SuperNova: Proving universal machine executions without universal circuits

Abhiram Kothapalli[†] Srinath Setty[⋆]

[†]Carnegie Mellon University [⋆]Microsoft Research

Abstract. This paper introduces SuperNova, a new recursive proof system for incrementally producing succinct proofs of correct execution of programs on a stateful machine with a particular instruction set (e.g., EVM, RISC-V). A distinguishing aspect of SuperNova is that the cost of proving a step of a program is proportional only to the size of the circuit representing the instruction invoked by the program step. This is a stark departure from prior works that employ universal circuits where the cost of proving a program step is proportional at least to the sum of sizes of circuits representing each supported instruction—even though a particular program step invokes only one of the supported instructions. Naturally, SuperNova can support a rich instruction set without affecting the per-step proving costs. SuperNova achieves its cost profile by building on Nova, a prior high-speed recursive proof system, and leveraging its internal building block, folding schemes, in a new manner. We formalize SuperNova's approach as a way to realize *non-uniform IVC*, a generalization of IVC. Furthermore, SuperNova's prover costs and the recursion overhead are the same as Nova's, and in fact, SuperNova is equivalent to Nova for machines that support a single instruction.

1 Introduction

We study the problem of producing succinct cryptographic proofs of correct execution of programs on a stateful machine with a particular instruction set (e.g., EVM, RISC-V, Wasm).[1]

As a motivating example, such a proof system enables "replicated systems without replicated execution" or "rollups" [22, 25, 36]. In particular, it enables replicated systems such as blockchains to safely "outsource" transaction processing to centralized infrastructure (e.g., the cloud) without trust: the cloud executes the blockchain's virtual machine with a batch of transactions and then produces a succinct proof that it correctly executed the virtual machine. The replicated system only needs to replicate the verification of proofs, which reduces per-transaction costs as long as the cost of producing a proof for a batch of transactions and the replicated verification of the proof is cheaper than the replicated execution of the batch of transactions.

[1] The paper uses "proofs" and "arguments" interchangeably. In both cases, we mean an argument where soundness holds only under cryptographic hardness assumptions.

In more detail, we seek a cryptographic proof system consisting of a *prover* and a *verifier*, where given the specification of a (virtual) machine and a program designed to run on such a machine, the prover can produce a proof to convince the correct execution of the program to the verifier, with the following requirements (the program can potentially take a secret, non-deterministic witness from the prover and some public inputs from the verifier).

- *Succinctness:* The size of a proof and the time to verify a proof are at most polylogarithmic in the time to execute the program.
- *Zero-knowledge:* The proof does not reveal anything beyond what is implied by the correct execution of a specified program.
- *"A la carte" cost profile:* The cost of proving a step of a program execution is proportional only to the size of the circuit representing the instruction invoked by the program step.
- *Incremental proof generation:* The prover can produce a separate proof for each step of the program execution independently (and potentially in parallel), and then recursively combine those proofs into a single proof (as we discuss next, this offers substantial benefits).

Benefits of incremental proof systems. As a core advantage, incremental proofs do not require static bounds on loop iterations in a program and therefore are well-suited for stateful computations with dynamic control flow. Furthermore, incremental proof generation imposes minimal memory overhead: at each step of a program execution, the prover only needs space proportional to the space needed to execute that particular step.

Moreover, incremental proof systems *can* enable distribution and parallelization of proof generation. Specifically, the prover can first execute a desired program on a machine, recording inputs, outputs, and state changes. Then, in parallel and on different CPUs, for each step in the execution, the prover can produce a *separate* proof proving the correct execution of the step. The prover can then use the incremental capabilities of the proof system to aggregate different proofs (e.g., in a binary tree style manner) into a single proof to be checked by the verifier. Such parallel proving is especially important for large-scale applications of proof systems such as rollups [22, 25, 36] where the size of the circuit whose satisfiability is proven can be on the order of a billion gates or more.

Finally, state-of-the-art incremental proof systems, such as Nova [21], offer a new paradigm for reducing proof generation costs [12]. Specifically, to prove an iterative computation consisting of N steps, Nova recursively folds together N circuit satisfiability instances into one instance, and then invokes a general-purpose proof system (e.g., [29]) to prove that folded instance. Compared to directly applying a general-purpose proof system to prove N steps, Nova's approach to apply the fold operation and then invoke a general-purpose proof system incurs at least an order of magnitude lower resource costs.[2]

[2] This assumes a sufficiently large computation (e.g., at least 10,000 constraints) performed at each step to amortize recursion overheads.

