HyperNova: Recursive arguments for customizable constraint systems

Abhiram Kothapalli[†] Srinath Setty[⋆]

[†]Carnegie Mellon University [⋆]Microsoft Research

Abstract.

This paper introduces *HyperNova*, a recursive argument for proving incremental computations whose steps are expressed with CCS (Setty et al. ePrint 2023/552), a customizable constraint system that simultaneously generalizes Plonkish, R1CS, and AIR without overheads. A distinguishing aspect of HyperNova is that the prover's cost at each step is dominated by a *single* multi-scalar multiplication (MSM) of size equal to the number of variables in the constraint system, which is optimal when using an MSM-based commitment scheme.

To construct HyperNova, we generalize folding schemes (CRYPTO 22) to allow instances from two (potentially) different NP relations, that share a *compatible* structure, to be folded; we refer to this generalization as *multi-folding schemes*. Furthermore, we devise a public-coin multi-folding scheme for instances in CCS and *linearized CCS* (a variant of CCS that we introduce). This construction can be viewed as an "early stopping" version of Spartan (CRYPTO 20), applied to a carefully-crafted polynomial that includes claims about prior linearized CCS instances. The prover's work in the multi-folding scheme is a linear number of finite field operations and the verifier's work is a logarithmic number of finite field operations and a single group scalar multiplication. We then construct incrementally verifiable computation (IVC) from non-interactive multi-folding schemes with the lowest prover costs. We also provide an alternate realization of HyperNova with a black box use of Nova, which nearly eliminates the need for deferred arithmetic when instantiated with a cycle of elliptic curves.

As an additional contribution, we describe nlookup, a lookup argument, that is particularly suited for recursive arguments based on folding schemes. Specifically, at a particular step in an incremental computation, for m lookups into a table of size n ($m << n$), the prover's work is dominated by $O(n)$ finite field operations and it requires only $O(m \cdot \log n)$ degree-2 constraints and $O(\log n)$ hash evaluations in the incremental computation. nlookup is currently not suitable for efficiently encoding bitwise operations, but it provides a powerful tool for efficiently encoding (large) finite state machines and proving their transitions with recursive arguments.

1 Introduction

This paper continues a recent line of work on folding schemes (e.g., Nova [KST22], SuperNova [KS22]) to build prover-efficient succinct arguments. There are two slightly different ways of interpreting results from this line of work. First, they provide the fastest succinct argument for semi-structured, non-deterministic computations where the computation to be proven can be decomposed into multiple steps (different steps need not have the same circuit representation) [Dra22]. Second, they provide the most efficient recursive argument scheme, with the lowest recursion overheads.

(Super)Nova's computational model admits incremental computations where a particular step invokes one of the pre-defined "instructions". This model captures various semi-structured computational statements that arise frequently in practice. For example, in a verifiable delay function (VDF), there is only one instruction and it verifies that a certain iterations of a delay function (e.g., MinRoot [KMT22]) is executed correctly. As a more complex example, in the so-called "zero-knowledge virtual machines" (zkVMs), at each step, the incremental computation invokes one of the pre-defined instructions. Concretely, in the case of Ethereum's virtual machine (EVM), an instruction might perform operations like add/mul, or an elliptic curve scalar multiplication over the BN-254 curve.

To prove such incremental computations, Nova provides an inexpensive mechanism, called a folding scheme, to (recursively) transform the task of proving N steps of a program execution into the task of proving a single step of the program execution. It then applies a general-purpose zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) to prove that single step. This approach builds on a recent line of work to achieve efficient recursive arguments [BCTV14,BGH19,BCMS20,BDFG21,BCL$^+$21], but it differs from them in that Nova does not rely on zkSNARKs or even arguments in general.

Compared to directly employing a general-purpose zkSNARK, built from from polynomial IOPs and polynomial commitment schemes (e.g., Spartan [Set20], Plonk [GWC19], Marlin [CHM$^+$20], HyperPlonk [CBBZ23]), to prove the entire incremental computation, Nova's approach is substantially cheaper (as long as each step is sufficiently large, to offset recursion overheads). Specifically, at each incremental step, Nova's prover incurs only *two* MSMs of size proportional to the size of the circuit proven. Whereas, general-purpose zkSNARKs need many more MSMs. For example, Marlin [CHM$^+$20, Fig 1] reports 22 MSMs and many more FFTs of size proportional to the circuit size.

In addition, by design, Nova's proof generation is incremental (i.e., it produces a proof for each step and then uses its recursion capabilities to produce a single proof), so it can be more easily distributed and parallelized than with a non-recursive zkSNARK where one must unroll program executions into monolithic circuits. The latter rules out applications where one cannot statically unroll program executions (e.g., VDF) or makes it inconvenient (e.g., program executions on machines such as EVM or RISC-V). As presented, Nova does not immediately support parallel proof generation. There exists a generic compiler [BCCT13] to transform constructions such as Nova to support parallel proving.

Problem statement. In Nova, each step of an incremental computation is expressed with R1CS, an NP-complete problem that generalizes arithmetic circuit satisfiability and is implicit in the QAPs formalism [GGPR13]. In practice, when zkSNARKs (e.g., Plonk) are applied to prove program executions, practitioners use custom constraint systems (e.g., Plonkish) that are tailored to a particular classes of applications. Specifically, Plonkish constraints are multivariate high-degree polynomials. Whereas, R1CS is restricted to checking quadratic constraints in a specific form. A key question is whether one can build a recursive argument for Plonkish, with Nova-like performance characteristics. In particular, our goal is to prove CCS [STW23], a customizable constraint system that simultaneously generalizes Plonkish, R1CS, and AIR without overheads.

In more detail, for any solution that handles high-degree constraints, we require the prover's cryptographic work to be independent of the degree of constraints supported. That is, the number of MSMs (or their sizes) performed by the prover must not depend on the degree of the supported constraints. This requirement is crucial because otherwise one can simply lower the degree of constraints to two and use Nova rather than attempting to use high-degree constraints.

Prior approaches and challenges.

Halo2 [BGH20] replaces the polynomial IOP in Halo [BGH19] from Sonic [MBKM19] to Plonk [GWC19], and its widely used implementation supports Plonkish. Unfortunately, it incurs substantial prover costs as the prover must necessarily produce a SNARK using Plonk at each step of the program execution. Furthermore, the prover incurs $O(n \cdot d)$ cryptographic operations, where n is the size of the circuit at each step and d is the maximum degree of constraints proven. Switching from Plonk to HyperPlonk [CBBZ23] would reduce the prover's cryptographic operations to $O(n)$, but it does not avoid the need to produce a SNARK.

Split accumulation [BCL$^+$21] reduces the recursion overheads of Halo-type approaches. It also avoids the need for succinct arguments to construct recursive SNARKs. Unfortunately, the existing construction of recursive SNARKs from split accumulation [BCL$^+$21] is limited to R1CS. It is not clear how to extend their approach to handle Plonkish without making the prover incur $O(n \cdot d)$ cryptographic operations, which, as noted above, is undesirable.

Sangria [Moh23] shows that Nova's folding scheme for R1CS can be easily adapted to handle Plonkish. However, the number of cross-terms that the prover must commit to increases linearly with d. Furthermore, the prover must incur $O(n \cdot d)$ cryptographic operations to commit to $O(d)$ cross-terms. As a result, in general, it is cheaper to lower the constraints to degree-2 and prove them with Nova than use higher-degree constraints with Sangria. For instance, a single iteration of the MinRoot [KMT22] can be represented with a single degree-5 constraint in Plonkish [Zha23]. Whereas, with R1CS, one would need three constraints.

However, if Sangria is applied to prove MinRoot in degree-5 Plonkish, it incurs more work for the prover than when Nova is applied to prove MinRoot in R1CS.

1.1 Our approach: HyperNova

This section provides a high-level overview of HyperNova's ingredients. As noted earlier, HyperNova's target is to prove incremental computations where each step of the incremental computation is expressed with CCS [STW23]. However, if we naively build a folding scheme for CCS, perhaps for a "relaxed" variant of CCS (analogous to relaxed R1CS in Nova), it will have the efficiency issues noted above for Sangria.

To avoid those issues, HyperNova takes a different approach that involves leveraging the power of the sum-check protocol [LFKN90], as done in Spartan [Set20] and its recent extension to handle CCS called SuperSpartan [STW23]. In particular, we provide two instantiations, one that directly builds HyperNova, and another that uses Nova as a black box. Both variants can be instantiated efficiently using a cycle of elliptic curves and they both have the same prover efficiency. The second variant however has minimal "deferred arithmetic" (i.e., arithmetic that must be passed to the other curve in the curve cycle for efficiency) that shows up when implementing recursive arguments using cycles of elliptic curves.

Remark 1. Given that CCS is a strict generalization of Plonkish, we focus on describing HyperNova for CCS. HyperNova for CCS can be naturally translated to HyperNova for Plonkish, with the same efficiency guarantees.

(1) Multi-folding schemes. To construct HyperNova, we introduce a generalization of folding schemes, and we refer to it as *multi-folding schemes*. Recall that a folding scheme for a relation $\mathcal{R}$ is a protocol between a *prover* and *verifier* in which the prover and the verifier reduce the task of checking two instances in $\mathcal{R}$ with the same structure s into the task of checking a single instance in $\mathcal{R}$ with structure s. A multi-folding scheme is defined with respect to a pair of relations $(\mathcal{R}_1, \mathcal{R}_2)$. It is an interactive protocol between a prover and a verifier in which the prover and the verifier reduce the task of checking an instance in $\mathcal{R}_1$ with structure s_1 and an instance in $\mathcal{R}_2$ with structure s_2 into the task of checking a single instance in $\mathcal{R}_1$ with structure s_1—as long as $(\mathsf{s}_1, \mathsf{s}_2)$ satisfy a pre-defined predicate (e.g., that the two structures are equal). Below, we clarify how this generalization unlocks additional power for constructing IVC.

Remark 2. Multi-folding schemes are related to split-accumulation schemes [BCL+21]. However, multi-folding schemes use the language of NP relations and are specified in the standard interactive model. For example, our definitions do not require an auxiliary decider algorithm and the associated complexity. Furthermore, by using the language of NP relations, the split between a witness and instance is immediate in the context of folding schemes and multi-folding schemes. Overall, our definitions, security notions, and proofs are conceptually simpler.

(2) A multi-folding scheme for CCS. Our starting point to construct a multi-folding scheme for CCS is the following observation. We observe that the Spartan proof system [Set20] (more specifically its generalization to handle CCS called SuperSpartan [STW23]) transforms the task of checking the satisfiability of a CCS instance into the task of checking if a multivariate polynomial g of total degree $d+1$, where d is the degree of the CCS constraints, sums to zero over a suitable Boolean hypercube. Spartan then invokes the sum-check protocol [LFKN90] to prove that claim about g. At the end of the sum-check invocation, the prover and the verifier are left with checking certain claims. Fortunately, these claims concern a restricted form of CCS (we formalize this restricted form of CCS and refer to it as *linearized CCS*). Note Spartan then proves those claims about the restricted form of CCS with an additional invocation of the sum-check protocol.

While an "early stopping" version of Spartan (the one with a single invocation of the sum-check protocol) provides a reduction of knowledge [KP23] from CCS to linearized CCS, it is not a folding (or a multi-folding) scheme. So, our second idea is to redefine the polynomial g to additionally include claims from a running linearized CCS instance using a random challenge from the verifier. This is possible as long as the running instance and the CCS instance that is being folded share a *compatible* structure (e.g., the same CCS matrices).

As a result, we obtain a public-coin multi-folding scheme that can fold a CCS instance with structure s into a linearized CCS instance with structure s to get a new linearized CCS instance with structure s. The prover's work in the folding scheme is a linear number of finite field operations — notably, avoiding commitments to cross-terms *altogether* — and the verifier's work is a logarithmic number of finite field operations. Since the multi-folding scheme is public coin, we make it non-interactive in the random oracle model using the Fiat-Shamir transform [FS86] and heuristically instantiate it in the plain model using a concrete cryptographic hash function.

Moreover, by leveraging linearity, our multi-folding scheme for CCS can be easily generalized to take an arbitrary number of CCS instances and an arbitrary number of linearized CCS instances that all share the same structure and output a single linearized CCS instance.

