (10) can be reduced to checking four grand product instances (Definition 11). In particular, the prover first computes c and f as guaranteed by Lemma 10 and send the corresponding commitments $\overline{c}$ and $\overline{f}$. The verifier samples and sends challenges $r_X \in \mathbb{F}$ and $r_Y \in \mathbb{F}$ to check Equation (10) at a single random point. By the Schwartz-Zippel lemma, this implies checking the original polynomial identity with overwhelming probability. The prover then sends claimed products p_1, p_2, p_3, and p_4 for each of the grand products in Equation (10) The verifier checks that $p_1 \cdot p_2 = p_3 \cdot p_4$, and then homomorphically compute and output commitments to each of the inner vectors to be checked in constant time. Formally, we have the following reduction.

Construction 6 (From Lookup to grand-product). We construct a reduction of knowledge of type

$$\mathsf{LKP} \to \mathsf{GP}^4.$$

Consider size bounds $n, m \in \mathbb{N}$. Let $(\mathsf{Gen}, \mathsf{Commit})$ denote an arbitrary additively homomorphic commitment scheme. We define the generator, encoder, prover,

and verifier as follows.

$\underline{\mathcal{G}(\lambda, (n, m))}$: Output $\mathsf{pp} \xleftarrow{\$} \mathsf{Gen}(\lambda, \max(n, m))$.

$\underline{\mathcal{K}(\mathsf{pp})}$:

1. Compute vector commitments

$$\begin{aligned} G_n &\leftarrow \mathsf{Commit}(\mathsf{pp}, 1^n) \\ G_m &\leftarrow \mathsf{Commit}(\mathsf{pp}, 1^m) \\ H &\leftarrow \mathsf{Commit}(\mathsf{pp}, (1, \ldots, n)). \end{aligned}$$

   $G_n \leftarrow \mathsf{Commit}(\mathsf{pp}, 1^n)$, $G_m \leftarrow \mathsf{Commit}(\mathsf{pp}, 1^m)$, $H \leftarrow \mathsf{Commit}(\mathsf{pp}, (1, \ldots, n))$.

2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pp}, (G_n, G_m, H)), (G_n, G_m, H))$.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\overline{t}, \overline{a}, \overline{v}), (t, a, v))}$:

1. $\mathcal{P}$: Compute $c \in \mathbb{F}^m$ and $f \in \mathbb{F}^n$ as guaranteed by Lemma 10 and send $\overline{c} \leftarrow \mathsf{Commit}(\mathsf{pp}, c)$ and $\overline{f} \leftarrow \mathsf{Commit}(\mathsf{pp}, f)$.
2. $\mathcal{V}$: Send random challenges $r_X \in \mathbb{F}$ and $r_Y \in \mathbb{F}$.
3. $\mathcal{P}$: Compute the underlying grand-product witnesses

$$\begin{aligned} v_1 &\leftarrow \{(i + t_i \cdot r_X - r_Y)\}_{i \in [n]} \\ v_2 &\leftarrow \{(a_i + v_i \cdot r_X + (c_i + 1) \cdot r_X^2 - r_Y)\}_{i \in [m]} \\ v_3 &\leftarrow \{(a_i + v_i \cdot r_X + c_i \cdot r_X^2 - r_Y)\}_{i \in [m]} \\ v_4 &\leftarrow \{(i + t_i \cdot r_X + f_i \cdot r_X^2 - r_Y)\}_{i \in [n]}. \end{aligned}$$

4. $\mathcal{P}$: Compute and send the claimed products $p_k \leftarrow \prod_i v_{k,i}$ for $k \in \{1, 2, 3, 4\}$.
5. $\mathcal{V}$: Check that $p_1 \cdot p_2 = p_3 \cdot p_4$.
6. $\mathcal{P}, \mathcal{V}$: Compute the corresponding vector commitments

$$\begin{aligned} \overline{v}_1 &\leftarrow H + \overline{t} \cdot r_X - G_n \cdot r_Y \\ \overline{v}_2 &\leftarrow \overline{a} + \overline{v} \cdot r_X + (\overline{c} + G_m) \cdot r_X^2 - G_m \cdot r_Y \\ \overline{v}_3 &\leftarrow \overline{a} + \overline{v} \cdot r_X + \overline{c} \cdot r_X^2 - G_m \cdot r_Y \\ \overline{v}_4 &\leftarrow H + \overline{t} \cdot r_X + \overline{f} \cdot r_X^2 - G_n \cdot r_Y. \end{aligned}$$

7. $\mathcal{P}, \mathcal{V}$: Output the following grand-product instance-witness pairs (the verifier only outputs instances)

$$\{((p_k, \overline{v}_k), v_k)\}_{k \in [4]} \in \mathsf{GP}^4$$

We formally prove the following lemma in Appendix B.4.

Lemma 11 (From lookup to grand-product). *Construction 6 is an RoK of type* $\mathsf{LKP} \to \mathsf{GP}^4$ *with* 2 *rounds, a communication cost of* 6 *field elements, and* 2 *elements in the message space of the commitment scheme over vectors of size* n *and* m*. The prover's time complexity is* $O(n + m)$ *field operations, time to compute commitments to vectors of size* n *and* m*, and* $O(1)$ *operations in the message space of the commitment scheme. The verifier's time complexity is* $O(1)$*.*

By Lemma 9, we have a reduction of knowledge from lookup to four zero-check instances. By Corollary 1, we get the desired folding scheme for the lookup relation. However, by our optimization in Section 4, to combine two sum-check reductions into one, we can only fold the same number of "fresh" instances and "running" instances. However, in our setting, we must fold n lookup instances into $4n$ running instances. This is not an issue as we can always generate trivially satisfying running instances or forgo the optimization to remove this dependence.

Acknowledgments

We thank Justin Thaler for helpful conversations and comments on a prior version of this paper. Abhiram Kothapalli was supported by the Carnegie Mellon University (CMU) Secure Blockchain Initiative and Anaxi Labs through the CMU CyLab partnership program.

References

AS24. Arasu Arun and Srinath Setty. Nebula: Efficient read-write memory and switchboard circuits for folding schemes. Cryptology ePrint Archive, 2024.

AST24. Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: SNARKs for VMs via lookups. In *EUROCRYPT*, 2024.

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. In *ASIACRYPT*, 2023.

BC24a. Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257, 2024.

BC24b. Benedikt Bünz and Jessica Chen. Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325, 2024.

BCC+16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, 2016.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *STOC*, 2013.

BCL+21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *CRYPTO*, 2021.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. In *TCC*, 2020.

BCS21. Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *CRYPTO*, pages 742–773, 2021.

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, 2014.

BEG+91. Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *FOCS*, 1991.

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

BFLS91. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, 1991.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

BMNW24. Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. Cryptology ePrint Archive, Paper 2024/474, 2024.

BSBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO*, 2019.

BTVW14. Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. ePrint Report 2014/846, 2014.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT*, 2023.

CDD+03. Dwaine Clarke, Srinivas Devadas, Marten Van Dijk, Blaise Gassend, G. Edward, and Suh Mit. Incremental multiset hash functions and their application to memory integrity checking. In *ASIACRYPT*, 2003.

DGMV24. Nikolaos Dimitriou, Albert Garreta, Ignacio Manzur, and Ilia Vlasov. Mova: Nova folding without committing to error terms. Cryptology ePrint Archive, Paper 2024/1220, 2024.

DP23. Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023.

DT24. Quang Dao and Justin Thaler. More optimizations to sum-check proving. Cryptology ePrint Archive, Paper 2024/1210, 2024.

EG23. Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106, 2023.

EGS+24. Liam Eagen, Ariel Gabizon, Marek Sefranek, Patrick Towa, and Zachary J. Williamson. Stackproofs: Private proofs of stack and contract execution using protogalaxy. Cryptology ePrint Archive, Paper 2024/1281, 2024.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, 2008.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO*, 2023.

Gru24. Angus Gruen. Some improvements for the PIOP for ZeroCheck. Cryptology ePrint Archive, Paper 2024/108, 2024.

GW20. Ariel Gabizon and Zachary J Williamson. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive*, 2020.

GWC19. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530, 2022.

KP23. Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. In *CRYPTO*, 2023.

KS24. Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. In *CRYPTO*, pages 345–379, 2024.

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *CRYPTO*, 2022.

LFKN90. Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *FOCS*, October 1990.

NDC+24. Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. In *CRYPTO*, 2024.

nex24a. Summary of Nexus zkVM costs. `https://docs.nexus.xyz/specs/nexus-costs`, 2024.

nex24b. The Nexus zkVM. `https://github.com/nexus-xyz/nexus-zkvm`, 2024.

ova. Ova: A slightly better Nova. `https://hackmd.io/V4838nnlRKal9ZiTHiGYzw`.