1.1 Prior approaches and challenges

A classic approach to prove machine executions is based on incrementally verifiable computation (IVC) [33]. In this approach, one first designs a universal circuit (e.g., [1, 4, 16, 23, 28]) that can execute any instruction supported by the machine (the circuit implements the fetch-decode-execute loop of the corresponding machine). To prove the correct execution of programs on the corresponding machine, it suffices to recursively prove repeated invocations of this circuit on an input program and memory state [3]. In more detail, at each step of the program execution, the prover employs a succinct non-interactive argument of knowledge (SNARK) [5,15,19,24] to prove the correct execution of an augmented circuit, where the augmented circuit contains an invocation of the universal circuit and a verifier circuit that verifies the SNARK produced by the prover for the previous step of program execution. Unfortunately, the cost of proving a program's step is proportional to the size of the universal circuit—even though the corresponding program step invokes only one of the supported instructions.

Given the high costs imposed by universal circuits, designers of these machines aim to employ a minimal instruction set, to keep the size of the universal circuit and thereby the cost of proving a program step minimal [2,4,16]. However, this is a not a panacea: for real applications, one needs to execute an enormous number of iterations of the minimal circuit (e.g., billions of iterations), making the prover's work largely untenable. This also means that emulating real programs that target *existing* virtual machines with rich instruction sets (e.g., EVM, RISC-V, Wasm) via a machine with a minimal instruction sets would incur enormous costs.

Buffet [34], building on Pantry [10] and Ben-Sasson et al. [4], avoids the high cost of universal circuits yet supports a general class of programs. For example, Buffet supports any program in the C programming language as long as it neither invokes goto statements nor uses function pointers. Furthermore, Buffet provides an "a la carte" cost profile where the prover's proof generation costs are proportional only to the sum of sizes of circuits of the operations invoked by the program execution. However, Buffet adopts a "line-by-line compilation" approach [9, 10, 26, 32], where it unrolls programs into non-uniform circuits by translating each program statement into a concise set of constraints. Unfortunately, this approach requires static bounds on program execution lengths. More importantly, it is unclear how to prove the satisfiability of unrolled non-uniform circuits in an incremental fashion. Furthermore, although general, it is unclear how to use Buffet's approach to prove program executions on a stateful machine *without* producing a non-uniform circuit for each program. Having a separate circuit for each program is undesirable in practice as it is not clear how in that model one program can invoke another program (a la "composability").

A subsequent work, called vRAM [37], achieves Buffet-like costs for program executions on vnTinyRAM [3], a RAM machine with a minimal instruction set. In particular, during program execution, at the granularity of a processor cycle, vRAM uses a "trimmed" version of the vnTinyRAM universal circuit where the trimmed version eliminates circuit elements corresponding to instructions that were not invoked. Unfortunately, like Buffet, this approach is not incremental.

Specifically, it requires proving that certain global invariants hold over the entire trace of program execution (e.g., to prove that the trimmed version of the circuit is correct), using randomized fingerprinting techniques. As with Buffet, it is unclear how to prove these global invariants hold in an incremental fashion. Furthermore, this approach reveals, for each program execution, the number of invocations of each instruction supported by the machine to the verifier, so vRAM's approach does not ensure zero-knowledge.

MIRAGE [20] adapts vRAM's techniques in the context of Groth's SNARK [17] (vRAM uses a CMT-based argument [11]). Like vRAM, MIRAGE still relies on proving invariants over the entire execution trace via fingerprinting techniques, making its techniques incompatible with incremental proof systems.

Our solution: SuperNova We describe SuperNova, a new incremental proof system for proving arbitrary stateful machine executions, where the cost of proving a step of a program is proportional only to size of the circuit representing the requested instruction. SuperNova can be viewed as a way to achieve Buffet's and vRAM's "a la carte" cost profile (i.e., pay the prover's costs only for instructions that were executed) in the context of incremental proof systems but without any of their downsides.

SuperNova achieves these results by leveraging folding schemes, a cryptographic primitive introduced and employed by Nova [21] to construct IVC [33]. We formalize SuperNova's approach as a way to realize *non-uniform IVC*, a generalization of IVC that formally captures the "a la carte" cost profile. Indeed, SuperNova can be viewed as a generalization of Nova: whereas Nova supports machines with a single instruction, SuperNova supports machines with an arbitrary instruction set. Perhaps surprisingly, this generality does *not* add (substantial) overheads: SuperNova's recursion overhead and the prover's costs are the similar to that of Nova. Furthermore, the prover's cost at each step is dominated by two multiexponentiations of size proportional to the size of the circuit representing the executed instruction.