(3) IVC and its generalizations from multi-folding schemes. Recall that Nova constructs IVC [Val08] from non-interactive folding schemes. Similarly, we show how to construct an IVC scheme from non-interactive multi-folding schemes such as the non-interactive version of the multi-folding scheme for CCS discussed above. Our construction and security proofs follow the same blueprint. The prover initializes a "default" running instance. Then, at each step of the incremental computation, the prover commits to the witness for an augmented CCS instance (the augmented CCS not only runs a step of the incremental computation, but it also runs the verifier of the non-interactive multi-folding scheme). The prover then runs the non-interactive multi-folding scheme to fold that CCS instance into a running instance. The prover then sends the CCS instance and the output of the multi-folding instance as input to the next invocation of the augmented circuit, to prove the next step of the incremental computation.

As in Nova, in the "end" (or at any point in the incremental computation), the prover is left with a running instance U (which in the case of HyperNova is a linearized CCS instance) and a CCS instance u that represents the last step of the incremental computation augmented with a verifier circuit. The prover folds U and u to get U' with an invocation of the multi-folding scheme for CCS, and suppose that the prover's output in the folding scheme is π_{fs}. HyperNova's IVC proof is then $\pi = (\mathsf{U}, \mathsf{u}, \pi_{\mathsf{fs}}, w)$ where w is a purported witness to U'. HyperNova's prover can then send π to a verifier, but the size of the proof is linear in the size of the step of an incremental computation. To avoid this, as in Nova, HyperNova's prover can use a general zkSNARK to prove the knowledge of a valid π. In particular, Spartan excluding the first sum-check invocation provides a necessary zkSNARK. Of course, other proof systems can also be used as long as they can prove the knowledge of a valid π.

Remark 3. Because a multi-folding scheme folds an arbitrary number of running instances incoming instances into a single running instance, it affords a natural generalization of IVC [Val08] called proof-carrying data [BCCT13] using the approach of Bünz et al. [BCL^{+}21]. We focus on IVC for its conceptual simplicity.

(4) A construction of IVC with a black box use of Nova. In addition to the direct construction of IVC from a non-interactive multi-folding scheme for CCS, we describe an approach that leverages Nova as a black box. In particular, the high-level idea is to express the verifier of the non-interactive multi-folding scheme for CCS as a step circuit in Nova (i.e., with R1CS). The step circuit, as part of its public IO, stores a running instance (which in our case is a linearized CCS instance). At each step, the prover first represents a step of the incremental computation as a CCS instance u. It folds that into a running instance U to obtain a new running instance U' (and the associated witnesses) and a proof π_{fs}. The prover then runs Nova where the step circuit takes as non-deterministic input $(\mathsf{u}, \pi_{\mathsf{fs}})$. The step circuit then folds u into a running instance that it tracks via its public IO using π_{fs}. In the "end" or at any point in the incremental computation, a verifier can verify the Nova proof associated with the incremental computation and check that the running instance in the public IO of the step circuit is indeed satisfying using a purported witness from the prover. As with the direct construction discussed above, the prover can compress such an IVC proof into a succinct proof using a general zkSNARK.

The primary advantage of this version of HyperNova is that it only needs to implement a step circuit. Furthermore, much of the work in representing the verifier of the non-interactive folding scheme is a logarithmic number of finite field operations and hash evaluations, both of which can be encoded natively and efficiently with R1CS (e.g., with a SNARK-friendly hash function). The step circuit will have to perform a *single* group scalar multiplication, but when using a cycle of elliptic curves, this task can be easily (and with minimal public IO) be deferred to the other curve in the curve cycle.

2 Preliminaries

We use λ to denote the security parameter and $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p). We use $\mathsf{negl}(\lambda)$ to denote a negligible function in λ. Throughout the paper, the depicted asymptotics depend on λ, but we elide this for brevity. We use "PPT algorithms" to refer to probabilistic polynomial time algorithms. 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.

2.1 Polynomials and low-degree extensions

We adapt this subsection from prior work [Set20]. We start by recalling several facts about polynomials.

Definition 1 (Multilinear polynomial). *A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.*

Definition 2 (Low-degree polynomial). *A multivariate polynomial g over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree of g in each variable is exponentially smaller than $|\mathbb{F}|$.*

Low-degree extensions (LDEs). Suppose $g : \{0,1\}^\ell \to \mathbb{F}$ is a function that maps ℓ-bit elements into an element of $\mathbb{F}$. A *polynomial extension* of g is a low-degree ℓ-variate polynomial, denoted $\widetilde{g}$, such that $\widetilde{g}(x) = g(x)$ for all $x \in \{0,1\}^\ell$.

A *multilinear* polynomial extension (or simply, a multilinear extension, or MLE) is a low-degree polynomial extension where the extension is a multilinear polynomial (i.e., the degree of each variable in $\widetilde{g}$ is at most one). Given a function $Z : \{0,1\}^\ell \to \mathbb{F}$, the multilinear extension of Z is the unique multilinear polynomial $\widetilde{Z} : \mathbb{F}^\ell \to \mathbb{F}$. It can be computed as follows.

$$\begin{aligned}
\widetilde{Z}(x_1, \ldots, x_\ell) &= \sum_{e \in \{0,1\}^\ell} Z(e) \cdot \prod_{i=1}^{\ell} (x_i \cdot e_i + (1 - x_i) \cdot (1 - e_i)) \\
&= \sum_{e \in \{0,1\}^\ell} Z(e) \cdot \widetilde{eq}(x, e) \\
&= \langle (Z(0), \ldots, Z(2^\ell - 1)), (\widetilde{eq}(x, 0), \ldots, \widetilde{eq}(x, 2^\ell - 1) \rangle
\end{aligned}$$

Note that $\widetilde{eq}(x, e) = \prod_{i=1}^{\ell} (e_i \cdot x_i + (1 - e_i) \cdot (1 - x_i))$, which is the MLE of the following function:

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

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

Dense representation for multilinear polynomials. Since the MLE of a function is unique, it offers the following method to represent any multilinear polynomial. Given a multilinear polynomial $g : \mathbb{F}^\ell \to \mathbb{F}$, it can be represented uniquely by the list of tuples L such that for all $i \in \{0,1\}^\ell$, $(\mathsf{to\text{-}field}(i), g(i)) \in L$ if and only if $g(i) \neq 0$, where $\mathsf{to\text{-}field}$ is the canonical injection from $\{0,1\}^\ell$ to $\mathbb{F}$. We denote such a representation of g as $\mathsf{DenseRepr}(g)$.

Definition 3. *A multilinear polynomial g in ℓ variables is a sparse multilinear polynomial if $|\mathsf{DenseRepr}(g)|$ is sub-linear in $O(2^\ell)$. Otherwise, it is a dense multilinear polynomial.*

As an example, suppose $g : \mathbb{F}^{2s} \to \mathbb{F}$. Suppose $|\mathsf{DenseRepr}(g)| = O(2^s)$, then $g(\cdot)$ is a sparse multilinear polynomial because $O(2^s)$ is sublinear in $O(2^{2s})$.

Lemma 1 (Sums over evaluations). *Consider size $\ell \in \mathbb{N}$. For multilinear polynomial $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ we have that*

$$P(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{eq}(X, x) \cdot P(x).$$

Proof. Let $Q(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{eq}(X, x) \cdot P(x)$ By the definition of $\widetilde{eq}$, we have that

$$P(x) = Q(x)$$

for all $x \in \{0,1\}^\ell$. However, because $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ is multilinear it is completely determined by 2^ℓ evaluation points. The same is holds for Q. Because P and Q agree on 2^ℓ points, they must be the same polynomial. □

Lemma 2 (Schwartz-Zippel). *let $g : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variate polynomial of total degree at most d. Then, on any finite set $S \subseteq \mathbb{F}$,*

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

2.2 The sum-check protocol

Suppose there is an ℓ-variate low-degree polynomial, g, where the degree of each variable in g is at most d. Suppose that a verifier $\mathcal{V}$ is interested in checking a claim of the following form by an untrusted prover $\mathcal{P}$:

$$T = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_\ell \in \{0,1\}} g(x_1, x_2, \ldots, x_\ell)$$

Of course, given g, $\mathcal{V}$ can deterministically evaluate the above sum and verify whether the sum is T. But, this computation takes time exponential in ℓ. Lund et al. [LFKN90] describe the sum-check protocol that requires far less computation on $\mathcal{V}$'s behalf, but provides a probabilistic guarantee. In the protocol, $\mathcal{V}$ takes as input randomness $r \in \mathbb{F}^\ell$ and interacts with $\mathcal{P}$ over a sequence of ℓ rounds. At the end of this interaction, $\mathcal{V}$ outputs a claim about the evaluation $g(r)$. Let $\langle \mathcal{P}, \mathcal{V}(r) \rangle$ denote the interaction between the prover and verifier with verifier randomness r. We treat $\langle \mathcal{P}, \mathcal{V}(r) \rangle$ as a function that takes prover and verifier input (g, ℓ, d, T) and outputs the claimed evaluation to be checked.

For any ℓ-variate polynomial g with degree at most d in each variable, the sum-check protocol satisfies the following properties.

1. Completeness: If $T = \sum_{x \in \{0,1\}^\ell} g(x)$, then for all $r \in \mathbb{F}^\ell$,
$$\Pr_r[\langle \mathcal{P}, \mathcal{V}(r) \rangle (g, \ell, d, T) = g(r)] = 1.$$
2. Soundness: If $T \neq \sum_{x \in \{0,1\}^\ell} g(x)$, then for any $\mathcal{P}^\star$ and for all $r \in \mathbb{F}^\ell$,
$$\Pr_r[\langle \mathcal{P}^\star, \mathcal{V}(r) \rangle (g, \ell, d, T) = c \wedge g(r) = c] \leq \ell \cdot d / |\mathbb{F}|.$$
3. Succinctness: The communication cost is $O(\ell \cdot d)$ elements of $\mathbb{F}$.

2.3 Arguments of knowledge and SNARKs

We adapt the definition provided in [KST22].

Definition 4. *Consider a relation $\mathcal{R}$ over public parameters, structure, instance, and witness tuples. A non-interactive argument of knowledge for $\mathcal{R}$ consists of PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$, denoting the generator, the prover, the verifier and the encoder respectively with the following interface.*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: On input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: On input structure* s*, representing common structure among instances, outputs the prover key* pk *and verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, u, w) \rightarrow \pi$*: On input instance u and witness w, outputs a proof π proving that* $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
- $\mathcal{V}(\mathsf{vk}, u, \pi) \rightarrow \{0,1\}$*: On input the verifier key* vk*, instance u, and a proof π, outputs 1 if the instance is accepting and 0 otherwise.*

A non-interactive argument of knowledge satisfies the following properties.

1. *Completeness: If for any PPT adversary $\mathcal{A}$*
$$\Pr\left[\mathcal{V}(\mathsf{vk}, u, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{s}, (u, w)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, u, w) \end{array} \right] = 1.$$

2. *Knowledge Soundness: if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that for all randomness* r

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

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

2.4 Incrementally Verifiable Computation

Definition 5 (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*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: on input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: on input public parameters* pp*, and polynomial-time function F, deterministically produces a prover key* pk *and a verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \rightarrow \Pi_{i+1}$*: 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 $z_{i+1} = F(z_i, \omega_i)$.*
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \rightarrow \{0, 1\}$*: 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 $\mathcal{A}$*

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

where F is a polynomial-time computable function.

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 r

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

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

2.5 Polynomial commitment scheme

We adapt the following definition from [BFS20].

Definition 6. *An extractable polynomial commitment scheme for multilinear polynomials over finite field $\mathbb{F}$ is a tuple of four protocols $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$:*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell)$*: takes as input ℓ (the number of variables in a multivariate polynomial); produces public parameters* pp*.*
- $\mathcal{C} \leftarrow \mathsf{Commit}(\mathsf{pp}, g)$*: takes as input a ℓ-variate multilinear polynomial $g \in \mathbb{F}[X_1, \ldots, X_\ell]$; produces a commitment $\mathcal{C}$.*
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g)$*: verifies the opening of commitment $\mathcal{C}$ to the ℓ-variate multilinear polynomial $g \in \mathbb{F}[X_1, \ldots, X_\ell]$; outputs $b \in \{0, 1\}$.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathcal{C}, r, v, \ell, g)$ *is a protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}$, the number of variables ℓ, a scalar $v \in \mathbb{F}$, and $r \in \mathbb{F}^\ell$. $\mathcal{P}$ additionally knows an ℓ-variate multilinear polynomial $g \in \mathbb{F}[\ell]$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $g(r) = v$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0, 1\}$.*

An extractable polynomial commitment scheme $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ for multilinear polynomials over a finite field $\mathbb{F}$ must satisfy the following conditions.