SAGL18. Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *OSDI*, October 2018.

Sch80. Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4), 1980.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, 2020.

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

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

ST24. Srinath Setty and Justin Thaler. Folding (Jolt Book). `https://jolt.a16zcrypto.com/future/folding.html`, 2024.

STW23. Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, 2023.

STW24. Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with Lasso. In *EUROCRYPT*, 2024.

Tha13. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, 2013.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, pages 552–576, 2008.

VSBW13. Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for verifiable computation. In *S&P*, 2013.

ZV23. Yan Zhang and Aard Vark. Parallelizing Nova – Visualizations and Mental Models behind Paranova. `https://zkresear.ch/t/parallelizing-nova-visualizations-and-mental-models-behind-paranova/198`, 2023.

ZZD23. Zibo Zhou, Zongyang Zhang, and Jin Dong. Proof-carrying data from multi-folding schemes. Cryptology ePrint Archive, Paper 2023/1282, 2023.

A Additional Background

A.1 Polynomials and multilinear extensions

We adapt this subsection from prior work [Set20]. We recall several definitions and results regarding multivariate polynomials.

Definition 13 (Multilinear polynomial). *A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.*

Definition 14 (Multilinear polynomial extension). *Given a vector $v \in \mathbb{F}^n$ a* multilinear polynomial extension *of v is an $(\log n)$-variate multilinear polynomial, denoted $\widetilde{v}$, such that $\widetilde{v}(x) = v_x$ for all $x \in \{0,1\}^{\log n}$. Specifically, $\widetilde{v}$ can be computed as follows.*

$$\widetilde{v}(x) = \sum_{y \in \{0,1\}^\ell} v_y \cdot \mathsf{eq}(x, y)$$

where $\mathsf{eq}(x,y) = \prod_{i=1}^{\ell}(x_i \cdot y_i + (1-x_i)\cdot(1-y_i))$, outputs 1 if $x = y$ and 0 otherwise for $x, y \in \{0,1\}^{\log n}$.

For any $r \in \mathbb{F}^\ell$, $\widetilde{v}(r)$ can be computed in $O(2^\ell)$ operations in $\mathbb{F}$ [VSBW13,Tha13].

Lemma 12 (Schwartz-Zippel [Sch80]). *let $g : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variate polynomial of total degree at most d. Then, on any finite set $S \subseteq \mathbb{F}$,*

$$\Pr_{x \leftarrow S^\ell}[g(x) = 0] \leq d/|S|.$$

A.2 Commitment Schemes

Definition 15 (Commitment Scheme). *A commitment scheme is defined by polynomial-time algorithm $\mathsf{Gen} : \mathbb{N}^2 \to P$ that produces public parameters given the security parameter and size parameter, a deterministic polynomial-time algorithm $\mathsf{Commit} : P \times M \times R \to C$ that produces a commitment in C given a public parameters, message, and randomness tuple such that binding holds. That is, for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, and given $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(\mathsf{pp})$ we have that*

$$\Pr[(m_1, r_1) \neq (m_2, r_2) \wedge \mathsf{Commit}(\mathsf{pp}, m_1, r_1) = \mathsf{Commit}(\mathsf{pp}, m_2, r_2)] \approx 0.$$

The commitment scheme is deterministic if Commit does not use its randomness.

Definition 16 (Homomorphic). *The commitment scheme $(\mathsf{Gen}, \mathsf{Commit})$ is homomorphic if the message space M, randomness space R, and commitment space C are groups and for all $n \in \mathbb{N}$, and $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, we have that for any $m_1, m_2 \in M$ and $r_1, r_2 \in R$*

$$\mathsf{Commit}(\mathsf{pp}, m_1, r_1) + \mathsf{Commit}(\mathsf{pp}, m_2, r_2) = \mathsf{Commit}(\mathsf{pp}, m_1 + m_2, r_1 + r_2).$$

Definition 17 (Succinct Commitments). *A commitment scheme* $(\mathsf{Gen}, \mathsf{Commit})$*, over message space* M *and commitment space* R*, provides succinct commitments if for all* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$*, and any* $m \in M$ *and* $r \in R$*, we have that* $|\mathsf{Commit}(\mathsf{pp}, m, r)| = O_\lambda(\mathsf{polylog}(|m|))$*.*

A.3 Reductions of Knowledge

First, we define additional properties of reductions of knowledge.

Definition 18 (Succinctness). *A reduction of knowledge is succinct if the communication complexity and the verifier time complexity is at most polylogarithmic in the size of the structure and witness.*

Definition 19 (Non-interactivity). *A reduction of knowledge is non-interactive if the interaction consists of a single message from the prover to the verifier. In this case, we denote this single message as the output of the prover, and as an input to the verifier.*

Definition 20 (Public-coin). *A reduction of knowledge is public-coin if the verifier only sends uniformly random challenges during the interaction.*

Next, we define the semantics of the sequential and parallel composition operators.

Lemma 13 (Sequential composition [KP23]). *For reductions* $\Pi_1 = (\mathcal{G}, \mathcal{K}_1, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}_2, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \to \mathcal{R}_3$*, we have that* $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \to \mathcal{R}_3$ *where* $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ *computes* $(\mathsf{pk}_1, \mathsf{vk}_1, \mathsf{s}_2) \leftarrow \mathcal{K}_1(\mathsf{pp}, \mathsf{s}_1)$*,* $(\mathsf{pk}_2, \mathsf{vk}_2, \mathsf{s}_3) \leftarrow \mathcal{K}_2(\mathsf{pp}, \mathsf{s}_2)$ *and outputs* $((\mathsf{pk}_1, \mathsf{pk}_2), (\mathsf{vk}_1, \mathsf{vk}_2), \mathsf{s}_3)$ *and where*

$$\mathcal{P}((\mathsf{pk}_1, \mathsf{pk}_2), u_1, w_1) = \mathcal{P}_2(\mathsf{pk}_2, \mathcal{P}_1(\mathsf{pk}_1, u_1, w_1))$$
$$\mathcal{V}((\mathsf{vk}_1, \mathsf{vk}_2), u_1) = \mathcal{V}_2(\mathsf{vk}_2, \mathcal{V}_1(\mathsf{vk}_1, u_1, w_1))$$

Lemma 14 (Parallel composition [KP23]). *Consider relations* $\mathcal{R}_1$*,* $\mathcal{R}_2$*,* $\mathcal{R}_3$*, and* $\mathcal{R}_4$*. For reductions of knowledge* $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \to \mathcal{R}_4$ *we have that* $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$ *where*

$$\mathcal{P}(\mathsf{pk}, (u_1, u_3), (w_1, w_3)) = (\mathcal{P}_1(\mathsf{pk}, u_1, w_1), \mathcal{P}_2(\mathsf{pk}, u_3, w_3))$$
$$\mathcal{V}(\mathsf{vk}, (u_1, u_3)) = (\mathcal{V}_1(\mathsf{vk}, u_1), \mathcal{V}_2(\mathsf{vk}, u_3))$$

Next, we can define arguments of knowledge as a special type of reduction of knowledge.

Definition 21 (Argument of Knowledge). *Consider the boolean relation* $\mathsf{TRUE} = \{(\mathsf{true}, \bot)\}$*. A proof of knowledge for relation* $\mathcal{R}$ *is a reduction of knowledge of type* $\mathcal{R} \to \mathsf{TRUE}$*.*

When proving the security of protocols, reasoning about knowledge soundness directly is typically cumbersome. To alleviate this issue, Bootle et al. [BCC+16] show that to prove knowledge soundness for the vast majority of public-coin interactions, it is sufficient to show that there exists an extractor that can produce a satisfying witness when provided a tree of accepting transcripts with refreshed verifier randomness at each layer. This property is known as *tree-extractability* and we formally state the corresponding result for reductions of knowledge (proven by Kothapalli and Parno [KP23]).

Definition 22 (Tree of transcripts). *Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 1. For a public parameter, structure, instance tuple $(\mathsf{pp}, \mathsf{s}, u_1)$, a $(n_1, \ldots, n_m)$-tree of accepting transcripts is a tree of depth m where each vertex at layer i has n_i outgoing edges such that (1) each vertex in layer $i \in [m]$ is labeled with a prover message for round i; (2) each outgoing edge from layer $i \in [m]$ is labeled with a different choice of verifier randomness for round i; (3) each leaf is labeled with an accepting statement-witness pair output by the prover and verifier corresponding to the interaction along the path.*

Lemma 15 (Tree extraction [KP23]). *Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 1 and satisfies completeness. Then $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge if there exists a PPT extractor χ that outputs a satisfying witness w_1 with probability $1 - \mathsf{negl}(\lambda)$, given an $(n_1, \ldots, n_m)$-tree of accepting transcripts for public parameter, structure, instance tuple $(\mathsf{pp}, \mathsf{s}, u_1)$ where the verifier's randomness is sampled from space Q such that $|Q| = O(2^\lambda)$, and $\prod_i n_i = \mathsf{poly}(\lambda)$.*

A.4 Incrementally Verifiable Computation

Recall that incrementally verifiable computation (IVC), enables a prover to to produce a proof π_{i+1} for $i + 1$ steps of a computation given a proof π_i for i steps of the computation in a way that ensures that the proof size remains independent of the total steps of computation. As IVC is a major application of folding schemes [BCL+21,KST22,KS24], for completeness, we recall the formal definition below.