As presented in this paper, SuperNova does not immediately support parallel proof generation. There exists a generic compiler [6] to transform constructions such as SuperNova into a form that does support parallel proving. We leave it to the near-term future work to provide a solution tailored to SuperNova.

1.2 A technical overview of SuperNova

SuperNova provides a realization of non-uniform IVC, a generalization of IVC [33] that we introduce. As we discuss below, non-uniform IVC implies succinct proofs of program execution on a specified machine. In particular, one can define the behavior of the stateful machine by specifying its instruction set and state passed from one instruction to the next. SuperNova can then prove correct executions of programs designed to run on such a stateful machine.

Below, we first provide an overview of the computational model supported by non-uniform IVC. We then describe how that computational model can be used to build stateful machines ranging from "ASICs" that perform a highly specific task (e.g., execute or verify certain iterations of a delay function, verify cryptocurrency payment operations) to general-purpose CPUs (e.g., RISC-V) and virtual machines (e.g., EVM). Finally, we discuss how to achieve non-uniform IVC by extending Nova.

Computational model of non-uniform IVC. Consider a collection of $\ell + 1$ non-deterministic, polynomial time computable functions $(\{F_1, \ldots, F_\ell\}, \varphi)$, where $\ell \geq 1$. Suppose that each function F_j, where $1 \leq j \leq \ell$ takes s inputs and produces s outputs, where $s > 0$; F_j can additionally take arbitrary non-deterministic input. Furthermore, φ is a function that takes s inputs and arbitrary non-deterministic input and produces an element of $\mathbb{Z}^*_{\ell+1}$ (i.e., the set $\{1, \ldots, \ell\}$). In SuperNova's realization of non-uniform IVC, each of these $\ell + 1$ functions are specified with R1CS, a popular NP-complete problem that is implicit in the QAPs formalism [14, 31].

A non-uniform IVC scheme enables a prover to incrementally prove that it has performed an n-step computation with an initial input z_0 to produce an output z_n. In particular, at step i, the prover proves that it has applied F_j on input (z_{i-1}, ω_{i-1}) to produce an output z_i, where z_{i-1} is output of step $i-1$, ω_{i-1} is a (potentially secret) non-deterministic input from the prover for step i, and $j = \varphi(z_{i-1}, \omega_{i-1})$. That is, φ selects one of the possible ℓ functions to apply at step i using inputs to step i. A bit more concisely, for a specified $(\{F_1, \ldots, F_\ell\}, \varphi)$ and (n, z_0, z_n), the prover proves the knowledge of a set of non-deterministic values $\{\omega_0, \ldots, \omega_{n-1}\}$ and $\{z_1, \ldots, z_{n-1}\}$ such that $\forall i \in \{0, \ldots, n-1\}$, we have that $z_{i+1} = F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$.

Devising stateful machines using non-uniform IVC. We now describe how one can use non-uniform IVC to prove program executions on stateful machine.

In general, with SuperNova, one can build a machine with a *custom* instruction set and then prove program executions on such a machine. For example, one can build a RAM machine where one of the instructions performs an application-specific task such as validating a cryptocurrency transaction and appropriately updating balances in a Merkle tree based key-value store. More generally, one can support an instruction that performs certain expensive operations (e.g., a signature verification, a hash computation, etc.). Fortunately, because of SuperNova's cost profile, the prover's cost at a particular step is proportional only to the size of the circuit to encode the invoked instruction. In particular, the per-step proving cost is independent of the sizes of circuits of "uninvoked" instructions.

A VDF machine. As a warm-up, we use non-uniform IVC to prove executions of a machine whose instruction set includes invocations of a delay function (a function that takes non-trivial sequential time to compute). This realizes a verifiable delay function (VDF) [8, 35]. In more detail, consider a stateful machine that supports a single instruction i.e., $\ell = 1$. In particular, F_1 executes a certain, fixed number of iterations of a delay function (e.g., MinRoot [18]).[3] Furthermore,

[3] For MinRoot [18], the instruction verifies that the function was executed correctly with 3 arithmetic constraints and some non-deterministic advice; executing the MinRoot computation itself would need hundreds of constraints, which is unnecessary.

$\varphi(\_,\_) = 1$ since there is only one instruction. Finally, z_0 is the initial input to the VDF and z_i is the output the VDF after i executions of the delay instruction.

A RAM machine. We now show how to use non-uniform IVC to prove program executions on a RAM machine (e.g., RISC-V).