1. *Completeness: For any ℓ-variate multilinear polynomial $g \in \mathbb{F}[X_1, \ldots, X_\ell]$,*

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, \mathcal{C}, r, g(r), \ell, g) = 1 \,\middle|\, \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell), \\ \mathcal{C} \leftarrow \mathsf{Commit}(\mathsf{pp}, g) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

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

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell), \\ b_0 = b_1 \neq 0, & (g_0, g_1) \in \mathbb{F}^1[X_1, \ldots, X_\ell], \mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp}), \\ g_0 \neq g_1 & b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g_0), \\ & b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, g_1) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

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

$$\mathcal{R}_{\mathit{Eval}}(\mathsf{pp}) = \left\{ ((\mathcal{C}, r, v), g) \;\middle|\; \begin{array}{l} g \in \mathbb{F}^1[X_1, \ldots, X_\ell], \\ g(r) = v, \\ \mathit{Open}(\mathsf{pp}, \mathcal{C}, g) = 1 \end{array} \right\}$$

Definition 7. *A polynomial commitment scheme for multilinear polynomials* $\mathit{PC} = (\mathit{Gen}, \mathit{Commit}, \mathit{Open}, \mathit{Eval})$ *is additively homomorphic if for all* ℓ *and* $\mathsf{pp} \leftarrow \mathit{Gen}(1^\lambda, \ell)$*, and for any* $g_1, g_2 \in \mathbb{F}^1[X_1, \ldots, X_\ell]$*,*

$$\mathit{Commit}(\mathsf{pp}, g_1) + \mathit{Commit}(\mathsf{pp}, g_2) = \mathit{Commit}(\mathsf{pp}, g_1 + g_2).$$

3 Multi-folding schemes

Recall that a folding scheme [KST22] for a relation $\mathcal{R}$ is a protocol between a *prover* and *verifier* in which the prover and the verifier reduce the task of checking two instances in $\mathcal{R}$ with the same structure s into the task of checking a single instance in $\mathcal{R}$ with structure s.

We introduce a generalization of folding schemes, which we refer to as *multi-folding schemes*. A multi-folding scheme is defined with respect to a pair of relations $(\mathcal{R}_1, \mathcal{R}_2)$, a predicate compat, and size parameters μ and ν. It is an interactive protocol between a prover and a verifier in which the prover and the verifier reduce the task of checking a collection of μ instances in $\mathcal{R}_1$ with structure s_1 and a collection of ν instances in $\mathcal{R}_2$ with structure s_2 into the task of checking a single instance in $\mathcal{R}_1$ with structure s_1—as long as s_1 and s_2 satisfy a predicate compat (e.g., compat might require that $\mathsf{s}_1 = \mathsf{s}_2$). Below, we formally define multi-folding schemes.

Definition 8 (Multi-folding schemes). *Consider relations* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *over public parameters, structure, instance, and witness tuples, a predicate* compat *that structures for instances in* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *must satisfy, and size parameters* $\mu, \nu \in \mathbb{N}$*. A multi-folding scheme for* $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ *consists of a PPT generator algorithm* $\mathcal{G}$*, a deterministic encoder algorithm* $\mathcal{K}$*, and a pair of PPT algorithms* $\mathcal{P}$ *and* $\mathcal{V}$ *denoting the prover and the verifier respectively, with the following interface:*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$*: on input security parameter* λ*, samples public parameters* pp*.*
- $\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: on input* pp*, and structures* s_1 *and* s_2 *among the instances to be folded, outputs a prover key* pk *and a verifier key* vk*.*
- $\mathcal{P}(\mathsf{pk}, (\vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}), (\vec{\mathsf{u}_2}, \vec{\mathsf{w}_2})) \rightarrow (\mathsf{u}, \mathsf{w})$*: on input a vector of instances* $\vec{\mathsf{u}_1}$ *in* $\mathcal{R}_1$ *of size* μ *with structure* s_1 *and a vector of instances* $\vec{\mathsf{u}_2}$ *in* $\mathcal{R}_2$ *of size* ν *with structure* s_2*, and corresponding witness vectors* $\vec{\mathsf{w}_1}$ *and* $\vec{\mathsf{w}_2}$ *outputs a folded instance-witness pair* (u, w) *in* $\mathcal{R}_1$ *with structure* s_1*.*

- $\mathcal{V}(\mathsf{vk}, (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2})) \to \mathsf{u}$*: on input a vector of instances* $\vec{\mathsf{u}_1}$ *and a vector of instances* $\vec{\mathsf{u}_2}$ *outputs a new instance* u*.*

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* $((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}), (\vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}))$ *and runs the interaction on prover input* $(\mathsf{pk}, (\vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}), (\vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}))$ *and verifier input* $(\mathsf{vk}, (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}))$*. At the end of interaction* $\langle \mathcal{P}, \mathcal{V} \rangle$ *outputs* (u, w) *where* u *is the verifier's output folded instance, and* w *is the prover's output folded witness.*

Let $\mathcal{R}^{(n)}$ *be the relation such that* $(\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}, \vec{\mathsf{w}}) \in \mathcal{R}^{(n)}$ *if and only if* $(\mathsf{pp}, \mathsf{s}, \vec{\mathsf{u}}_i, \vec{\mathsf{w}}_i) \in \mathcal{R}$ *for all* $i \in [n]$*. A multi-folding scheme for* $(\mathcal{R}_1, \mathcal{R}_2, \mu, \nu)$ *satisfies the following requirements.*

1. *Perfect Completeness: For all PPT adversaries* $\mathcal{A}$*, we have that*

$$\Pr\left[(\mathsf{pp}, \mathsf{s}_1, \mathsf{u}, \mathsf{w}) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), (\vec{\mathsf{w}_1}, \vec{\mathsf{w}_2})) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}) \in \mathcal{R}_1^{(\mu)}, (\mathsf{pp}, \mathsf{s}_2, \vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}) \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1, \mathsf{s}_2), \\ (u, w) \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), (\vec{\mathsf{w}_1}, \vec{\mathsf{w}_2})) \end{array} \right] = 1.$$

2. *Knowledge Soundness: For any expected polynomial-time adversaries* $\mathcal{A}$ *and* $\mathcal{P}^*$ *there is an expected polynomial-time extractor* $\mathcal{E}$ *such that over all randomness* r

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{s}_1, \vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}) \in \mathcal{R}_1^{(\mu)}, \\ (\mathsf{pp}, \mathsf{s}_2, \vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}) \in \mathcal{R}_2^{(\nu)} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\vec{\mathsf{w}_1}, \vec{\mathsf{w}_2}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{r}) \end{array} \right] \approx$$

$$\Pr\left[(\mathsf{pp}, \mathsf{s}_1, u, w) \in \mathcal{R}_1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ (u, w) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{st}) \end{array} \right]$$

3. *Efficiency: The communication costs and* $\mathcal{V}$*'s computation are lower in the case where* $\mathcal{V}$ *participates in the multi-folding scheme and then checks a witness sent by* $\mathcal{P}$ *for the folded instance than the case where* $\mathcal{V}$ *checks witnesses sent by* $\mathcal{P}$ *for each of the original instances.*

A multi-folding scheme is secure in the random oracle model if the above requirements hold when all parties are provided access to a random oracle.

Definition 9 (Zero-knowledge). *Let* $\mathsf{trace}(\langle \mathcal{P}, \mathcal{V} \rangle, \mathsf{input})$ *denote the non-deterministic function which takes as input an interaction function* $\langle \mathcal{P}, \mathcal{V} \rangle$ *and a prescribed input* input*, and produces an interaction transcript between* $\mathcal{P}$ *and* $\mathcal{V}$ *on input* input*.*

A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mu, \nu)$ satisfies zero-knowledge if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ and $\mathcal{V}^*$, and input randomness r

$$\left\{ (\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{tr}, \mathsf{r}) \left| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}), (\vec{\mathsf{u}_2}, \vec{\mathsf{w}_2})) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}) \in \mathcal{R}_1^{(\mu)}, (\mathsf{pp}, \mathsf{s}_2, \vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}) \in \mathcal{R}_2^{(\nu)}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ \mathsf{tr} \leftarrow \mathsf{trace}(\langle \mathcal{P}, \mathcal{V}^*(\mathsf{r}) \rangle, ((\mathsf{pk}, \mathsf{vk}), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), (\vec{\mathsf{w}_1}, \vec{\mathsf{w}_2}))) \end{array} \right. \right\} \cong$$

$$\left\{ (\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{tr}, \mathsf{r}) \left| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ ((\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}), (\vec{\mathsf{u}_2}, \vec{\mathsf{w}_2})) \leftarrow \mathcal{A}(\mathsf{pp}), \\ \mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}, \\ (\mathsf{pp}, \mathsf{s}_1, \vec{\mathsf{u}_1}, \vec{\mathsf{w}_1}) \in \mathcal{R}_1, (\mathsf{pp}, \mathsf{s}_2, \vec{\mathsf{u}_2}, \vec{\mathsf{w}_2}) \in \mathcal{R}_2, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)), \\ \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2), (\vec{\mathsf{u}_1}, \vec{\mathsf{u}_2}), \mathsf{r}) \end{array} \right. \right\}$$

Definition 10 (Non-interactive). *A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. This single message is denoted as $\mathcal{P}$'s output and as $\mathcal{V}$'s input.*

Definition 11 (Public-coin). *A multi-folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public-coin if all the messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled from a uniform distribution.*

3.1 Achieving non-interactivity for multi-folding schemes

Construction 1 (Fiat-Shamir transformation for multi-folding schemes). Consider a public-coin multi-folding scheme $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ with ℓ rounds. Let ρ denote a random oracle. We construct a non-interactive multi-folding scheme $\Pi' = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}')$ for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ in the random oracle model as follows.

- $\mathcal{G}'(1^\lambda) \rightarrow \mathsf{pp}$: Compute and output $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$.
- $\mathcal{K}'(\mathsf{pp}, \mathsf{s}) \rightarrow \mathsf{pp}$:
  1. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$.
  2. Compute $\mathsf{hs} \leftarrow \rho(\mathsf{pp}, \mathsf{s})$.
  3. Output $(\mathsf{pk}', \mathsf{vk}') \leftarrow ((\mathsf{pk}, \mathsf{hs}), (\mathsf{vk}, \mathsf{hs}))$.
- $\mathcal{P}'(\mathsf{pk}', (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2))$:
  1. Parse pk' as $(\mathsf{pk}, \mathsf{hs})$
  2. Run $\mathcal{P}(\mathsf{pk}, (\vec{u}_1, \vec{u}_2), (\vec{w}_1, \vec{w}_2))$. On the ith message m_i, respond with verifier randomness $r_{i+1} \leftarrow \rho(m_i, r_i)$ where $r_1 = \mathsf{hs}$. Let (u, w) be the output of $\mathcal{P}$ and let $\pi = (m_1, \ldots, m_\ell)$ consist of messages from $\mathcal{P}$.

3. Send π to the verifier.
4. Output (u, w).

- $\mathcal{V}'(\mathsf{vk}', (\vec{u}_1, \vec{u}_2))$:
  1. Parse vk' as $(\mathsf{vk}, \mathsf{hs})$
  2. Recieve $\pi = (m_1, \ldots, m_\ell)$ from the prover. Compute $r_{i+1} \leftarrow \rho(m_i, r_i)$ for $r_1 = \mathsf{hs}$.
  3. Run $\mathcal{V}(\mathsf{vk}, (\vec{u}_1, \vec{u}_2))$ with randomness $(r_1, \ldots, r_{\ell+1})$. In round i, send the prover message m_i. Let u be the output of $\mathcal{V}$.
  4. Output u.

Lemma 3 (Fiat-Shamir transformation for multi-folding schemes). *Construction 1 transforms a public-coin multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ into a non-interactive multi-folding scheme for $(\mathcal{R}_1, \mathcal{R}_2, \mathsf{compat}, \mu, \nu)$ in the random oracle model.*

4 Customizable constraint systems

Setty et al. [STW23] recently introduced customizable constraint systems (CCS), a constraint system that simultaneously generalizes R1CS, Plonkish, and AIR without overheads. We first recall its definition and then describe variants that sections ahead will show are amenable to constructing multi-folding schemes. The definitions below are characterized by a finite field $\mathbb{F}$, but we leave this implicit.