Definition 23 (Incrementally verifiable computation (IVC)). *An incrementally verifiable computation (IVC) scheme is defined by 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*

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$*: on input security parameter λ and size bounds N, samples public parameters* pp*.*
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$*: on input public parameters* pp*, and polynomial-time function F, deterministically produces a prover key* pk *and a verifier key* vk*.*

- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$*: on input a prover key* pk*, a counter* i*, an initial input* z_0*, a claimed output after* i *iterations* z_i*, a non-deterministic advice* ω_i*, and an IVC proof* Π_i *attesting to* z_i*, produces a new proof* Π_{i+1} *attesting to* $z_{i+1} = F(z_i, \omega_i)$*.*
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$*: on input a verifier key* vk*, a counter* i*, an initial input* z_0*, a claimed output after* i *iterations* z_i*, and an IVC proof* Π_i *attesting to* z_i*, outputs* 1 *if* Π_i *is accepting, and* 0 *otherwise.*

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *satisfies the following requirements.*

1. *Perfect Completeness: For any PPT adversary* $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ F, (i, z_0, z_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ z_{i+1} \leftarrow F(z_i, \omega_i), \\ \mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1, \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

   where F *is a polynomial-time computable function represented as an arithmetic circuit.*

2. *Knowledge Soundness: Consider constant* $n \in \mathbb{N}$*. For all expected polynomial-time adversaries* $\mathcal{P}^*$ *there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that over all randomness* ρ

$$\Pr\left[\begin{array}{l} z_n = z \text{ where} \\ z_{i+1} \leftarrow F(z_i, \omega_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (F, (z_0, z_i), \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F), \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1, \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, \rho) \end{array}\right] \approx 1.$$

   Moreover, F *is a polynomial-time computable function represented as an arithmetic circuit.*

3. *Succinctness: The size of an IVC proof* Π *is independent of the number of iterations* n*.*

B Deferred Proofs

B.1 Proof of Theorem 2 (SumFold)

Lemma 16 (Completeness). *Construction 1 is complete.*

Proof. Consider an arbitrary generator and encoder algorithm $(\mathcal{G}, \mathcal{K})$, and an arbitrary PPT adversary $\mathcal{A}$. For an arbitrary size parameter N, and a structure (F, G), let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$ and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (F, G))$. Suppose now that the adversary $\mathcal{A}$ on input pp generates input instance-witness pairs

$$(\{(T_i, \vec{u}_i, \mathsf{x}_i)\}_{i \in [n]}, \{\vec{w}_i\}_{i \in [n]}) \in \mathsf{SC}^n. \tag{11}$$

The prover and the verifier on input $(\mathsf{pk}, \mathsf{vk}), \{(T_i, \vec{u}_i, \mathsf{x}_i)\}_{i\in[n]}, \{\vec{w}_i\}_{i\in[n]}$ interactively produce the output instance-witness pairs

$$((T', \vec{u}, \mathsf{x}), \vec{w}).$$

We must show that $((T', \vec{u}, \mathsf{x}), \vec{w}) \in \mathsf{SC}$.

Indeed, by the linearity of the commitment scheme and by the satisfiability of the input instances, we have that for $j \in [s]$

$$\begin{aligned} u_j &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot u_{i,j} \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot \mathsf{Commit}(\mathsf{pp}, w_{i,j}) \\ &= \mathsf{Commit}(\mathsf{pp}, \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot w_{i,j}) \\ &= \mathsf{Commit}(\mathsf{pp}, w_j). \end{aligned} \tag{12}$$

That is, we have that all output witnesses are satisfying openings to the output commitments.

Moreover, given $\vec{g}_i \leftarrow G(\vec{w}_i, \mathsf{x}_i)$, we have that

$$\begin{aligned} T &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(\rho, i) \cdot T_i && \text{By construction} \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(\rho, i) \cdot \sum_{x\in\{0,1\}^\ell} F(\vec{g}_i(x)) && \text{By Precondition (11)} \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(\rho, i) \cdot \sum_{x\in\{0,1\}^\ell} F(\vec{f}(i, x)) && \text{By construction} \\ &= \sum_{i\in\{0,1\}^\nu} Q(i) && \text{By construction} \end{aligned}$$

Therefore, by the completeness of the sum-check protocol, we have that

$$c = Q(r_b).$$

This means that

$$\begin{aligned} T' &= \mathsf{eq}(\rho, r_b)^{-1} \cdot c \\ &= \mathsf{eq}(\rho, r_b)^{-1} \cdot Q(r_b) \\ &= \sum_{x\in\{0,1\}^\ell} F(\vec{f}(r_b, x)). \end{aligned} \tag{13}$$

But, by the linearity of G (Definition 7), we have that

$$\begin{aligned} G(\vec{w}, \mathsf{x}) &= G(\sum_{i \in \{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot \vec{w}_i, \sum_{i \in \{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot \mathsf{x}_i) \\ &= \sum_{i \in \{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot G(\vec{w}_i, \mathsf{x}_i) \\ &= \sum_{i \in \{0,1\}^\nu} \mathsf{eq}(r_b, i) \cdot \vec{g}_i \\ &= \vec{f}(r_b, x). \end{aligned} \tag{14}$$

Thus, by derivations (12), (13), and (14), we have that $((T', \vec{u}, \mathsf{x}), \vec{w}) \in \mathsf{SC}$. □

Lemma 17 (Knowledge soundness). *Construction 1 is knowledge sound (and in particular tree-extractable).*

Proof. We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct a PPT extractor χ that outputs a satisfying input witness with probability $1 - \mathsf{negl}(\lambda)$ given a tree of accepting transcripts and the corresponding output instance-witness pairs.

Indeed, let χ_{sc} denote the tree-extractor for the sum-check protocol which reduces from the unstructured sum-check relation USC to the polynomial evaluation relation PE. Suppose that χ_{sc} can extract using an L-tree of accepting transcripts where $L \in \mathbb{N}^\nu$. Then, we provide χ with a $(2^\nu, L)$-tree of accepting transcripts for public parameter, structure, instance tuple $(\mathsf{pp}, (F, G), \{(T_i, \vec{u}_i, \mathsf{x}_i)\}_{i \in [n]})$. For $k \in [2^\nu]$ and $l \in ([L_1], \ldots, [L_\nu])$, let $\rho^{(k)}$ denote the verifier's first challenge, and for each such challenge let $r_b^{(k,l)}$ denote the verifier's challenges during the sum-check protocol. Let

$$((T'^{(k,l)}, \vec{u}^{(k,l)}, \mathsf{x}^{(k,l)}), \vec{w}^{(k,l)}) \in \mathsf{SC} \tag{15}$$

denote the corresponding output instance-witness pairs.

For $k \in [2^\ell]$, the extractor χ interpolates $\vec{w}_i^{(k)}$ for $i \in [n]$ such that for $j \in [s]$

$$w_j^{(k,l)} = \sum_{i \in \{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i) \cdot w_{i,j}^{(k)}. \tag{16}$$

Now, by the satisfiability requirement of SC (Precondition 15), we have that

$$T'^{(k,l)} = \sum_{x \in \{0,1\}^l} F(G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)})(x)) \tag{17}$$

$$u_j^{(k,l)} = \mathsf{Commit}(\mathsf{pp}, w_j^{(k,l)}) \tag{18}$$

Moreover, by the verifier's computation we have that

$$\begin{aligned} G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)}) &= G(\sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot \vec{w}_i^{(k)}, \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot \mathsf{x}_i^{(k)}) \quad \text{By (16)} \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot G(\vec{w}_i^{(k)}, \mathsf{x}_i^{(k)}) \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot \vec{g}_{i,1}^{(k)} \\ &= \vec{f}^{(k)}(r_b^{(k,l)}) \end{aligned}$$