Let the RAM machine support ℓ instructions, s registers of width w bits, and a memory of size 2^w. Let $\{F_1, \ldots, F_\ell\}$ denote a collection of non-deterministic functions, where a function F_j verifies the input/output behavior of instruction j supported by the machine. Each instruction takes certain input values (which we specify next) and arbitrary non-deterministic input, and outputs certain values. In particular, the input of each F_j consists of $s + 1$ field elements, where the first entry holds a commitment to a memory (e.g., the root of a Merkle tree with 2^w leaves) that stores both a program and its state, and the remaining entries are the values of s registers. Furthermore, the output of each F_j consists of $s + 1$ field elements that are updated values of the provided input.

Without loss of generality, let the first of the s registers be designated as the "program counter". We define φ as follows: For step i, given input z_{i-1} and the non-deterministic input ω_{i-1}, $\varphi(z_{i-1}, \omega_{i-1})$ picks the instruction in the memory (whose commitment is at $z_{i-1}[1]$) at address in the program counter register $z_{i-1}[2]$. The initial state $z_0[1]$ holds a commitment to the verifier's desired memory of size 2^w with its program stored in it and the rest of z_0 contains the verifier's desired initial values of the machine's registers.

SuperNova's mechanisms to achieve non-uniform IVC. SuperNova leverages folding schemes [21], a cryptographic primitive that enables a prover and a verifier to fold two N-sized NP instances into a single N-sized NP instance such that the folded instance is satisfiable only if the original instances are satisfiable. In particular, SuperNova leverages a folding scheme for (a variant of) R1CS.

At each step, SuperNova's prover folds an R1CS instance (and its associated witness) representing the prior step of the program execution into a running instance (and running witness). Furthermore, the prover feeds that instance and certain advice generated by the folding scheme to an *augmented circuit*, which contains a *verifier circuit* in addition to one of the ℓ functions in $\{F_1, \ldots, F_\ell\}$. In particular, the verifier circuit contains two components: (1) the verifier of the non-interactive folding scheme for R1CS (to fold the incoming instance into a running instance); and (2) a circuit for computing φ.

In more detail, SuperNova uses multiple running instances, one for each instruction supported by the machine. As such, the verifier circuit folds an incoming instance into an appropriate running instance. The choice of which running instance to use is constrained by φ embedded in the verifier circuit. A natural design of the verifier circuit makes the size of the verifier circuit scale linearly with ℓ. We later discuss how to use offline memory checking techniques [7, 22, 30] to make the verifier circuit size independent of ℓ.

At the end of N steps, the prover holds ℓ running instances and an R1CS instance that represents the last step of the program execution. To prove these, the prover can send the associated witnesses, which the verifier can check. Unfortunately, the proof size, while *not* dependent on the number of steps of the program execution, it is proportional to the sum of sizes of circuits for functions in $(\{F_1, \ldots, F_\ell\}, \varphi)$, so it is not *concretely* small. Furthermore, the proof is not zero-knowledge. As in Nova, one can obtain compressed proofs and achieve zero-knowledge by invoking a general-purpose zkSNARK (e.g., Spartan [29]).

2 Preliminaries and prior results

Let $\mathbb{F}$ denote a finite field with $|\mathbb{F}| = 2^{\Theta(\lambda)}$, where λ is the security parameter. Let $\cong$ denote computational indistinguishability with respect to a PPT adversary. We globally assume that generator algorithms that produce public parameters are additionally provided appropriate size bounds.

2.1 Incrementally verifiable computation (IVC)

IVC [33] is a cryptographic proof system that allows producing proofs of knowledge of witnesses to a non-deterministic computation in an incremental fashion. We provide a formal definition below.

Definition 1 (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*

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

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

(i) Perfect Completeness: For any PPT adversary $\mathcal{A}$

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

where F is a polynomial-time computable function.

(ii) Knowledge Soundness: For any constant $n \in \mathbb{N}$, and expected polynomial-time adversaries $\mathcal{P}^*$ there exists expected polynomial-time extractor $\mathcal{E}$ such that

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

(iii) Succinctness: The size of an IVC proof Π is independent of the number of iterations n.

2.2 A commitment scheme for vectors

Throughout this paper, we employ a commitment scheme for vectors over $\mathbb{F}$ that is additively homomorphic and provides succinct commitments. Below, we provide a formal definition.