Definition 12 (CCS [STW23]). *We define the customizable constraint system (CCS) relation $\mathcal{R}_{\mathsf{CCS}}$ as follows. Let the public parameter consists of size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$.*

An $\mathcal{R}_{\mathsf{CCS}}$ structure s consists of:

- *a sequence of matrices $M_1, \ldots, M_t \in \mathbb{F}^{m \times n}$ with at most $N = \Omega(\max(m, n))$ non-zero entries in total;*
- *a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.*
- *a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.*

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

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

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

4.1 Committed CCS

Consider a CCS structure

$$\mathsf{s}_{\mathsf{CCS}} = (m, n, N, \ell, t, q, d, [M_1, \ldots, M_t], [S_1, \ldots, S_t], [c_1, \ldots, c_t]).$$

Let $s = \log m$ and $s' = \log n$. We interpret each M_i (for $i \in [t]$) as functions of type $\{0,1\}^s \times \{0,1\}^{s'} \rightarrow \mathbb{F}$. For $i \in [t]$, let $\widetilde{M_i}$ denote the multilinear extension of M_i; that is, $\widetilde{M_i}$ is the unique multilinear polynomial in $\log m + \log n$ variables such that for all $x \in \{0,1\}^s$ and $y \in \{0,1\}^{s'}$ we have that $\widetilde{M_i}(x,y) = M_i(x,y)$. Similarly, for a purported witness $w \in \mathbb{F}^{n-\ell-1}$ let $\widetilde{w}$ denote the unique multilinear extension of w viewed as a function. Without loss of generality, below, we let $|w| = \ell + 1$.

Definition 13 (Committed CCS). *Let $\mathit{PC} = (\mathit{Gen}, \mathit{Commit}, \mathit{Open}, \mathit{Eval})$ denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$.*

We define the committed customizable constraint system (CCCS) relation $\mathcal{R}_{\mathsf{CCCS}}$ as follows. Let the public parameter consists of size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n = 2 \cdot (\ell + 1)$ and $\mathsf{pp} \leftarrow \mathit{Gen}(1^\lambda, s' - 1)$. Let $s = \log m$ and $s' = \log n$.

An $\mathcal{R}_{\mathsf{CCCS}}$ structure s consists of:

- *a sequence of sparse multilinear polynomials in $s + s'$ variables $\widetilde{M_1}, \ldots, \widetilde{M_t}$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^s \times \{0,1\}^{s'}$;*
- *a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.*
- *a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.*

An $\mathcal{R}_{\mathsf{CCCS}}$ instance is (C, x), where C is a commitment to a multilinear polynomial in $s' - 1$ variables and $\mathsf{x} \in \mathbb{F}^\ell$. An $\mathcal{R}_{\mathsf{CCCS}}$ witness consists of a multilinear polynomial $\widetilde{w}$ in $s' - 1$ variables. An $\mathcal{R}_{\mathsf{CCCS}}$ structure-instance tuple is satisfied by an $\mathcal{R}_{\mathsf{CCCS}}$ witness if $\mathit{Commit}(\mathsf{pp}, \widetilde{w}) = C$ and if for all $x \in \{0,1\}^s$,

$$\sum_{i=1}^{q} c_i \cdot \left(\Pi_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{M_j}(x,y) \cdot \widetilde{z}(y) \right) \right) = 0,$$

where $\widetilde{z}$ is an s'-variate multilinear polynomial such that $\widetilde{z}(x) = \widetilde{(w, 1, \mathsf{x})}(x)$ for all $x \in \{0,1\}^{s'}$.

4.2 Linearized committed CCS

Definition 14 (Linearized committed CCS). *Let $\mathit{PC} = (\mathit{Gen}, \mathit{Commit}, \mathit{Open}, \mathit{Eval})$ denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$.*

We define the linearized committed customizable constraint system (LCCS) relation $\mathcal{R}_{\mathsf{LCCCS}}$ as follows. Let the public parameter consists of size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n = 2 \cdot (\ell + 1)$ and $\mathsf{pp} \leftarrow \mathit{Gen}(1^\lambda, s' - 1)$. Let $s = \log m$ and $s' = \log n$.

An $\mathcal{R}_{\mathsf{LCCCS}}$ structure s consists of:

- *a sequence of sparse multilinear polynomials in $s + s'$ variables $\widetilde{M}_1, \ldots, \widetilde{M}_t$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^s \times \{0,1\}^{s'}$;*
- *a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.*
- *a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.*

An $\mathcal{R}_{\mathsf{LCCCS}}$ instance is $(C, u, \mathsf{x}, r, v_1, \ldots, v_t)$, where $u \in \mathbb{F}$, $\mathsf{x} \in \mathbb{F}^\ell$, $r \in \mathbb{F}^s$, $v_i \in \mathbb{F}$ for all $i \in [t]$, and C is a commitment to a multilinear polynomial in $s' - 1$ variables. An $\mathcal{R}_{\mathsf{LCCCS}}$ witness is a multilinear polynomial $\widetilde{w}$ in $s' - 1$ variables.

An $\mathcal{R}_{\mathsf{LCCCS}}$ structure-instance tuple is satisfied by an $\mathcal{R}_{\mathsf{LCCCS}}$ witness if $\mathit{Commit}(\mathsf{pp}, \widetilde{w}) = C$ and if for all $i \in [t]$, $v_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r, y) \cdot \widetilde{z}(y)$, where $\widetilde{z}$ is an s'-variate multilinear polynomial such that $z(x) = (\widetilde{w, u, \mathsf{x}})(x)$ for all $x \in \{0,1\}^{s'}$.

5 A multi-folding scheme for CCS

This section describes a multi-folding scheme for CCS. In particular, we provide a multi-folding scheme for $\mathcal{R}_1 = \mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_2 = \mathcal{R}_{\mathsf{CCCS}}$, with $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2)$ requiring $\mathsf{s}_1 = \mathsf{s}_2$. To keep the description simple, we describe and prove the case of $\mu = \nu = 1$. However, we stress that both the construction and proofs naturally generalize to the case of arbitrary values of μ and ν. In particular, the generalized version simply uses more powers of a random challenge when combining claims.

Construction 2 (A multi-folding scheme for CCS). We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$, where compat is defined as follows.

$\underline{\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \rightarrow \{\mathsf{true}, \mathsf{false}\}}$

1. If $\mathsf{s}_1 = \mathsf{s}_2$, then return true, otherwise return false

Let $\mathsf{s}_1 = \mathsf{s}_2 = ([\widetilde{M}_1, \ldots, \widetilde{M}_t], [S_1, \ldots, S_q], [c_1, \ldots, c_q])$. Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials.

We define the generator and the encoder as follows.

$\underline{\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}}$:

1. Sample size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ with $n = 2 \cdot (\ell + 1)$
2. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Gen}(1^\lambda, \log n - 1)$

3. Output $(m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}})$

$\underline{\mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2)) \rightarrow (\mathsf{pk}, \mathsf{vk})}$:

1. Let $\mathsf{pk} \leftarrow (\mathsf{pp}, \mathsf{s}_1)$ and $\mathsf{vk} \leftarrow \mathsf{pp}$
2. Output $(\mathsf{pk}, \mathsf{vk})$

The verifier $\mathcal{V}$ takes a linearized committed CCS instance $(C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t)$. and a committed CCS instance (C_2, x_2) and the prover $\mathcal{P}$, in addition to the two instances, takes witnesses to both instances, $\widetilde{w_1}$ and $\widetilde{w_2}$.

Let $s = \log m$ and $s' = \log n$. Let $\widetilde{z_1} = \widetilde{(w_1, u, \mathsf{x}_1)}$ and $\widetilde{z_2} = \widetilde{(w_2, 1, \mathsf{x}_2)}$.

The prover and the verifier proceed as follows.

1. $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}, \beta \xleftarrow{\$} \mathbb{F}^s$, and sends them to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $r'_x \xleftarrow{\$} \mathbb{F}^s$.
3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle (g, s, d+1, \sum_{j \in [t]} \gamma^j \cdot v_j)$, where:

$$g(x) := \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x)$$

$$L_j(x) := \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_1(y) \right)$$

$$Q(x) := \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(x, y) \cdot \widetilde{z}_2(y) \right) \right)$$

4. $\mathcal{P} \rightarrow \mathcal{V}$: $((\sigma_1, \ldots, \sigma_t), (\theta_1, \ldots, \theta_t))$, where for all $i \in [t]$:

$$\sigma_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_1(y)$$

$$\theta_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_i}(r'_x, y) \cdot \widetilde{z}_2(y)$$

5. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(r_x, r'_x)$ and $e_2 \leftarrow \widetilde{eq}(\beta, r'_x)$, and abort if:

$$c \neq \left(\sum_{j \in [t]} \gamma^j \cdot e_1 \cdot \sigma_j + \gamma^{t+1} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_j \right) \right)$$

6. $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.

7. $\mathcal{V}, \mathcal{P}$: Output the folded linearized committed CCS instance $(C', u', \mathsf{x}', r_x', v_1', \ldots, v_t')$, where for all $i \in [t]$:

$$\begin{aligned} C' &\leftarrow C_1 + \rho \cdot C_2 \\ u' &\leftarrow u \quad + \rho \cdot 1 \\ \mathsf{x}' &\leftarrow \mathsf{x}_1 \ + \rho \cdot \mathsf{x}_2 \\ v_i' &\leftarrow \sigma_i \ + \rho \cdot \theta_i \end{aligned}$$

8. $\mathcal{P}$: Output the folded witness $\widetilde{w}' \leftarrow \widetilde{w}_1 + \rho \cdot \widetilde{w}_2$.

Theorem 1 (A multi-folding scheme for CCS). *Construction 2 is a public-coin multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, \mu = 1, \nu = 1)$ with perfect completeness and knowledge soundness.*

Lemma 4 (Perfect Completeness). *Construction 2 satisfies perfect completeness.*

Proof. Consider the public parameters $\mathsf{pp} = (m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}}) \leftarrow \mathcal{G}(1^\lambda)$ and let $s = \log m$ and $s' = \log n$.

Consider arbitrary structures $(\mathsf{s}_1, \mathsf{s}_2)$, where $\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) = \mathsf{true}$

$$\mathsf{s}_1 = \mathsf{s}_2 = (\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q) \leftarrow \mathcal{A}(\mathsf{pp}).$$

Consider the prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$. Suppose that the prover and the verifier are provided a linearized committed CCS instance

$$(C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t),$$

and a committed CCS instance

$$(C_2, \mathsf{x}_2).$$

Suppose that the prover additionally is provided with the corresponding satisfying witnesses $\widetilde{w}_1$ and $\widetilde{w}_2$.

Because the input linearized committed CCS instance-witness pair is satisfying, we have, for $\widetilde{z}_1 = \widetilde{(w_1, u, \mathsf{x}_1)}$, that

$$\begin{aligned} v_j &= \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z}_1(y) && \text{By precondition.} \\ &= \sum_{x \in \{0,1\}^{s}} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_1(y) \right) && \text{By Lemma 1.} \\ &= \sum_{x \in \{0,1\}^{s}} L_j(x) && \text{By construction.} \end{aligned}$$

Moreover, because the input committed CCS instance-witness pair is satisfying, we have, for all $x \in \{0,1\}^s$ and for $\widetilde{z}_2(x) = \widetilde{(w, 1, \mathsf{x})}(x)$, that

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right)$$

Treating the right-hand side of the above equation as a polynomial in x, because it vanishes on all $x \in \{0,1\}^s$, we have that it must be the zero polynomial. Therefore, we have, for β sampled by the verifier, that

$$\begin{aligned}
0 &= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(\beta, y) \cdot \widetilde{z}_2(y) \right) \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right) && \text{By Lemma 1.} \\
&= \sum_{x \in \{0,1\}^s} Q(x) && \text{By construction.}
\end{aligned}$$

Therefore, for γ sampled by the verifier, by linearity, we have that

$$\begin{aligned}
\sum_{j \in [t]} \gamma^j \cdot v_j &= \sum_{x \in \{0,1\}^s} \left(\left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x) \right) \\
&= \sum_{x \in \{0,1\}^s} g(x) && \text{By construction.}
\end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have for $e_1 = \widetilde{eq}(r_x, r'_x)$, $e_2 = \widetilde{eq}(\beta, r'_x)$ and

$$\sigma_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_1(y) \quad \text{and} \quad \theta_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_2(y)$$

that

$$\begin{aligned}
c &= g(r'_x) \\
&= \left(\left(\sum_{j \in [t]} \gamma^j \cdot L_j(r'_x) \right) + \gamma^{t+1} \cdot Q(r'_x) \right) \\
&= \left(\left(\sum_{j \in [t]} \gamma^j \cdot e_1 \cdot \sigma_j \right) + \gamma^{t+1} \cdot e_2 \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_j \right).
\end{aligned}$$

This implies that the verifier will not abort.