Then, by the verifier's computation in Step 3 and Equation (17), this means that

$$\begin{aligned} c^{(k,l)} &= \mathsf{eq}(\rho^{(k)}, r_b^{(k,l)})\cdot T'^{(k,l)} \\ &= \mathsf{eq}(\rho^{(k)}, r_b^{(k,l)})\cdot \sum_{x\in\{0,1\}^\ell} F(G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)})(x)) \\ &= \mathsf{eq}(\rho^{(k)}, r_b^{(k,l)})\cdot \sum_{x\in\{0,1\}^\ell} F(\vec{f}^{(k)}(r_b^{(k,l)}, x)) \\ &= Q^{(k)}(r_b^{(k,l)}) \end{aligned} \tag{19}$$

Moreover, by the verifier's computation for $i \in [n]$ and $j \in [s]$, we have that for all $r_b^{(k,l)}$

$$\begin{aligned} \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot u_{i,j}^{(k)} &= u_j^{(k,l)} \\ &= \mathsf{Commit}(\mathsf{pp}, w_j^{(k,l)}) && \text{By (18)} \\ &= \mathsf{Commit}(\mathsf{pp}, \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot w_{i,j}^{(k)}) && \text{By (16)} \\ &= \sum_{i\in\{0,1\}^\nu} \mathsf{eq}(r_b^{(k,l)}, i)\cdot \mathsf{Commit}(\mathsf{pp}, w_{i,j}^{(k)}) && \text{By linearity} \end{aligned}$$

Then, by interpolation, we have that for all $i \in [n]$ and $j \in [s]$ that

$$u_{i,j}^{(k)} = \mathsf{Commit}(\mathsf{pp}, w_{i,j}^{(k)}). \tag{20}$$

Then, by Derivation (19) and Equation (20), we have that for each challenge $\rho^{(k)}$, χ can produce an L-sub-tree of accepting transcripts for an input unstructured sum-check instance $(T^{(k)}, \overline{Q}^{(k)})$ where $\overline{Q}^{(k)} = ((F, G), \rho^{(k)}, (\vec{u}_i, \mathsf{x}_i)_{i\in[n]})$, with output polynomial evaluation instance-witness pairs

$$((\overline{Q}^{(k)}, c^{(k,l)}, r_b^{(k,l)}), Q^{(k)}) \in \mathsf{PE}$$

where $Q^{(k)}$ is represented as the tuple $((F, G), \rho^{(k)}, (\vec{w}_i^{(k)}, \mathsf{x}_i)_{i\in[n]})$.

Now, for each challenge $\rho^{(k)}$, given the corresponding L-sub-tree χ_{sc} can produce a witness $Q^{(k)}$ represented as the tuple $((F, G), \rho^{(k)}, (\vec{w}_i^{(k)}, \mathsf{x}_i)_{i\in[n]})$ such that

$$((T^{(k)}, \overline{Q}^{(k)}), Q^{(k)}) \in \mathsf{USC}. \tag{21}$$

Then, because $Q^{(k)}$ must be a valid opening to $\overline{Q}^{(k)}$ which contains the same commitment $\vec{u}_i$ across all transcripts by Equation (21), we must have that $\vec{w}_i^{(k)} = \vec{w}_i^{(k')}$ for all $k, k' \in [2^\ell]$. We denote this value across all transcripts simply as $\vec{w}_i$.

By the satisfiability condition of USC and the verifier's computation, this means that

$$\begin{aligned}\sum_{i\in\{0,1\}^\nu} \mathsf{eq}(\rho^{(k)}, i) \cdot T_i = T^{(k)} \\ &= \sum_{b\in\{0,1\}^\nu} Q^{(k)}(\rho^{(k)}) \\ &= \sum_{b\in\{0,1\}^\nu} \mathsf{eq}(\rho^{(k)}, b) \cdot \sum_{x\in\{0,1\}^\ell} F(\vec{f}(b, x))\end{aligned}$$

Then, interpolating, we must have for $i \in \{0, 1\}^\nu$

$$\begin{aligned}T_i &= \sum_{x\in\{0,1\}^\ell} (\vec{f}(i, x)) \\ &= F(\vec{g}_i(x)) \\ &= F(G(\vec{w}_i, \mathsf{x}_i)(x))\end{aligned}$$

This means that

$$(\{(T_i, \vec{u}_i, \mathsf{x}_i)\}_{i\in[n]}, \{\vec{w}_i\}_{i\in[n]}) \in \mathsf{SC}^n.$$

Thus, χ can compute and output a satisfying input witness $\vec{w}_i$ for $i \in [n]$. □

B.2 Proof of Lemma 5 (Zero-check reduction)

Lemma 18 (Completeness). *Construction 3 is complete.*

Proof. Consider an arbitrary generator and encoder algorithms $(\mathcal{G}, \mathcal{K})$, and an arbitrary PPT adversary $\mathcal{A}$. For an arbitrary size parameter N, let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$. Suppose now that the adversary $\mathcal{A}$ on input pp, generates input structure-instance-witness pairs

$$(F, G), (\vec{\mathsf{u}}, \vec{\mathsf{u}}_{\mathsf{pc}}), (\vec{\mathsf{w}}, \vec{\mathsf{w}}_{\mathsf{pc}}) \in \mathsf{ZC}^n \times \mathsf{ZC}^m_{\mathsf{PC}}. \tag{22}$$

Suppose that for $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (F, G))$, the prover and the verifier on input $(\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}}, \vec{\mathsf{u}}_{\mathsf{pc}}), (\vec{\mathsf{w}}, \vec{\mathsf{w}}_{\mathsf{pc}})$ interactively produce the output instance-witness pairs

$$(\{(0, \mathsf{u}_i, \overline{e})\}_{i\in[n]}, \{(\mathsf{w}_i, e)\}_{i\in[n]}), (\{(0, \mathsf{u}_{\mathsf{pc},j}, \overline{e})\}_{j\in[m]}, \{(\mathsf{w}_{\mathsf{pc},j}, e)\}_{j\in[m]}), ((\overline{e}, \tau), e).$$

We must show that

$$(\{(0, \mathsf{u}_i, \overline{e})\}_{i\in[n]}, \{(\mathsf{w}_i, e)\}_{i\in[n]}) \in \mathsf{NSC}^n \tag{23}$$

$$(\{(0, \mathsf{u}_{\mathsf{pc},j}, \overline{e})\}_{j\in[m]}, \{(\mathsf{w}_{\mathsf{pc},j}, e)\}_{j\in[m]}) \in \mathsf{NSC}^m_{\mathsf{PC}} \tag{24}$$

$$((\overline{e}, \tau), e) \in \mathsf{ZC}_{\mathsf{PC}}. \tag{25}$$

Indeed, by the prover's construction (Step 2), we immediately have that Equation 25 holds.

Now, for each $i \in [n]$, we parse $(\vec{u}_i, \mathsf{x}_i) \leftarrow \mathsf{u}_i$, and $\vec{w}_i \leftarrow \mathsf{w}_i$. By Precondition (22) we have that

$$\mathsf{Commit}(\mathsf{pp}, w_{i,j}) = u_{i,j}. \tag{26}$$

Moreover, by the precondition, we have that for all $x \in \{0,1\}^\ell$

$$0 = F(G(\vec{w}_i, \mathsf{x}_i)(x)).$$

This means that for any choice of e

$$0 = \sum_{x\in\{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(G(\vec{w}_i, \mathsf{x}_i)(x)) \tag{27}$$

where x_1 and x_2 (likewise e_1 and e_2) represent the first and the second half of the original vector. Then, by Equations (26) and (27), we have that Equation (23) holds. By an identical line of reasoning, we have that Equation (24) holds as well. Thus, we have that completeness holds. □

Lemma 19 (Knowledge Soundness). *Construction 3 is knowledge sound.*

Proof. We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct a PPT extractor χ that outputs a satisfying input witness with probability $1 - \mathsf{negl}(\lambda)$ given a tree of accepting transcripts and the corresponding output instance-witness pairs.

Indeed, suppose χ is provided with a (2^ℓ)-tree of accepting transcripts for public parameter, structure, instance tuple $(\mathsf{pp}, (F, G), (\{\vec{\mathsf{u}}_i\}_{i\in[n]}, \{\vec{\mathsf{u}}_{\mathsf{pc},j}\}_{j\in[m]}))$. For $k \in [2^\ell]$, let $\tau^{(k)}$ denote the verifier challenge in each of these transcripts, and let

$$(\{(0, \mathsf{u}_i^{(k)}, \overline{e}^{(k)})\}_{i\in[n]}, \{(\mathsf{w}_i^{(k)}, e^{(k)})\}_{i\in[n]}) \in \mathsf{NSC}^n \tag{28}$$

$$(\{(0, \mathsf{u}_{\mathsf{pc},j}^{(k)}, \overline{e}^{(k)})\}_{j\in[m]}, \{(\mathsf{w}_{\mathsf{pc},j}^{(k)}, e^{(k)})\}_{j\in[m]}) \in \mathsf{NSC}^m_{\mathsf{PC}} \tag{29}$$

$$((\overline{e}^{(k)}, \tau^{(k)}), e^{(k)}) \in \mathsf{ZC}_{\mathsf{PC}} \tag{30}$$

denote the corresponding output instance-witness pairs.