Now, consider the linearized CCS instance

$$(C_2, 1, \mathsf{x}_2, r'_x, \theta_1, \ldots, \theta_t).$$

By the precondition that the committed CCS instance (C_2, x_2) is satisfied by $\widetilde{w}_2$ and by the definition of $(\theta_1, \ldots, \theta_t)$, we have that this linearized CCS instance is satisfied by the witness $\widetilde{w}_2$.

Therefore, for a random ρ sampled by the verifier, and for $C' = C_1 + \rho \cdot C_2$, $u' = u + \rho \cdot 1$, $\mathsf{x}' = \mathsf{x}_1 + \rho \cdot \mathsf{x}_2$, $v'_i = \sigma_i + \rho \cdot \theta_i$, we have that the output linearized CCS instance

$$(C', u', \mathsf{x}', r'_x, v'_1, \ldots, v'_t)$$

is satisfied by the witness $\widetilde{w}' = \widetilde{w}_1 + \rho \cdot \widetilde{w}_2$ by the linearity and the additive homomorphism property of the polynomial commitment scheme. $\square$

Some of our probabilistic analysis below is adapted from the proof of forking lemma for folding schemes [KST22], which itself builds on the proof of the forking lemma for interactive arguments [BCC+16].

Lemma 5 (Knowledge Soundness). *Construction 2 satisfies knowledge soundness.*

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structure and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$. Suppose on input pp and random tape r, the adversary $\mathcal{A}$ picks structures satisfying compat

$$\mathsf{s} = \mathsf{s}_1 = \mathsf{s}_2 = ([M_1, \ldots, M_t], [S_1, \ldots, S_q], [c_1, \cdots, c_q]),$$

a linearized committed CCS instance

$$\varphi_1 = (C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t),$$

a committed CCS instance

$$\varphi_2 = (C_2, \mathsf{x}_2),$$

and some auxiliary state st.

We construct an expected-polynomial time extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances as follows.

$\underline{\mathcal{E}(\mathsf{pp}, \mathsf{r})}$:

1. Obtain the output tuple from $\mathcal{A}$:

$$((\mathsf{s}_1, \mathsf{s}_2), (\varphi_1, \varphi_2), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}).$$

2. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$.

3. Run the interaction

$$(\varphi^{(1)}, \widetilde{w}^{(1)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

once with the final verifier challenge $\rho^{(1)} \xleftarrow{\$} \mathbb{F}$.

4. Abort if $(\mathsf{pp}, \mathsf{s}, \varphi^{(1)}, \widetilde{w}^{(1)}) \notin \mathcal{R}_{\mathsf{LCCCS}}$.

5. Rewind the interaction

$$(\varphi^{(2)}, \widetilde{w}^{(2)}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \varphi_1, \varphi_2, \mathsf{st})$$

with a different verifier's final challenge $\rho^{(2)} \xleftarrow{\$} \mathbb{F}$ while maintaining the same prior randomness. Keep doing so until $(\mathsf{pp}, \mathsf{s}, \varphi^{(2)}, \widetilde{w}^{(2)}) \in \mathcal{R}_{\mathsf{LCCCS}}$.

6. Abort if $\rho^{(1)} = \rho^{(2)}$.

7. Interpolating points $(\rho^{(1)}, \widetilde{w}^{(1)})$ and $(\rho^{(2)}, \widetilde{w}^{(2)})$, retrieve the witness polynomials $\widetilde{w}_1$ and $\widetilde{w}_2$ such that for $i \in \{1, 2\}$

$$\widetilde{w}_1 + \rho^{(i)} \cdot \widetilde{w}_2 = \widetilde{w}^{(i)}. \tag{1}$$

8. Output $(\widetilde{w}_1, \widetilde{w}_2)$.

We first demonstrate that the extractor $\mathcal{E}$ runs in expected polynomial time. Observe that $\mathcal{E}$ runs the interaction once, and if it does not abort, keeps rerunning the interaction until $\mathcal{P}^*$ succeeds. Thus, the expected number of times $\mathcal{E}$ runs the interaction is

$$1 + \Pr[\text{First call to } \langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}] \cdot \frac{1}{\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle \text{ succeeds}]} = 1 + \epsilon \cdot \frac{1}{\epsilon} = 2.$$

Therefore, we have that the extractor runs in expected polynomial-time.

We now analyze $\mathcal{E}$'s success probability. We must demonstrate that $\mathcal{E}$ succeeds in producing $\widetilde{w}_1$ and $\widetilde{w}_2$ such that

$$(\mathsf{pp}, \mathsf{s}, \varphi_1, \widetilde{w}_1) \in \mathcal{R}_{\mathsf{LCCCS}} \quad \text{and} \quad (\mathsf{pp}, \mathsf{s}, \varphi_2, \widetilde{w}_2) \in \mathcal{R}_{\mathsf{CCCS}}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$.

To do so, we first show that the extractor successfully produces *some* output (i.e., does not abort) in under $|\mathbb{F}|$ rewinding steps with probability $\epsilon - \mathsf{negl}(\lambda)$. Note that $|\mathbb{F}|$ is a worst case bound and we have already established that the extractor runs in expected polynomial time. By the malicious prover's success probability, we have that the extractor does not abort in step (4) with probability ϵ. Given that the extractor does not abort in step (4), by Markov's inequality, we have that the extractor rewinds more than $|\mathbb{F}|$ times with probability $2/|\mathbb{F}|$. Thus, the probability that the extractor does not abort in step (4) and requires less than $|\mathbb{F}|$ rewinds is $(1 - 2/|\mathbb{F}|) \cdot \epsilon$.

Now, suppose that the the extractor does not abort in step (4) and requires less than $|\mathbb{F}|$ rewinds. Then, because the extractor randomly samples $\rho^{(2)}$, we have that $\rho^{(1)} \neq \rho^{(2)}$ with probability $1/|\mathbb{F}|$. Thus, we have that the probability the extractor successfully produces some output in under $|\mathbb{F}|$ rewinding steps is

$$\left(1 - \frac{2}{|\mathbb{F}|}\right) \cdot \epsilon \cdot \left(1 - \frac{1}{|\mathbb{F}|}\right) = \epsilon - \mathsf{negl}(\lambda).$$

Next, if the extractor does not abort, we show that the extractor succeeds in producing satisfying witnesses with probability $1 - \mathsf{negl}(\lambda)$. This brings the overall extractor success probability to $\epsilon - \mathsf{negl}(\lambda)$.

Indeed, for $i \in \{1, 2\}$, let $\varphi^{(i)} = (C^{(i)}, u^{(i)}, \mathsf{x}^{(i)}, r_x^{(i)}, v_1^{(i)}, \ldots, v_t^{(i)})$. We first show that the retrieved polynomials are valid openings to the corresponding commitments in the instance. For $i \in \{1, 2\}$, because $\widetilde{w}^{(i)}$ is a satisfying witness, by construction,

$$\begin{aligned}
&\mathsf{Commit}(\mathsf{pp}, \widetilde{w}_1) + \rho^{(i)} \cdot \mathsf{Commit}(\mathsf{pp}, \widetilde{w}_2) && \\
&= \mathsf{Commit}(\mathsf{pp}, \widetilde{w}_1 + \rho^{(i)} \cdot \widetilde{w}_2) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}, \widetilde{w}^{(i)}) && \text{By Equation (1).} \\
&= C^{(i)} && \text{Witness } \widetilde{w}^{(i)} \text{ is a satisfying opening.} \\
&= C_1 + \rho^{(i)} \cdot C_2 && \text{By the verifier's computation.}
\end{aligned}$$

Interpolating, we have that

$$\mathsf{Commit}(\mathsf{pp}, \widetilde{w}_1) = C_1 \tag{2}$$

$$\mathsf{Commit}(\mathsf{pp}, \widetilde{w}_2) = C_2. \tag{3}$$

Next, we must argue that $\widetilde{w}_1$ and $\widetilde{w}_2$ satisfy the remainder of the instances φ_1 and φ_2 respectively under the structure s.

Indeed, consider $(\sigma_1, \ldots, \sigma_t)$ and $(\theta_1, \ldots, \theta_t)$ sent by the prover which by the extractor's construction are identical across all executions of the interaction. By the verifier's computation we have that for $i \in \{1, 2\}$ and all $j \in [t]$

$$\sigma_j + \rho^{(i)} \cdot \theta_j = v_j^{(i)} \tag{4}$$

Now, because $\widetilde{w}^{(i)}$ is a satisfying witness, for $i \in \{1, 2\}$ we have for all $j \in [t]$ that

$$v_j^{(i)} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x', y) \cdot \widetilde{z}^{(i)}(y),$$

where $\widetilde{z}^{(i)} = (w^{(i)}, \widetilde{u^{(i)}, \mathsf{x}^{(i)}})$.

However, by Equations (1) and (4), for $i \in \{1,2\}$ and $j \in [t]$, this implies that

$$\sigma_j + \rho^{(i)} \cdot \theta_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_1}(y) + \rho^{(i)} \cdot \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_2}(y),$$

where $\widetilde{z_1} = (\widetilde{w_1, u, \mathsf{x}_1})$ and $\widetilde{z_2} = (\widetilde{w_2, 1, \mathsf{x}_2})$. Interpolating, we have that, for all $j \in [t]$

$$\sigma_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_1}(y)$$
$$\theta_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_2}(y)$$

Thus, because that the verifier does not abort, we have that

$$\begin{aligned}
c &= \left(\sum_{j \in t} \gamma^j \cdot e_1 \cdot \sigma_j\right) + \left(\gamma^{t+1} \cdot e_2 \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_j\right) \\
&= \left(\sum_{j \in t} \gamma^j \cdot \widetilde{eq}(r_x, r'_x) \cdot \sigma_j\right) + \left(\gamma^{t+1} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_j\right) \\
&= \left(\sum_{j \in t} \gamma^j \cdot \widetilde{eq}(r_x, r'_x) \cdot \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_1}(y)\right) + \\
&\quad \left(\gamma^{t+1} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z_2}(y)\right) \\
&= \sum_{j \in [t]} \gamma_j \cdot L_j(r'_x) + \gamma^{t+1} \cdot Q(r'_x) \\
&= g(r'_x)
\end{aligned}$$

by the soundness of the sum-check protocol, this implies that with probability $1 - O(d \cdot s)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of r'_x,

$$\begin{aligned}
\sum_{j \in [t]} \gamma^j \cdot v_j + \gamma^{t+1} \cdot 0 &= \sum_{x \in \{0,1\}^s} g(x) \\
&= \sum_{x \in \{0,1\}^s} \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) + \gamma^{t+1} \cdot Q(x)\right) \\
&= \sum_{j \in [t]} \gamma^j \cdot \left(\sum_{x \in \{0,1\}^s} L_j(x)\right) + \gamma^{t+1} \cdot \sum_{x \in \{0,1\}^s} Q(x)
\end{aligned}$$

By the Schwartz-Zippel lemma [Sch80], this implies that with probability $1 - O(t)/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of γ, we have

$$v_j = \sum_{x \in \{0,1\}^s} L_j(x)$$

for all $j \in [t]$ and

$$0 = \sum_{x \in \{0,1\}^s} Q(x).$$

Now, for all $j \in [t]$, we have

$$\begin{aligned} v_j &= \sum_{x \in \{0,1\}^s} L_j(x) \\ &= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} M_j(x,y) \cdot \widetilde{z}_1(y) \right) \\ &= \sum_{y \in \{0,1\}^{s'}} M_j(r_x, y) \cdot \widetilde{z}_1(y) \end{aligned}$$

This implies that $\widetilde{w_1}$ is a satisfying witness to φ_1.

Finally, we have that

$$\begin{aligned} 0 &= \sum_{x \in \{0,1\}^s} Q(x) \\ &= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x,y) \cdot \widetilde{z}_2(y) \right) \right) \\ &= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(\beta,y) \cdot \widetilde{z}_2(y) \right) \end{aligned}$$

By the Schwartz-Zippel lemma, this implies that with probability $1 - s/|\mathbb{F}| = 1 - \mathsf{negl}(\lambda)$ over the choice of β, we have that for all $x \in \{0,1\}^s$

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x,y) \cdot \widetilde{z}_2(y) \right)$$

This implies that $\widetilde{w_2}$ is a satisfying witness to φ_2.

Thus, if the extractor does not abort, it succeeds in producing satisfying witness $(\widetilde{w_1}, \widetilde{w_2})$ with probability $1 - \mathsf{negl}(\lambda)$. $\square$

Assumption 1 (Non-interactive multi-folding scheme). There exists a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, 1, 1)$ in the plain model.