By the verifier's construction (Step 3) we have that $\mathsf{u}_i^{(k)}$ is the same as the input instance u_i for all $k \in [2^\ell]$. Then, by the binding property of the commitment scheme, we have that $\mathsf{w}_i^{(k)} = \mathsf{w}_i^{(k')}$ for all $k, k' \in [2^\ell]$. Thus, we denote this value as w_i. Now, for each $i \in [n]$, we parse $(\vec{u}_i, \mathsf{x}_i) \leftarrow \mathsf{u}_i$, and $\vec{w}_i \leftarrow \mathsf{w}_i$. By Equation (28), for each $k \in [2^\ell]$, given

$$\vec{g}_i \leftarrow G(\vec{w}_i, \mathsf{x}_i),$$

we have that

$$0 = \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1^{(k)}(x_1) \cdot \widetilde{e}_2^{(k)}(x_2) \cdot F(\vec{g}_i(x)),$$

where x_1 and x_2 (likewise $e_1^{(k)}$ and $e_2^{(k)}$) represent the first and the second half of the original vector. But by Equation (30), we have that

$$\begin{aligned} e_1 &= ((\tau^{(k)})^0, (\tau^{(k)})^1, (\tau^{(k)})^2, \ldots, (\tau^{(k)})^{\sqrt{m}-1}) \\ e_2 &= ((\tau^{(k)})^0, (\tau^{(k)})^{\sqrt{m}}, (\tau^{(k)})^{2\sqrt{m}}, \ldots, (\tau^{(k)})^{(\sqrt{m}-1)\cdot\sqrt{m}}). \end{aligned}$$

Substituting, we have that

$$0 = \sum_{x \in \{0,1\}^\ell} (\tau^{(k)})^{X_1 + \sqrt{m} \cdot X_2} \cdot F(\vec{g}_i(x)),$$

where $X_i = \sum_{j \in [\ell/2]} x_{i,j} \cdot 2^{\ell/2 - j}$ (i.e. the corresponding decimal representation if x_i is treated as the bit representation). Then, by interpolation, we have that the Lagrange polynomial

$$Q(Y) = \sum_{x \in \{0,1\}^\ell} Y^{X_1 + \sqrt{m} \cdot X_2} \cdot F(\vec{g}_i(x))$$

is the zero polynomial. This means that

$$0 = F(\vec{g}_i(x))$$

for all $x \in \{0,1\}^\ell$. Hence, we have that

$$(\vec{\mathsf{u}}, \vec{\mathsf{w}}) \in \mathsf{ZC}^n.$$

Then, χ can produce $\vec{\mathsf{w}}_i$ for all $i \in [n]$ by reading the output witness from any one of the transcripts. By an identical line of reasoning χ can produce $\vec{\mathsf{w}}_{\mathsf{pc}}$ such that

$$(\vec{\mathsf{u}}_{\mathsf{pc}}, \vec{\mathsf{w}}_{\mathsf{pc}}) \in \mathsf{ZC}^m_{\mathsf{PC}}.$$

Thus, we have that tree-extractability holds. □

B.3 Proof of Lemma 6 (Folding nested sum-check)

Lemma 20 (Completeness). *Construction 4 is complete.*

Proof. Consider an arbitrary generator and encoder algorithm $(\mathcal{G}, \mathcal{K})$, and an arbitrary PPT adversary $\mathcal{A}$. For an arbitrary size parameter N, let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, N)$. Suppose now that the adversary $\mathcal{A}$ on input pp generates a structure (F, G), and input instance-witness pairs

$$\{((T_i, \mathsf{u}_i, \mathsf{x}_i), (T_{\mathsf{pc},i}, \mathsf{u}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}))\}_{i\in[n]}, \{(\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i})\}_{i\in[n]} \in \mathsf{NSC}^n \times \mathsf{NSC}^n_{\mathsf{PC}}. \quad (31)$$

Suppose for $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (F, G))$, the prover and the verifier on input $(\mathsf{pk}, \mathsf{vk})$ and this instance-witness pair interactively produce the output instance-witness pair

$$(((T, \mathsf{u}, \mathsf{x}), (T_{\mathsf{pc}}, \mathsf{u}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}})), (\mathsf{w}, \mathsf{w}_{\mathsf{pc}})).$$

We must show that

$$((T, \mathsf{u}, \mathsf{x}), \mathsf{w}) \in \mathsf{NSC}$$
$$((T_{\mathsf{pc}}, \mathsf{u}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}}), \mathsf{w}_{\mathsf{pc}}) \in \mathsf{NSC}_{\mathsf{PC}}.$$

Indeed, for each $i \in [n]$, we parse $(\vec{w}_i, e_i) \leftarrow \mathsf{w}_i$ and $(\vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}) \leftarrow \mathsf{w}_{\mathsf{pc},i}$. Then, for $\vec{g}_i \leftarrow G(\vec{w}_i, \mathsf{x}_i)$ and $\vec{g}_{\mathsf{pc},i} \leftarrow G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i})$, by Precondition 31, we have that

$$\begin{aligned}
&T_i + \gamma \cdot T_{\mathsf{pc},i} \\
&= \sum_{x\in\{0,1\}^\ell} \widetilde{e}_{i,1}(x_1) \cdot \widetilde{e}_{i,2}(x_2) \cdot F(\vec{g}_i(x)) + \gamma \cdot \widetilde{e}_{\mathsf{pc},i,1}(x_1) \cdot \widetilde{e}_{\mathsf{pc},i,2}(x_2) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc},i}(x)) \\
&= \sum_{x\in\{0,1\}^\ell} h_{1,i}(x) \cdot h_{2,i}(x) \cdot F(\vec{g}_i(x)) + \gamma \cdot h_{\mathsf{pc},1,i}(x) \cdot h_{\mathsf{pc},2,i}(x) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc},i}(x)) \\
&= \sum_{x\in\{0,1\}^\ell} F'(\vec{g}_i(x), h_{1,i}(x), h_{2,i}(x), \vec{g}_{\mathsf{pc,i}}, h_{\mathsf{pc},1,i}(x), h_{\mathsf{pc},2,i}(x)) \\
&= \sum_{x\in\{0,1\}^\ell} F'(G'(\vec{w}_i, e_i, \vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}, (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i}))(x)),
\end{aligned}$$

where $h_{1,i}$, $h_{2,i}$, $h_{\mathsf{pc},1,i}$, and $h_{\mathsf{pc},2,i}$ are defined as in the construction. This implies that the completeness precondition holds for the sum-check folding reduction. That is, for $i \in [n]$

$$(\mathsf{pp}, (F', G'), (T_i + \gamma \cdot T_{\mathsf{pc},i}, (\mathsf{u}_i, \mathsf{u}_{\mathsf{pc},i}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i})), (\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i})) \in \mathsf{SC}.$$

Then, by the completeness of the sum-check folding reduction, we have that the instance-witness pair output from the folding sum-check reduction is satisfying. That is

$$(\mathsf{pp}, (F', G'), (T_\gamma, (\mathsf{u}, \mathsf{u}_{\mathsf{pc}}), (\mathsf{x}, \mathsf{x}_{\mathsf{pc}})), (\mathsf{w}, \mathsf{w}_{\mathsf{pc}})) \in \mathsf{SC}. \quad (32)$$

This implies that given $(\vec{w}, e) \leftarrow \mathsf{w}$, $(\vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}) \leftarrow \mathsf{w}_{\mathsf{pc}}$, $\vec{g} \leftarrow G(\vec{w}, e)$, and $\vec{g}_{\mathsf{pc}} \leftarrow G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}})$ we have that

$$\begin{aligned}
T_\gamma &= \sum_{x \in \{0,1\}^\ell} F'(G'(\vec{w}, e, \vec{w}_{\mathsf{pc}}, e_{\mathsf{pc}}, (\mathsf{x}, \mathsf{x}_{\mathsf{pc}}))(x)) \\
&= \sum_{x \in \{0,1\}^\ell} h_1(x) \cdot h_2(x) \cdot F(\vec{g}(x)) + \gamma \cdot h_{\mathsf{pc},1}(x) \cdot h_{\mathsf{pc},2}(x) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}(x)) \\
&= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(\vec{g}(x)) + \gamma \cdot \widetilde{e}_{\mathsf{pc},1}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2}(x) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}(x)) \\
&= T + \gamma \cdot T_{\mathsf{pc}}.
\end{aligned}$$

Therefore, we have that the verifier's check in Step 5 passes.

Moreover, by the prover's computation, we have that

$$\begin{aligned}
T &= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_1(x_1) \cdot \widetilde{e}_2(x_2) \cdot F(\vec{g}(x)) \\
T_{\mathsf{pc}} &= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{\mathsf{pc},1}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2}(x) \cdot F_{\mathsf{PC}}(\vec{g}_{\mathsf{pc}}(x))
\end{aligned}$$

Then, by the validity of the commitments implied by Equation 32, we have that

$$\begin{aligned}
((T, (\mathsf{u}, \mathsf{x})), \mathsf{w}) &\in \mathsf{NSC} \\
((T_{\mathsf{pc}}, (\mathsf{u}_{\mathsf{pc}}, \mathsf{x}_{\mathsf{pc}})), \mathsf{w}_{\mathsf{pc}}) &\in \mathsf{NSC}_{\mathsf{PC}}.
\end{aligned}$$

Therefore, completeness holds. □

Lemma 21 (Knowledge Soundness). *Construction 4 is knowledge sound.*

Proof. We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct a PPT extractor χ that outputs a satisfying input witness with probability $1 - \mathsf{negl}(\lambda)$ given a tree of accepting transcripts and the corresponding output instance-witness pairs.

Indeed, let χ_{FSC} denote the tree-extractor for the sum-check folding reduction, which reduces from SC^n to SC in Step 3. Suppose that χ_{FSC} can extract using an L-tree of accepting transcripts where $L \in \mathbb{N}^{\nu+1}$. Then, we provide χ with a $(2, L)$-tree of accepting transcripts for public parameter, structure, instance tuple

$$(\mathsf{pp}, (F, G), \{((T_i, \mathsf{u}_i, \mathsf{x}_i), (T_{\mathsf{pc},i}, \mathsf{u}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}))\}_{i \in [n]}).$$

For $k \in [2]$ and $l \in ([L_1], \ldots, [L_{\nu+1}])$, let $\gamma^{(k)}$ denote the verifier's first challenge, and for each such challenge let $r^{(k,l)}$ denote the verifier's challenges during the sum-check folding reduction. Let

$$\begin{aligned}
((T^{(k,l)}, \mathsf{u}^{(k,l)}, \mathsf{x}^{(k,l)}), \mathsf{w}^{(k,l)}) &\in \mathsf{NSC} \\
((T^{(k,l)}_{\mathsf{pc}}, \mathsf{u}^{(k,l)}_{\mathsf{pc}}, \mathsf{x}^{(k,l)}_{\mathsf{pc}}), \mathsf{w}^{(k,l)}_{\mathsf{pc}}) &\in \mathsf{NSC}_{\mathsf{PC}}
\end{aligned}$$

denote the corresponding output instance-witness pairs.

Now, by the satisfiability condition of NSC and $\mathsf{NSC}_{\mathsf{PC}}$ for

$$\begin{aligned}(\vec{u}^{(k,l)}, \overline{e}^{(k,l)}) &\leftarrow \mathsf{u}^{(k,l)}\\ (\vec{u}^{(k,l)}_{\mathsf{pc}}, \overline{e}^{(k,l)}_{\mathsf{pc}}) &\leftarrow \mathsf{u}^{(k,l)}_{\mathsf{pc}}\\ (\vec{w}^{(k,l)}, e^{(k,l)}) &\leftarrow \mathsf{w}^{(k,l)}\\ (\vec{w}^{(k,l)}_{\mathsf{pc}}, e^{(k,l)}_{\mathsf{pc}}) &\leftarrow \mathsf{w}^{(k,l)}_{\mathsf{pc}},\end{aligned}$$

we have that

$$\begin{aligned}T^{(k,l)} &= \sum_{x\in\{0,1\}^{\ell}} \widetilde{e}^{(k,l)}_1(x_1)\cdot\widetilde{e}^{(k,l)}_2(x_2)\cdot F(G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)})(x))\\ T^{(k,l)}_{\mathsf{pc}} &= \sum_{x\in\{0,1\}^{\ell}} \widetilde{e}^{(k,l)}_{\mathsf{pc},1}(x_1)\cdot\widetilde{e}^{(k,l)}_{\mathsf{pc},2}(x_2)\cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}^{(k,l)}_{\mathsf{pc}}, \mathsf{x}^{(k,l)}_{\mathsf{pc}})(x))\end{aligned}$$

and that

$$\overline{e}^{(k,l)} = \mathsf{Commit}(\mathsf{pp}, e^{(k,l)}) \tag{33}$$
$$\overline{e}^{(k,l)}_{\mathsf{pc}} = \mathsf{Commit}(\mathsf{pp}, e^{(k,l)}_{\mathsf{pc}}) \tag{34}$$
$$u^{(k,l)}_j = \mathsf{Commit}(\mathsf{pp}, w^{(k,l)}_j) \tag{35}$$
$$u^{(k,l)}_{\mathsf{pc},j} = \mathsf{Commit}(\mathsf{pp}, w^{(k,l)}_{\mathsf{pc},j}) \tag{36}$$

for $j \in [s]$.

Then, by the verifier's check in Step 5, we have that

$$
\begin{aligned}
& T_{\gamma}^{(k,l)} \\
&= T^{(k,l)} + \gamma^{(k,l)} \cdot T_{\mathsf{pc}}^{(k,l)} \\
&= \sum_{x \in \{0,1\}^{\ell}} \widetilde{e}_1^{(k,l)}(x_1) \cdot \widetilde{e}_2^{(k,l)}(x_2) \cdot F(G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)})(x)) + \\
&\quad \gamma^{(k,l)} \cdot \sum_{x \in \{0,1\}^{\ell}} \widetilde{e}_{\mathsf{pc},1}^{(k,l)}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2}^{(k,l)}(x_2) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)})(x)) \\
&= \sum_{x \in \{0,1\}^{\ell}} h_1^{(k,l)}(x) \cdot h_2^{(k,l)}(x) \cdot F(G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)})(x)) + \\
&\quad \gamma^{(k,l)} \cdot \sum_{x \in \{0,1\}^{\ell}} h_{\mathsf{pc},1}^{(k,l)}(x) \cdot h_{\mathsf{pc},2}^{(k,l)}(x) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)})(x)) \\
&= \sum_{x \in \{0,1\}^{\ell}} F'((G(\vec{w}^{(k,l)}, \mathsf{x}^{(k,l)}), h_1^{(k,l)}, h_2^{(k,l)}, G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc}}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)}), h_{\mathsf{pc},1}^{(k,l)}, h_{\mathsf{pc},2}^{(k,l)})(x)) \\
&= \sum_{x \in \{0,1\}^{\ell}} F'(G'(\vec{w}^{(k,l)}, e^{(k,l)}, \vec{w}_{\mathsf{pc}}^{(k,l)}, e_{\mathsf{pc}}^{(k,l)}, (\mathsf{x}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)}))(x)) \\
&= \sum_{x \in \{0,1\}^{\ell}} F'(G'((\mathsf{w}^{(k,l)}, \mathsf{w}_{\mathsf{pc}}^{(k,l)}), (\mathsf{x}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)}))(x))
\end{aligned}
\tag{37}
$$

Therefore, by Derivation (37), and Equations (33), (34), (35), and (36) we have that

$$(\mathsf{pp}, (F', G'), (T_{\gamma}^{(k,l)}, (\mathsf{u}^{(k,l)}, \mathsf{u}_{\mathsf{pc}}^{(k,l)}), (\mathsf{x}^{(k,l)}, \mathsf{x}_{\mathsf{pc}}^{(k,l)})), (\mathsf{w}^{(k,l)}, \mathsf{w}_{\mathsf{pc}}^{(k,l)})) \in \mathsf{SC}.$$

This means that for each challenge $\gamma^{(k)}$, and the corresponding input instance to the sum-check folding reduction

$$(\mathsf{pp}, (F', G'), \{(T_i^{(k)} + \gamma^{(k)} \cdot T_{\mathsf{pc},i}^{(k)}, (\mathsf{u}_i^{(k)}, \mathsf{u}_{\mathsf{pc},i}^{(k)}), (\mathsf{x}_i^{(k)}, \mathsf{x}_{\mathsf{pc},i}^{(k)}))\}_{i \in [n]}),$$

χ can produce a satisfying L-sub-tree of transcripts for the sum-check folding reduction with satisfying output instance-witness pairs in SC. Then, χ can run χ_{FSC} on each of these sub-trees to retrieve a corresponding satisfying input witness $\{(\mathsf{w}_i^{(k)}, \mathsf{w}_{\mathsf{pc},i}^{(k)})\}_{i \in [n]}$ such that

$$(\{(T_i^{(k)} + \gamma^{(k)} \cdot T_{\mathsf{pc},i}^{(k)}, (\mathsf{u}_i^{(k)}, \mathsf{u}_{\mathsf{pc},i}^{(k)}), (\mathsf{x}_i^{(k)}, \mathsf{x}_{\mathsf{pc},i}^{(k)}))\}_{i \in [n]}, \{(\mathsf{w}_i^{(k)}, \mathsf{w}_{\mathsf{pc},i}^{(k)})\}_{i \in [n]}) \in \mathsf{SC}^n$$

with respect to structure (F', G').