Justification. By applying the Fiat-Shamir transformation in the Construction 1 to the multi-folding scheme in Construction 2, we obtain a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, 1, 1)$ in the random oracle model. By instantiating the random oracle with an appropriate cryptographic hash function, we heuristically obtain a non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, 1, 1)$ in the plain model. □

6 HyperNova: IVC from multi-folding schemes

We now describe HyperNova, a prover-efficient and recursive zkSNARK for proving incremental computations where each step is expressed with committed CCS. For simplicity, we focus on constructing an IVC scheme, but our construction extends easily to provide a natural extension of IVC to distributed computations called proof-carrying data (PCD) [BCCT13]. We leave a full specification and an analysis of HyperNova's PCD scheme to the near-term future work.

As noted earlier, we provide two constructions, one that directly builds on our non-interactive multi-folding scheme for CCS, and another that uses the multi-folding scheme in conjunction with Nova as a black box.

6.1 A direct approach

This section provides a direct construction of IVC using the non-interactive multi-folding scheme. Our construction below is an adaptation of Nova's IVC scheme to the case of multi-folding schemes.

Construction 3 (A direct construction of HyperNova's IVC). Let NIFS be the non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, \mathsf{compat}, 1, 1)$ as described in Construction 2. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be a default trivially satisfying $\mathcal{R}_{\mathsf{LCCCS}}$ instance-witness pair for any structure and public parameters. We construct an IVC scheme as follows.

Consider a polynomial-time function F that takes non-deterministic input and a cryptographic hash function hash. We define our augmented function F', where all input arguments are taken as non-deterministic advice, as follows.

$\underline{F'(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi) \to \mathsf{x}:}$

1. If $i = 0$, output $\mathsf{hash}(\mathsf{vk}, 1, z_0, F(z_0, \omega_0), \mathsf{u}_\perp)$
2. Otherwise:
   (a) Check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$, where $\mathsf{u}_i.\mathsf{x}$ is the public IO of u_i
   (b) Compute $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \pi)$
   (c) Output $\mathsf{hash}(\mathsf{vk}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1})$

Because F' can be computed in polynomial time, it can be represented as a $\mathcal{R}_{\mathsf{CCCS}}$ structure. Let

$$(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi))$$

denote the satisfying $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ for the execution of F' on non-deterministic advice $(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi)$.

We define the IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}}$: Output $\mathsf{NIFS.G}(1^\lambda)$.

$\underline{\mathcal{K}(\mathsf{pp}, F) \rightarrow (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{pk_{fs}}, \mathsf{vk_{fs}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, F')$
2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, \mathsf{pk_{fs}}), (F, \mathsf{vk_{fs}}))$.

$\underline{\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \rightarrow \Pi_{i+1}}$:

1. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
2. $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow$ if $i = 0$ $\{(\mathsf{u}_\perp, \mathsf{w}_\perp, \perp)\}$ else $\{\mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))\}$
3. Compute $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F', (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi))$
4. Output $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$

$\underline{\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \rightarrow \{0, 1\}}$:

1. If $i = 0$, check that $z_i = z_0$
2. Otherwise:
   (a) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
   (b) Check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$
   (c) Check that W_i and w_i are satisfying witnesses to U_i and u_i with respect to the structure corresponding to F'.

Theorem 2 (A direct construction of HyperNova's IVC). *Construction 3 is an IVC scheme.*

Lemma 6 (Completeness). *Construction 3 is an IVC scheme that satisfies completeness.*

Proof. Consider $(F, i+1, z_0, z_{i+1})$ and the corresponding inputs (z_i, ω_i) such that

$$z_{i+1} = F(z_i, \omega_i)$$

Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, and let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$. Now consider a proof Π_i such that

$$\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$$

We must show that given

$$\Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i)$$

that

$$\mathcal{V}(\mathsf{vk}, i+1, z_0, z_{i+1}, \Pi_{i+1}) = 1$$

with probability 1. We show this by induction on i.

<u>Base Case ($i = 0$)</u>: Suppose the prover is provided Π_0 such that

$$\mathcal{V}(\mathsf{vk}, 0, z_0, z_0, \Pi_0) = 1.$$

By the base case of $\mathcal{P}$ and F', we have

$$\Pi_1 = ((\mathsf{u}_\perp, \mathsf{w}_\perp), (\mathsf{u}_1, \mathsf{w}_1))$$

for some $(\mathsf{u}_1, \mathsf{w}_1)$. By definition, the instance-witness pair $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ is satisfying. Moreover, by construction, $(\mathsf{u}_1, \mathsf{w}_1)$ must also be satisfying. Additionally, by the construction of F', we have

$$\mathsf{u}_1.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, 1, z_0, F(z_0, w_0), \mathsf{u}_\perp).$$

Therefore, we have

$$\mathcal{V}(\mathsf{pp}, 1, z_0, z_1, \Pi_1) = 1.$$

<u>Inductive Step ($i \geq 1$)</u>: Assume that for

$$\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$$

we have that

$$\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$$

and suppose that

$$\Pi_{i+1} = ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1})) \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i).$$

By the construction of $\mathcal{P}$, we have that

$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i)).$$

Thus, by the premise that $(\mathsf{u}_i, \mathsf{w}_i)$ and $(\mathsf{U}_i, \mathsf{W}_i)$ are satisfying instance-witness pairs (with respect to the same structure), and by the completeness of the underlying folding scheme, we have that $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ is a satisfying instance-witness pair. Additionally, by the premise, we have that $\mathsf{u}_i.x = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i)$. Therefore,

$\mathcal{P}$ can construct a satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ that represents the correct execution of F' on input $(\mathsf{U}, \mathsf{u}, (i, z_0, z_i), \omega_i, \pi)$. By construction, this particular input implies that

$$\begin{aligned}\mathsf{u}_{i+1}.\mathsf{x} &= \mathsf{hash}(\mathsf{vk}, \mathsf{u}_i.i+1, \mathsf{u}_i.z_0, F(z_i, w_i), \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \pi)) \\ &= \mathsf{hash}(\mathsf{vk}, \mathsf{u}_i.i+1, \mathsf{u}_i.z_0, z_{i+1}, \mathsf{U}_{i+1})\end{aligned} \tag{5}$$

by the correctness of the underlying folding scheme. Thus, by Equation (5) we have

$$\mathcal{V}(\mathsf{vk}, i+1, z_0, z_{i+1}, \mathit{\Pi}_{i+1}) = 1.$$

□

Lemma 7 (Knowledge Soundness). *Construction 3 is an IVC scheme that satisfies knowledge soundness.*

Proof. Our approach is inspired by a recursive extraction technique described by Bünz et al [BCL$^+$21]. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$. Consider expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Suppose $\mathcal{A}$ outputs a function F on input pp, and let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$. Suppose that, for a constant n, $\mathcal{P}^*$ additionally outputs $(z_0, z, \mathit{\Pi})$ such that

$$\mathcal{V}(\mathsf{vk}, n, z_0, z, \mathit{\Pi}) = 1$$

with probability ϵ. We must construct an expected polynomial-time extractor $\mathcal{E}$ that with input (pp, z_0, z), outputs $(\omega_0, \ldots, \omega_{n-1})$ such that by computing

$$z_i \leftarrow F(z_{i-1}, \omega_{i-1})$$

we have that $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

We show inductively that $\mathcal{E}$ can construct an expected polynomial-time extractor $\mathcal{E}_i(\mathsf{pp})$ that outputs $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \mathit{\Pi}_i)$ such that for all $j \in \{i+1, \ldots, n\}$,

$$z_j = F(z_{j-1}, \omega_{j-1})$$

and

$$\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \mathit{\Pi}_i) = 1 \tag{6}$$

for $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Then, because in the base case when $i = 0$, $\mathcal{V}$ checks that $z_0 = z_i$, the values $(\omega_0, \ldots, \omega_{n-1})$ retrieved by $\mathcal{E}_0(\mathsf{pp})$ are such that computing $z_{i+1} = F(z_i, \omega_i)$ for all $i \geq 1$ gives $z_n = z$. At a high level, to construct an extractor $\mathcal{E}_{i-1}$, we first assume the existence of $\mathcal{E}_i$ that satisfies the inductive hypothesis. We then use $\mathcal{E}_i(\mathsf{pp})$ to construct an adversary for the non-interactive folding scheme (which we denote as $\widetilde{\mathcal{P}}_{i-1}$). This in turn guarantees an extractor for the non-interactive folding scheme, which we denote as $\widetilde{\mathcal{E}}_{i-1}$. We then use $\widetilde{\mathcal{E}}_{i-1}$ to construct $\mathcal{E}_{i-1}$ that satisfies the inductive hypothesis.

In the base case, for $i = n$, let $\mathcal{E}_n(\mathsf{pp};\mathsf{r})$ output $(\bot, \bot, \Pi_n)$ where Π_n is the output of $\mathcal{P}^*(\mathsf{pp};\mathsf{r})$. By the premise, $\mathcal{E}_n$ succeeds with probability ϵ in expected polynomial-time.

For $i \geq 1$, suppose $\mathcal{E}$ can construct an expected polynomial-time extractor $\mathcal{E}_i$ that outputs $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}))$, and Π_i that satisfies the inductive hypothesis. To construct an extractor $\mathcal{E}_{i-1}$, $\mathcal{E}$ first constructs an adversary $\widetilde{\mathcal{P}}_{i-1}$ for the non-interactive folding scheme as follows:

$\underline{\widetilde{\mathcal{P}}_{i-1}(\mathsf{pp};\mathsf{r})}$:

1. Let $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i) \leftarrow \mathcal{E}_i(\mathsf{pp};\mathsf{r})$.
2. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$.
3. Parse w_i to retrieve $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1}, \pi_{i-1})$.
4. Output the common structure corresponding to F' unfolded instances $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$, folded instance witness pairs $((\mathsf{U}_i, \mathsf{W}_i), \pi_{i-1})$, and folding proof π_{i-1}.

By the inductive hypothesis, we have that $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, \Pi_i) = 1$, where $\Pi_i \leftarrow \mathcal{E}_i(\mathsf{pp})$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Therefore, by the the verifier's checks we have that $(\mathsf{u}_i, \mathsf{w}_i)$ and $(\mathsf{U}_i, \mathsf{W}_i)$ are satisfying instance-witness pairs, and that

$$\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i).$$

Therefore, we have that w_i is indeed a satisfying assignment for F' (and not just a trivially satisfying witness). Then, by the construction of F' and the binding property of the hash function, we have that

$$\mathsf{U}_i = \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_{i-1}, \mathsf{u}_{i-1}, \pi_{i-1})$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Thus, $\widetilde{\mathcal{P}}_{i-1}$ succeeds in producing an accepting folded instance-witness pair $(\mathsf{U}_i, \mathsf{W}_i)$, for instances $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$, with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial-time.

Then, by the knowledge soundness of the underlying non-interactive multi-folding scheme (Assumption 1) there exists an extractor $\widetilde{\mathcal{E}}_{i-1}$ that outputs $(\mathsf{w}_{i-1}, \mathsf{W}_{i-1})$ such that $(\mathsf{u}_{i-1}, \mathsf{w}_{i-1})$ and $(\mathsf{U}_{i-1}, \mathsf{W}_{i-1})$ satisfy F' with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial-time.

Given an expected polynomial-time $\widetilde{\mathcal{P}}_{i-1}$ and an expected polynomial-time $\widetilde{\mathcal{E}}_{i-1}$, $\mathcal{E}$ constructs an expected polynomial time $\mathcal{E}_{i-1}$ as follows

$\underline{\mathcal{E}_{i-1}(\mathsf{pp};\mathsf{r})}$:

1. $((\mathsf{U}_{i-1}, \mathsf{u}_{i-1}), (\mathsf{U}_i, \mathsf{W}_i), \pi_{i-1}) \leftarrow \widetilde{\mathcal{P}}_{i-1}(\mathsf{pp};\mathsf{r})$
2. Retrieve $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i)$ from the internal state of $\widetilde{\mathcal{P}}_{i-1}$.

3. Parse $\mathit{\Pi}_i.\mathsf{w}_i$ to retrieve z_{i-1} and ω_{i-1}
4. Let $(\mathsf{w}_{i-1}, \mathsf{W}_{i-1}) \leftarrow \widetilde{\mathcal{E}}_{i-1}(\mathsf{pp})$.
5. Let $\mathit{\Pi}_{i-1} \leftarrow ((\mathsf{U}_{i-1}, \mathsf{W}_{i-1}), (\mathsf{u}_{i-1}, \mathsf{w}_{i-1}))$.
6. Output $((z_{i-1}, \ldots, z_{n-1}), (\omega_{i-1}, \ldots, \omega_{n-1}), \mathit{\Pi}_{i-1})$.

We first reason that the output $(z_{i-1}, \ldots, z_{n-1})$, and $(\omega_{i-1}, \ldots, \omega_{n-1})$ are valid. By the inductive hypothesis, we already have that for all $j \in \{i+1, \ldots, n\}$,

$$z_j = F(z_{j-1}, \omega_{j-1}),$$

and that $\mathcal{V}(\mathsf{vk}, i, z_0, z_i, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i)) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. Because $\mathcal{V}$ additionally checks that

$$\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, z_0, z_i, \mathsf{U}_i) \tag{7}$$

by the construction of F' and the binding property of the hash function, we have

$$F(z_{i-1}, \omega_{i-1}) = z_i$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Next, we argue that $\mathit{\Pi}_{i-1}$ is valid. Because $(\mathsf{u}_i, \mathsf{w}_i)$ satisfies F', and $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1})$ were retrieved from w_i, by the binding property of the hash function, and by Equation (7), we have that

$$\mathsf{u}_{i-1}.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i-1, z_0, z_{i-1}, \mathsf{U}_{i-1})$$

Additionally, in the case where $i = 1$, by the base case check of F', we have that $z_{i-1} = z_0$. Because $\widetilde{\mathcal{E}}_{i-1}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$, we have that

$$\mathcal{V}(\mathsf{vk}, i-1, z_0, z_{i-1}, \mathit{\Pi}_{i-1}) = 1$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. □

Lemma 8 (Efficiency). *When instantiated with the Pedersen commitment scheme, we have that $|F'| = |F| + o(\mathsf{G} + 2 \cdot \mathsf{H} + d \cdot \log m \cdot \mathsf{F} + \log m \cdot \mathsf{R})$, where $|F|$ denotes the number of CCS constraints to encode a function F, G is the number of constraints required to encode a group scalar multiplication, H is the number of constraints required to encode hash, F is the number of constraints to encode field operations, and R is the number of constraints to encode the RO ρ.*

Proof. On input instances U and u, $\mathsf{NIFS.V}$ computes $\mathsf{U}.C \leftarrow \mathsf{U}.C + \rho \cdot u.C$, which costs a single group scalar multiplication. Verifying the non-interactive sum-check proof in the non-interactive multi-folding scheme proof requires the verifier to perform $O(d \cdot \log m)$ finite field operations and $O(\log m)$ calls to the RO to obtain challenges in the sum-check protocol. Finally, F' makes two additional calls to hash (details are in the description of F'). □

6.2 A simple approach to build HyperNova: Use Nova as a black box

We design a step circuit for Nova that runs the verifier's logic in the non-interactive multi-folding scheme for $\mathcal{R}_{\mathsf{LCCCS}}$ and $\mathcal{R}_{\mathsf{CCCS}}$. The step circuit is encoded with R1CS (a popular NP-complete constraint system [GGPR13]) and proven incrementally with Nova, but the step circuit is only in charge of running the verifier of the non-interactive multi-folding scheme, in addition to simple bookkeeping. As a result, this provides an IVC scheme, where each step of the incremental computation is expressed with committed CCS. Furthermore, we achieve this with a black box use of an IVC scheme for R1CS.

In Nova, each step circuit takes as input the output of the previous step and produces the output for the current step. In HyperNova, besides the application's IO, we augment them with the latest running instance. At each recursive step, the step circuit gets as non-deterministic input a purported instance u in $\mathcal{R}_{\mathsf{CCCS}}$ and π, where π is the prover's output in the non-interactive multi-folding scheme. The step circuit checks that the public input of u matches the application's input provided to the step circuit. If so, it runs the verifier of the non-interactive folding scheme on $(\mathsf{vk}, \mathsf{U}, \mathsf{u}, \pi)$, where vk is the verifier's key and U is the latest running instance passed from the prior step. It then provides uses the public output of u and the output of the folding scheme verifier to construct the step's output.

Construction 4 (A step circuit for Nova). Let NIFS be the non-interactive multi-folding scheme for $(\mathcal{R}_{\mathsf{LCCCS}}, \mathcal{R}_{\mathsf{CCCS}}, 1, 1)$. Let IVC denote the Nova's IVC scheme for functions expressed as R1CS constraints.

We first define a non-deterministic polynomial-time function step, represented as an R1CS structure, that iteratively folds instances expressed in $\mathcal{R}_{\mathsf{CCCS}}$.

$\underline{\mathsf{step}(\mathsf{vk}, \mathsf{U}_i, z_i; (\mathsf{u}, \pi)) \rightarrow (\mathsf{vk}, \mathsf{U}_{i+1}, z_{i+1})}$

1. Parse $\mathsf{u.x}$ as $(\mathsf{in}, \mathsf{out})$
2. Check that $\mathsf{in} = z_i$
3. Compute $\mathsf{U}_{i+1} \leftarrow \mathsf{NIFS.V}(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}, \pi)$
4. Output $(\mathsf{vk}, \mathsf{U}_{i+1}, \mathsf{out})$

Given F, expressed as an $\mathcal{R}_{\mathsf{CCCS}}$ structure, we define the corresponding IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$, which uses Nova in a black-box manner.

$\underline{\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}}$: Output $(\mathsf{NIFS.G}(1^\lambda), \mathsf{IVC.G}(1^\lambda))$

$\underline{\mathcal{K}((\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{pp}_{\mathsf{IVC}}), F) \rightarrow (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}_{\mathsf{NIFS}}, F)$
2. Compute $(\mathsf{pk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{IVC}}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}_{\mathsf{IVC}}, \mathsf{step})$
3. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((F, \mathsf{pk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}, \mathsf{pk}_{\mathsf{IVC}}), (\mathsf{step}, \mathsf{pp}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{NIFS}}, \mathsf{vk}_{\mathsf{IVC}}))$.

$\underline{\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \mathit{\Pi}_i) \rightarrow \mathit{\Pi}_{i+1}}$:

1. Parse $\mathit{\Pi}_i$ as $(\mathit{\Pi}'_i, \mathsf{U}_i, \mathsf{W}_i)$
2. Compute $(\mathsf{u}_i, \mathsf{w}_i) \leftarrow \mathsf{trace}(F, (z_i, \omega_i))$ (where trace is defined as in Construction 3)
3. Compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi_{i+1}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}_{\mathsf{NIFS}}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
4. Compute $\mathit{\Pi}'_{i+1} \leftarrow \mathsf{IVC.P}(\mathsf{pk}_{\mathsf{IVC}}, i, (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{u}_{\perp}, z_0), (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{U}_i, z_i), (\mathsf{u}_i, \pi_{i+1}), \mathit{\Pi}'_i)$
5. Output $\mathit{\Pi}_{i+1} = (\mathit{\Pi}'_{i+1}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1})$

$\underline{\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \mathit{\Pi}_i) \rightarrow \{0, 1\}}$:

1. Parse $\mathit{\Pi}_i$ as $(\mathit{\Pi}'_i, \mathsf{U}_i, \mathsf{W}_i)$.
2. Check that $\mathsf{IVC.V}(\mathsf{vk}_{\mathsf{IVC}}, i, (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{u}_{\perp}, z_0), (\mathsf{vk}_{\mathsf{NIFS}}, \mathsf{U}_i, z_i), \mathit{\Pi}'_i) = 1$
3. Check that $(\mathsf{pp}_{\mathsf{NIFS}}, \mathsf{step}, \mathsf{U}_i, \mathsf{W}_i) \in \mathcal{R}_{\mathsf{LCCCS}}$

Theorem 3 (A simple construction for IVC). *Construction 4 is an IVC scheme.*

7 nlookup: A lookup argument for Nova

This section describes a lookup argument, which we refer to as nlookup, that is suitable for use in recursive arguments such as Nova, HyperNova, and others.

Suppose that there is a table T of size n. Now consider m variables $v_1, \ldots, v_m$ in a CCS instance and we wish to enforce that those values are contained in T.

A classic approach is to store T as a Merkle tree for which the circuit gets as public input a commitment. Then to prove that a certain value is in T, the prover could supply as non-deterministic advice to the circuit a Merkle proof of inclusion, and the circuit verifies the Merkle proof of inclusion. This unfortunately requires $O(m \cdot \log n)$ hash evaluations inside the circuit, which is prohibitive. Plookup [GW20] provides an approach where the number of constraints is $O(\max(m, n))$, which is acceptable when $m \approx n$. It is unsuitable in the context of recursive SNARKs such as Nova where a particular recursive step may perform $m << n$ lookup operations. A recent flurry of works (e.g., see cq [EFG22] for the latest in this line of work) consider the case where $m << n$, but it is unclear how to adapt them to the setting of recursive SNARKs without incurring high recursion overheads.

We provide a conceptually simple and yet efficient lookup argument, that we refer to as nlookup. For m lookups on a table of size n entries, nlookup requires $O(m \log n)$ multiplications and $O(\log n)$ hash operations inside a circuit (with small constants) and the prover performs $O(n)$ finite field operations. In particular, the prover does not commit to any additional polynomials. This lookup argument is *not* suitable for accelerating bitwise operations in the circuit model of computation, but it is a perfect tool for expressing finite state machines efficiently with Nova and HyperNova (e.g., see [Sol23, §2.4]).

nlookup in a nutshell. Without loss of generality, assume that $n = 2^\ell$. We can view T as a function from $\{0,1\}^\ell \to \mathbb{F}$. Furthermore, let $\widetilde{T}$ denote the unique multilinear extension of the function T. In other words, $\widetilde{T}$ is a multilinear polynomial in ℓ variables where the entries in the table are evaluations of $\widetilde{T}$ over the Boolean hypercube $\{0,1\}^\ell$. To prove m lookup operations, the prover specifies m evaluation points $q_1, \ldots, q_m$ over the Boolean hypercube such that $\widetilde{T}(q_i) = v_i$ for all $i \in [m]$. This requires $O(m \log n)$ Booleanity checks in the circuit to ensure that $q_i \in \{0,1\}^\ell$ for all $i \in [m]$.

We now devise a multi-folding scheme where the prover and the verifier fold the task of checking the correctness of m lookup operations into task of checking an evaluation of $\widetilde{T}$ at a single point in its domain. Furthermore, in our context, the circuit maintains a running claim about an evaluation of $\widetilde{T}$ and the folding scheme folds incoming lookup claims into this running claim.

Suppose that the running claim is $\widetilde{T}(q_r) \overset{?}{=} v_r$ for some $q_r \in \mathbb{F}^{\log n}$ and $v_r \in \mathbb{F}$. At initialization, q_r can be arbitrary and $v_r \leftarrow \widetilde{T}(q_r)$. Now, the folding scheme reduces the following claim to an evaluation of $\widetilde{T}$, where $\rho \in \mathbb{F}$ is picked by the verifier at random.

$$v_r + \sum_{i=\{1,\ldots,m\}} \rho^i \cdot v_i \overset{?}{=}$$
$$\sum_{j \in \{0,1\}^{\log n}} \widetilde{eq}(q_r, j) \cdot \widetilde{T}(j) + \sum_{i=\{1,\ldots,m\}} \rho^i \cdot \sum_{j \in \{0,1\}^{\log n}} \widetilde{eq}(q_i, j) \cdot \widetilde{T}(j)$$

The folding scheme applies the sum-check protocol and outputs a new claim about $\widetilde{T}(q_r') \overset{?}{=} v_r'$. The prover's work in the folding scheme is $O(n)$ finite field operations. The verifier's work in the non-interactive folding scheme is $O(\log n)$ hash and field operations. Furthermore, at the end of the sum-check protocol (i.e., inside the folding scheme), the verifier computes evaluations of m $\widetilde{eq}$ polynomials at a random point, this takes $O(m \cdot \log n)$ multiplications.