By the verifier's construction, we have that for all $i \in [n]$, $T_i^{(k)} = T_i$, $T_{\mathsf{pc},i}^{(k)} = T_{\mathsf{pc},i}$, $\mathsf{x}_i^{(k)} = \mathsf{x}_i$, $\mathsf{x}_{\mathsf{pc},i}^{(k)} = \mathsf{x}_{\mathsf{pc},i}$, $\mathsf{u}_i^{(k)} = \mathsf{u}_i$, and $\mathsf{u}_{\mathsf{pc},i}^{(k)} = \mathsf{u}_{\mathsf{pc},i}$. Then, by the binding property of the commitment scheme, for all $i \in [n]$ we must have that for any $k, k' \in [2]$ $\mathsf{w}_i^{(k)} = \mathsf{w}_i^{(k')}$ and $\mathsf{w}_{\mathsf{pc},i}^{(k)} = \mathsf{w}_{\mathsf{pc},i}^{(k')}$. We denote these values simply as w_i and $\mathsf{w}_{\mathsf{pc},i}$.

Then, for

$$(\vec{w}_i, e_i) \leftarrow \mathsf{w}_i$$
$$(\vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}) \leftarrow \mathsf{w}_{\mathsf{pc},i},$$

by the satisfiability condition of SC we have that

$$\begin{aligned}
&T_i + \gamma^{(k)} \cdot T_{\mathsf{pc},i} \\
&= \sum_{x \in \{0,1\}^\ell} F'(G'((\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i}))(x)) \\
&= \sum_{x \in \{0,1\}^\ell} F'(G'((\vec{w}_i, e_i, \vec{w}_{\mathsf{pc},i}, e_{\mathsf{pc},i}), (\mathsf{x}_i, \mathsf{x}_{\mathsf{pc},i}))(x)) \\
&= \sum_{x \in \{0,1\}^\ell} F'((G(\vec{w}_i, \mathsf{x}_i), h_{1,i}, h_{2,i}, G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}), h_{\mathsf{pc},1,i}, h_{\mathsf{pc},2,i})(x)) \\
&= \sum_{x \in \{0,1\}^\ell} h_{1,i}(x) \cdot h_{2,i}(x) \cdot F(G(\vec{w}_i, \mathsf{x}_i)(x)) + \\
&\quad \gamma^{(k)} \cdot \sum_{x \in \{0,1\}^\ell} h_{\mathsf{pc},1,i}(x) \cdot h_{\mathsf{pc},2,i}(x) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i})(x)) \\
&= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{1,i}(x_1) \cdot \widetilde{e}_{2,i}(x_2) \cdot F(G(\vec{w}_i, \mathsf{x}_i)(x)) + \\
&\quad \gamma^{(k)} \cdot \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{\mathsf{pc},1,i}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2,i}(x_2) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i})(x)).
\end{aligned}$$

Then, by interpolation, we have that

$$\begin{aligned}
T_i &= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{1,i}(x_1) \cdot \widetilde{e}_{2,i}(x_2) \cdot F(G(\vec{w}_i, \mathsf{x}_i)(x)) \\
T_{\mathsf{pc},i} &= \sum_{x \in \{0,1\}^\ell} \widetilde{e}_{\mathsf{pc},1,i}(x_1) \cdot \widetilde{e}_{\mathsf{pc},2,i}(x_2) \cdot F_{\mathsf{PC}}(G_{\mathsf{PC}}(\vec{w}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i})(x)).
\end{aligned}$$

This means that for $i \in [n]$

$$((T_i, \mathsf{u}_i, \mathsf{x}_i), \mathsf{w}_i) \in \mathsf{NSC}$$
$$((T_{\mathsf{pc},i}, \mathsf{u}_{\mathsf{pc},i}, \mathsf{x}_{\mathsf{pc},i}), \mathsf{w}_{\mathsf{pc},i}) \in \mathsf{NSC}_{\mathsf{PC}}.$$

Therefore, χ can compute a satisfying input witness $(\mathsf{w}_i, \mathsf{w}_{\mathsf{pc},i})$ for $i \in [n]$. □

B.4 Proof of Lemma 11 (From lookup to grand-product)

Lemma 22 (Completeness). *Construction 6 is complete.*

Proof. Consider an arbitrary PPT adversary $\mathcal{A}$. For arbitrary size parameters (n, m), let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, (n, m))$, and $\mathsf{pk}, \mathsf{vk} \leftarrow \mathcal{K}(\mathsf{pp})$. Suppose now that the adversary on input pp generates

$$((\overline{t}, \overline{a}, \overline{v}), (t, a, v)) \in \mathsf{LKP}.$$

The prover and verifier on input $(\mathsf{pk}, \mathsf{vk}), (\overline{t}, \overline{a}, \overline{v}), (t, a, v)$ interactively produce output instance-witness pairs

$$\{((p_k, \overline{v}_k), v_k)\}_{k \in [4]}$$

We must show that $\{((p_k, \overline{v}_k), v_k)\}_{k \in [4]} \in \mathsf{GP}^4$.

Indeed, by the linearity of the commitment scheme and by definitions of $G_n = \mathsf{Commit}(\mathsf{pp}, 1^n)$, $G_m = \mathsf{Commit}(\mathsf{pp}, 1^m)$, and $H = \mathsf{Commit}(\mathsf{pp}, (1, \ldots, n))$, we have that

$$\overline{v}_k = \mathsf{Commit}(\mathsf{pp}, v_k) \tag{38}$$

for $k \in [4]$.

Next, by the prover's computation of f and c and by Lemma 10, we have that

$$\left(\prod_{i \in [n]} (i + t_i \cdot X - Y)\right) \cdot \left(\prod_{i \in [m]} (a_i + v_i \cdot X + (c_i + 1) \cdot X^2 - Y)\right) =$$
$$\left(\prod_{i \in [m]} (a_i + v_i \cdot X + c_i \cdot X^2 - Y)\right) \cdot \left(\prod_{i \in [n]} (i + t_i \cdot X + f_i \cdot X^2 - Y)\right).$$

But, this means that for an arbitrary $r_X, r_Y \in \mathbb{F}$

$$\left(\prod_{i \in [n]} (i + t_i \cdot r_X - r_Y)\right) \cdot \left(\prod_{i \in [m]} (a_i + v_i \cdot r_X + (c_i + 1) \cdot r_X^2 - Y)\right) =$$
$$\left(\prod_{i \in [m]} (a_i + v_i \cdot r_X + c_i \cdot r_X^2 - r_Y)\right) \cdot \left(\prod_{i \in [n]} (i + t_i \cdot r_X + f_i \cdot r_X^2 - r_Y)\right).$$

Then, by the definition of $v_1, \ldots, v_4$ we have that

$$\prod_{i \in [n]} v_{1,i} \cdot \prod_{i \in [m]} v_{2,i} = \prod_{i \in [m]} v_{3,i} \cdot \prod_{i \in [n]} v_{4,i}.$$

Hence, by construction, we have that

$$p_k = \prod_i v_{k,i} \tag{39}$$

for $k \in [4]$, by substitution, we have that

$$p_1 \cdot p_2 = p_3 \cdot p_4.$$

Therefore, we have that the verifier's check in Step 5 passes.

Then, by Equations 38 and 39, we have that $\{((p_k, \overline{v}_k), v_k)\}_{k\in[4]} \in \mathsf{GP}^4$. □

Lemma 23 (Knowledge Soundness). *Construction 6 is knowledge sound.*

Proof. We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct a PPT extractor χ that outputs a satisfying input witness with probability $1-\mathsf{negl}(\lambda)$ given a tree of accepting transcripts and the corresponding output instance-witness pairs.

Indeed, for a sufficiently large $N = O(nm)$ we provide χ with an N-tree of accepting transcripts for the public parameter, instance tuple $(\mathsf{pp}, (\overline{t}, \overline{a}, \overline{v}))$. For $l \in [N]$, let $r_X^{(l)}$ and $r_Y^{(l)}$ denote the verifier's challenges in Step 2. Let

$$\{((p_k^{(l)}, \overline{v}_k^{(l)}), v_k^{(l)})\}_{k\in[4]} \in \mathsf{GP}^4 \tag{40}$$

denote the corresponding output instance-witness pairs.