7.1 Details and security proofs

Definition 15 (Polynomial Evaluation Relation). *We define the polynomial evaluation relation $\mathcal{R}_{\mathsf{poly}}$ as follows. Let the public parameters consist of size parameter $\ell \in \mathbb{N}$. An $\mathcal{R}_{\mathsf{poly}}$ structure consists of $\widetilde{T}$, a multilinear polynomial in ℓ variables. An $\mathcal{R}_{\mathsf{poly}}$ instance is $(r, v) \in (\mathbb{F}^\ell, \mathbb{F})$ where r is an evaluation point and v is a claimed evaluation. An $\mathcal{R}_{\mathsf{poly}}$ witness is $\perp$. We define $\mathcal{R}_{\mathsf{poly}}$ as follows.*

$$\mathcal{R}_{\mathsf{poly}} = \left\{ ((\ell, \widetilde{T}), (r, v), \perp) \,\middle|\, \begin{array}{l} \ell \in \mathbb{N}, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell], (r, v) \in (\mathbb{F}^\ell, \mathbb{F}) \\ \widetilde{T}(r) = v \end{array} \right\}.$$

Definition 16 (Lookup Relation). *We define the lookup relation $\mathcal{R}_{\mathsf{lookup}}$ as follows. Let the public parameters consist of size parameter $\ell \in \mathbb{N}$. For vector $T \in \mathbb{F}^n$ (where $n = 2^\ell$), an $\mathcal{R}_{\mathsf{lookup}}$ structure consists of the corresponding multilinear extension in ℓ variables, $\widetilde{T}$. An $\mathcal{R}_{\mathsf{lookup}}$ instance consists of value $v \in \mathbb{F}$. An $\mathcal{R}_{\mathsf{lookup}}$ witness consists of index $q \in \{0,1\}^\ell$. We define $\mathcal{R}_{\mathsf{lookup}}$ as follows.*

$$\mathcal{R}_{\mathsf{lookup}} = \left\{ ((\ell, \widetilde{T}), v, q) \;\middle|\; \begin{array}{l} \ell \in \mathbb{N}, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell], v \in \mathbb{F}, q \in \{0,1\}^\ell \\ \widetilde{T}(q) = v \end{array} \right\}.$$

We now provide a multi-folding between two relations, a polynomial evaluation instance and a collection of lookup instances.

Construction 5 (A multi-folding scheme for lookup instances). We construct a multi-folding scheme for $(\mathcal{R}_{\mathsf{poly}}, \mathcal{R}_{\mathsf{lookup}}, \mathsf{compat}, \mu = 1, \nu)$ for arbitrary $\nu \in \mathbb{N}$.

$\underline{\mathsf{compat}(\mathsf{s}_1, \mathsf{s}_2) \to \{\mathsf{true}, \mathsf{false}\}}$

1. If $\mathsf{s}_1 = \mathsf{s}_2$, then return true, otherwise, return false.

We define the generator and the encoder as follows.

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$:
  1. Sample size bound $\ell \in \mathbb{N}$.
  2. Output $\mathsf{pp} = \ell$.
- $\mathcal{K}(\mathsf{pp} = \ell, \widetilde{T} \in \mathbb{F}^1[X_1, \ldots, X_\ell]) \to (\mathsf{pk}, \mathsf{vk})$: Output $(\mathsf{pk}, \mathsf{vk}) = ((\mathsf{pp}, \widetilde{T}), \mathsf{pp})$.

The prover takes as input $\mathsf{pk} = (\mathsf{pp} = \ell, \widetilde{T})$ and the verifier take as input $\mathsf{vk} = \mathsf{pp} = \ell$. The verifier $\mathcal{V}$ takes a polynomial evaluation instance (q_r, v_r) and a vector of lookup instances $(v_1, \ldots, v_m)$ The prover $\mathcal{P}$, in addition to the instances, takes witnesses to the lookup instances $(q_1, \ldots, q_m)$.

The prover and the verifier proceed as follows.

1. $\mathcal{P} \to \mathcal{V}$: $(q_1, \ldots, q_m)$.
2. $\mathcal{V}$: Check that for all $i \in [m]$, $q_i \in \{0,1\}^\ell$.
3. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}$ and send it to $\mathcal{P}$.
4. $\mathcal{V}$: Sample $q_r' \xleftarrow{\$} \mathbb{F}^s$.
5. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(q_r') \rangle (g, \ell, 2, v_r + \sum_{i \in [m]} \rho^i \cdot v_i)$, where:

$$g(x) := \widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x)$$

6. $\mathcal{P} \to \mathcal{V}$: v'_r, where $v'_r = \widetilde{T}(q'_r)$.
7. $\mathcal{V}$: Compute $e \leftarrow \widetilde{eq}(q_r, q'_r)$ and $e_i \leftarrow \widetilde{eq}(q_i, q'_r)$ for all $i \in [m]$. Abort if

$$c \neq e \cdot v'_r + \sum_{i \in [m]} \rho^i \cdot e_i \cdot v'_r.$$

8. $\mathcal{V}, \mathcal{P}$: Output the folded polynomial evaluation instance (q'_r, v'_r).

Theorem 4. *Construction 5 is a public-coin multi-folding scheme for* $(\mathcal{R}_{\mathsf{poly}}, \mathcal{R}_{\mathsf{lookup}}, \mathsf{compat}, \mu = 1, \nu)$ *for arbitrary* $\nu \in \mathbb{N}$.

Lemma 9 (Perfect Completeness). *Construction 5 satisfies perfect completeness.*

Proof. Consider public parameters $\mathsf{pp} = \ell \leftarrow \mathcal{G}(1^\lambda)$. Consider a common structure $\mathsf{s}_1 = \mathsf{s}_2 = \widetilde{T} \in \mathbb{F}[X_1, \dots, X_\ell]$. Consider the prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) = (\widetilde{T}, \perp) \leftarrow \mathcal{K}(\mathsf{pp}, \widetilde{T})$. Suppose that the prover and the verifier are provided an instance in $\mathcal{R}_{\mathsf{poly}}$

$$(q_r, v_r)$$

and a vector of $\mathcal{R}_{\mathsf{lookup}}$ instances

$$(v_1, \dots, v_m).$$

Suppose that the prover is additionally provided with the corresponding satisfying witnesses for the $\mathcal{R}_{\mathsf{lookup}}$ instances

$$(q_1, \dots, q_m).$$

By the satisfiability of the input instances we have that $v_r = \widetilde{T}(q_r)$ and $v_i = \widetilde{T}(q_i)$ for all $i \in [m]$.

Therefore, for $\rho \in \mathbb{F}$, we have that

$$\begin{aligned} v_r + \sum_{i \in [m]} \rho^i \cdot v_i &= \widetilde{T}(q_r) + \sum_{i \in [m]} \rho^i \cdot \widetilde{T}(q_i) && \text{By precondition.} \\ &= \sum_{x \in \{0,1\}^\ell} \left(\widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x) \right) && \text{By Lemma 1.} \\ &= g(x) && \text{By definition.} \end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have that $c = g(q'_r)$ Thus, for $v'_r = \widetilde{T}(q'_r)$, $e = \widetilde{eq}(q_r, q'_r)$, and $e_i = \widetilde{eq}(q_i, q'_r)$ for all $i \in [m]$, we have that

$$\begin{aligned}c &= g(q_r') \\ &= \widetilde{eq}(q_r, q_r') \cdot \widetilde{T}(q_r') + \sum_{i \in [m]} \rho^i \cdot \widetilde{eq}(q_i, q_r') \cdot \widetilde{T}(q_r') \\ &= e \cdot v_r' + \sum_{i \in [m]} \rho^i \cdot e_i \cdot v_r'\end{aligned}$$

Therefore, we have that the verifier does not abort.

By construction, we have that $v_r' = \widetilde{T}(q_r')$. Therefore, the folded polynomial evaluation instance is satisfying.

□

Lemma 10 (Knowledge Soundness). *Construction 5 satisfies knowledge soundness assuming that* $|\mathbb{F}| = \Theta(2^\lambda)$.

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structure and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} = \ell \leftarrow \mathcal{G}(1^\lambda)$. Suppose on input pp and random tape r, the adversary $\mathcal{A}$ picks a structure $\mathsf{s}_1 = \mathsf{s}_2 = \widetilde{T} \in \mathbb{F}[X_1, \dots, X_\ell]$, an $\mathcal{R}_{\mathsf{poly}}$ instance (q_r, v_r), a vector of $\mathcal{R}_{\mathsf{lookup}}$ instances $(v_1, \dots, v_m)$, and some auxiliary state st.

We construct an extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances. It works as follows.

On input pp and r, $\mathcal{E}$ first obtains the following tuple from $\mathcal{A}$:

$$(\widetilde{T}, (q_r, v_r), (v_1, \dots, v_m), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{r}).$$

$\mathcal{E}$ then computes $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \widetilde{T})$. Next, $\mathcal{E}$ runs

$$((q_r', v_r'), \perp) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (q_r, v_r), (v_1, \dots, v_m), \mathsf{st})$$

and obtains the first message $(q_1, \dots, q_m)$ from $\mathcal{P}^*$ by parsing the corresponding transcript. The extractor $\mathcal{E}$ outputs $(\perp, (q_1, \dots, q_m))$ as the witness. Because the extractor only runs $\mathcal{P}^*$ once, it runs in expected polynomial-time.

We must argue that $\perp$ is a satisfying $\mathcal{R}_{\mathsf{poly}}$ witness for (q_r, v_r), and that $(q_1, \dots, q_m)$ are satisfying $\mathcal{R}_{\mathsf{lookup}}$ witnesses for the input instances $(v_1, \dots, v_m)$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Suppose that we have that the witness $\perp$ output by $\mathcal{P}^*$ is satisfying for the corresponding verifier's output (q_r', v_r') with probability ϵ. By definition, this means that

$$v_r' = \widetilde{T}(q_r') \tag{8}$$

with probability ϵ. Moreover, this means that the verifier does not abort with probability at least ϵ, and thus we have the following:

$$\begin{aligned}
c &= e \cdot v_r' + \sum_{i\in[m]} \rho^i \cdot e_i \cdot v_r' \\
&= \widetilde{eq}(q_r, q_r') \cdot v_r' + \sum_{i\in[m]} \rho^i \cdot \widetilde{eq}(q_i, q_r') \cdot v_r' && \text{By the verifier's computation.} \\
&= \widetilde{eq}(q_r, q_r') \cdot \widetilde{T}(q_r') + \sum_{i\in[m]} \rho^i \cdot \widetilde{eq}(q_i, q_r') \cdot \widetilde{T}(q_r') && \text{By Equation 8.} \\
&= g(q_r') && \text{By definition.}
\end{aligned}$$

with probability ϵ.

Then, by the soundness of the sum-check protocol, we must have that

$$\begin{aligned}
v_r + \sum_{i\in[m]} \rho^i \cdot v_i &= \sum_{x\in\{0,1\}^\ell} g(x) \\
&= \sum_{x\in\{0,1\}^\ell} \left(\widetilde{eq}(q_r, x) \cdot \widetilde{T}(x) + \sum_{i\in[m]} \rho^i \cdot \widetilde{eq}(q_i, x) \cdot \widetilde{T}(x) \right) && \text{By definition.} \\
&= \widetilde{T}(q_r) + \sum_{i\in[m]} \rho^i \cdot \widetilde{T}(q_i) && \text{By Lemma 1.}
\end{aligned}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. By the Schwartz-Zippel lemma over ρ, this implies that $v_r = \widetilde{T}(q_r)$ and $v_i = \widetilde{T}(q_i)$ for all $i \in [m]$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Moreover, by the verifier's initial check, we have that $q_i \in \{0,1\}^\ell$ for all $i \in [m]$. Therefore, we have that

$$(\mathsf{pp}, \widetilde{T}, (q_r, v_r), \perp) \in \mathcal{R}_{\mathsf{poly}}$$

and

$$(\mathsf{pp}, \widetilde{T}, (v_1, \ldots, v_m), (q_1, \ldots, q_m)) \in \mathcal{R}^{(m)}_{\mathsf{lookup}}.$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Because the extractor $\mathcal{E}$ outputs the initial message from the prover, we have that the extractor succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. □

Acknowledgments

We thank Justin Drake, Ariel Gabizon, Bryan Parno, Carlos Pérez, Drew Stone, Justin Thaler, and Ioanna Tzialla for helpful conversations and comments on a prior version of this paper.

References

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.

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

BDFG21. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. In *CRYPTO*, 2021.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT*, 2020.

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

BGH20. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo2, 2020. `https://github.com/zcash/halo2`.

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

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT*, 2020.

Dra22. Justin Drake. ZK Whiteboard Sessions – Module Fourteen: Nova Crash Course with Justin Drake. `https://www.youtube.com/watch?v=SwonTtOQzAk`, 2022.

EFG22. Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive*, 2022.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

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

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. ePrint Report 2019/953, 2019.

KMT22. Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626, 2022.

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

KS22. Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, 2022.

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.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updateable structured reference strings. In *CCS*, 2019.

Moh23. Nicolas Mohnblatt. Sangria: a folding scheme for PLONK. `https://geometry.xyz/notebook/sangria-a-folding-scheme-for-plonk`, 2023.

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.

Sol23. Tomer Solberg. A brief history of lookup arguments. `https://github.com/ingonyama-zk/papers/blob/main/lookups.pdf`, 2023.

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

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.

Zha23. Zhenfei Zhang. Origami: Fold a Plonk for Ethereum's VDF. Cryptology ePrint Archive, 2023.