Using an arbitrary but sufficiently large subsets of transcripts, the extractor χ interpolates t such that for $i \in [n]$

$$v_{1,i}^{(l)} = i + t_i \cdot r_X^{(l)} - r_Y^{(l)}, \tag{41}$$

interpolates a, v, and c such that for $i \in [m]$

$$v_{2,i}^{(l)} = a_i + v_i \cdot r_X^{(l)} + (c_i + 1) \cdot {r_X^{(l)}}^2 - r_Y^{(l)}, \tag{42}$$

interpolates a', v', and c' such that for $i \in [m]$

$$v_{3,i}^{(l)} = a'_i + v'_i \cdot r_X^{(l)} + c'_i \cdot {r_X^{(l)}}^2 - r_Y^{(l)} \tag{43}$$

interpolates t' and f such that for $i \in [n]$

$$v_{4,i}^{(l)} = i + t'_i \cdot r_X^{(l)} + f_i \cdot {r_X^{(l)}}^2 - r_Y^{(l)}. \tag{44}$$

We will now argue that regardless of the extractors choice of transcripts, due to the binding property of the commitment scheme, it will derive the same interpolated vectors t, a, v, c, and f. Indeed, starting with t, regardless of the extractors choice of $l \in [N]$ to derive t, we must have that

$$\begin{aligned}
\mathsf{Commit}(\mathsf{pp}, t) &= \mathsf{Commit}(\mathsf{pp}, (v_1^{(l)} - (1, \ldots, n) + (r_Y^{(l)}, \ldots, r_Y^{(l)})) \cdot {r_X^{(l)}}^{-1}) \\
&= (\overline{v}_1^{(l)} - H + G_n \cdot r_Y^{(l)}) \cdot {r_X^{(l)}}^{-1} \\
&= \overline{t}
\end{aligned}$$

Therefore, because $\overline{t}$ is identical across all transcripts, by the binding property of the commitment scheme, we have that t is identically derived regardless of the choice of transcripts. For similar reasons, because $\overline{a}$, $\overline{v}$, $\overline{c}$, and $\overline{f}$ are identical across transcripts we must have that a, v, c, and f are identically derived regardless of the choice of transcripts, and moreover that $t = t'$, $a = a'$, $v = v'$ and $c = c'$. Then, we must have that Equations (41), (42) (43) (44) hold for all $l \in [N]$.

Then, by Precondition (40), and the verifier's check in Step (5), we have for all $l \in [N]$

$$\left(\prod_{i\in[n]} (i + t_i \cdot r_X^{(l)} - r_Y^{(l)})\right) \cdot \left(\prod_{i\in[m]} (a_i + v_i \cdot r_X^{(l)} + (c_i + 1) \cdot {r_X^{(l)}}^2 - r_Y^{(l)})\right)$$
$$= \prod_{i\in[n]} v_{1,i}^{(l)} \cdot \prod_{i\in[m]} v_{2,i}^{(l)}$$
$$= p_1^{(l)} \cdot p_2^{(l)}$$
$$= p_3^{(l)} \cdot p_4^{(l)}$$
$$= \prod_{i\in[n]} v_{3,i}^{(l)} \cdot \prod_{i\in[m]} v_{4,i}^{(l)} =$$
$$\left(\prod_{i\in[m]} (a_i + v_i \cdot r_X^{(l)} + c_i \cdot {r_X^{(l)}}^2 - r_Y^{(l)})\right) \cdot \left(\prod_{i\in[n]} (i + t_i \cdot r_X^{(l)} + f_i \cdot {r_X^{(l)}}^2 - r_Y^{(l)})\right).$$

Therefore, by the precondition that N is sufficiently large, by interpolation, we have that

$$\left(\prod_{i\in[n]} (i + t_i \cdot X - Y)\right) \cdot \left(\prod_{i\in[m]} (a_i + v_i \cdot X + (c_i + 1) \cdot X^2 - Y)\right) =$$
$$\left(\prod_{i\in[m]} (a_i + v_i \cdot X + c_i \cdot X^2 - Y)\right) \cdot \left(\prod_{i\in[n]} (i + t_i \cdot X + f_i \cdot X^2 - Y)\right).$$

Then, by Lemma 10, and by the derivation that $\mathsf{Commit}(\mathsf{pp}, t) = \overline{t}$ (and the similar implied derivations for a and v), we have that

$$(\mathsf{pp}, (\overline{t}, \overline{a}, \overline{v}), (t, a, v)) \in \mathsf{LKP}.$$

□

B.5 Proof of Lemma 9 (From grand-product to zero-check)

Lemma 24 (Completeness). *Construction 5 is complete.*

Proof. Consider an arbitrary PPT adversary $\mathcal{A}$. For an arbitrary size parameter n, let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, and $(\mathsf{pk}, \mathsf{vk}, (F_{\mathsf{GP}}, G_{\mathsf{GP}})) \leftarrow \mathcal{K}(\mathsf{pp})$. Suppose now that the

adversary on input pp generates

$$((p, \overline{v}), v) \in \mathsf{GP}. \tag{45}$$

The prover and verifier on input $(\mathsf{pk}, \mathsf{vk}), (p, \overline{v}), v$ interactively produce output instance-witness pairs $(((\overline{v}, \overline{v}'), p), (v, v'))$. We must show that

$$((F_{\mathsf{GP}}, G_{\mathsf{GP}}), ((\overline{v}, \overline{v}'), p), (v, v')) \in \mathsf{ZC}. \tag{46}$$

Indeed, by Precondition 45 and the prover's computation in Step 1, we have that

$$\begin{aligned} \overline{v} &= \mathsf{Commit}(\mathsf{pp}, v) \\ \overline{v}' &= \mathsf{Commit}(\mathsf{pp}, v') \end{aligned}$$

By Lemma 8 we have that

$$\begin{aligned} f(0, x) &= \widetilde{v}(x) \\ f(1, \ldots, 1, 0) &= p, \end{aligned}$$

and that for all $x \in \{0,1\}^{\log n}$

$$f(1, x) = f(x, 0) \cdot f(x, 1).$$

Then, for all $x \in \{0,1\}^{\log n}$, we have that

$$\begin{aligned} 0 &= f(1, x) - f(x, 0) \cdot f(x, 1) \\ &= \widetilde{(v, v')}(1, x) - \widetilde{(v, v')}(x, 0) \cdot \widetilde{(v, v')}(x, 1) \\ &= (v, \widetilde{(v'_1, \ldots, p, v'_n)})(1, x) - (v, \widetilde{(v'_1, \ldots, p, v'_n)})(x, 0) \cdot (v, \widetilde{(v'_1, \ldots, p, v'_n)})(x, 1) \\ &= F_{\mathsf{GP}}(G_{\mathsf{GP}}((v, v'), p)(x)) \end{aligned}$$

Therefore, we have that Equation 46 holds. □

Lemma 25 (Knowledge soundness). *Construction 5 is knowledge-sound*

Proof. We prove knowledge soundness via tree extraction (Lemma 15). That is, we construct a PPT extractor χ, that can produce a satisfying input witness given as input a single transcript for public parameters pp and input instance $(p, \overline{v})$, and output instance-witness pair

$$((F_{\mathsf{GP}}, G_{\mathsf{GP}}), ((\overline{v}, \overline{v}'), p), (v, v')) \in \mathsf{ZC}. \tag{47}$$

Indeed, by Precondition (47), we have that for all $x \in \{0,1\}^{\log n}$

$$\begin{aligned} 0 &= F_{\mathsf{GP}}(G_{\mathsf{GP}}((v, v'), p)(x)) \\ &= (v, \widetilde{(v'_1, \ldots, p, v'_n)})(1, x) - (v, \widetilde{(v'_1, \ldots, p, v'_n)})(x, 0) \cdot (v, \widetilde{(v'_1, \ldots, p, v'_n)})(x, 1) \\ &= f(1, x) - f(x, 0) \cdot f(x, 1) \end{aligned}$$

for $f = (v, (\widetilde{v'_1, \ldots, p}, v'_n))$. But this means that $f(1, x) = f(x, 0) \cdot f(x, 1)$ for all $x \in \{0,1\}^{\log n}$, $f(1, \ldots, 1, 0) = p$, and $f(0, x) = \widetilde{v}(x)$. Then, by Lemma 8, we must have that $p = \prod_{i \in [n]} v_i$. Moreover, by Precondition (47), we have that $\mathsf{Commit}(\mathsf{pp}, v) = \overline{v}$. Therefore, we have that

$$((p, \overline{v}), v) \in \mathsf{GP}.$$

□