Definition 2 (A commitment scheme for vectors). *A commitment scheme for $\mathbb{F}^m$ is a tuple of three protocols with the following interface:*

- $\leftarrow \mathsf{Gen}(1^\lambda, m) \rightarrow \mathsf{pp}$*: on input security parameter λ, and a length parameter $m \in \mathbb{N}$, produces public parameters* pp.
- $\mathsf{Com}(\mathsf{pp}, x, r) \rightarrow C$*: on input* pp*, a vector $x \in \mathbb{F}^m$, and randomness $r \in \mathbb{F}$, produces a commitment C.*
- $\mathsf{Open}(\mathsf{pp}, C, x, r) \rightarrow \{0, 1\}$*: on input* pp*, a commitment C, a vector $x \in \mathbb{F}^m$, and randomness $r \in \mathbb{F}$ verifies the opening of commitment C to $x \in \mathbb{F}^m$ and $r \in \mathbb{F}$; outputs 1 if verification passes and 0 otherwise.*

A commitment scheme $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ *satisfies the following requirements*

(i) Binding: For any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{l|l} b_0 = b_1 = 1, & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, m), \\ x_0 \neq x_1 & (C, x_0, x_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{pp}), \\ & b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, C, x_0, r_0), \\ & b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, C, x_1, r_1) \end{array}\right]$$

(ii) Hiding: For all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the following probability is $\mathsf{negl}(\lambda)$*:*

$$\left| \frac{1}{2} - \Pr\left[\begin{array}{l|l} b = \bar{b} & (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pp}), \\ & b \leftarrow_R \{0, 1\}, r \leftarrow_R \mathbb{F}, \\ & C \leftarrow \mathsf{Com}(\mathsf{pp}, x_b, r), \\ & \bar{b} \leftarrow \mathcal{A}_1(\mathsf{st}, C) \end{array}\right] \right|$$

If hiding holds for all adversaries, then the commitment is statistically hiding.

Definition 3 (Additive homomorphism). *A commitment scheme for vectors over* $\mathbb{F}^m$*,* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$*, is additively homomorphic if for all public parameters* pp *produced from* $\mathsf{Gen}(1^\lambda, m)$*, and for any* $x_1, x_2 \in \mathbb{F}^m$ *and for any* $r_1, r_2 \in \mathbb{F}$*,* $\mathsf{Com}(\mathsf{pp}, x_1, r_1) + \mathsf{Com}(\mathsf{pp}, x_2, r_2) = \mathsf{Com}(\mathsf{pp}, x_1 + x_2, r_1 + r_2)$*.*

Definition 4 (Succinctness). *A commitment scheme for vectors over* $\mathbb{F}^m$*,* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$*, provides succinct commitments if for all public parameters* pp *produced from* $\mathsf{Gen}(1^\lambda, m)$*, and any* $x \in \mathbb{F}^m$ *and* $r \in \mathbb{F}$*,* $|\mathsf{Com}(\mathsf{pp}, x, r)| = O_\lambda(\mathsf{polylog}(|x|))$*.*

Remark 1 (Pedersen commitment scheme). An example commitment scheme is Pedersen's commitment scheme [27], which relies on cryptographic groups where the discrete logarithm problem is hard. It provides a commitment scheme where a commitment to a vector is a single group element.

2.3 Folding schemes

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 into the task of checking a single instance in $\mathcal{R}$.

Kothapalli et al. [21] devise a folding scheme for a variant of a popular NP-complete relation, which they call *committed relaxed R1CS*, and use it to build an IVC scheme. Similarly, as we discuss later, SuperNova uses the same folding scheme to realize a generalization of IVC.

Below, we first formally define folding schemes and then state prior constructions of folding schemes that SuperNova builds upon.

Definition 5 (Folding scheme). *Consider a relation* $\mathcal{R}$ *over public parameters, structure, instance, and witness tuples. A folding scheme for* $\mathcal{R}$ *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*.*
- $\mathcal{K}(\mathsf{pp}, \mathsf{S}) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: on input* pp*, and a common structure* S *between the instances to be folded, outputs a prover key* pk *and a verifier key* vk*.*
- $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2)) \rightarrow (u, w)$*: on input instance-witness tuples* (u_1, w_1) *and* (u_2, w_2)*, outputs a new instance-witness tuple* (u, w) *of the same size.*
- $\mathcal{V}(\mathsf{vk}, u_1, u_2) \rightarrow u$*: on input instances* u_1 *and* u_2*, outputs a new instance* u*.*

Let

$$(u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk}) \rangle (u_1, u_2)$$

denote the the verifier's output instance u *and the prover's output witness* w *from the interaction of* $\mathcal{P}$ *and* $\mathcal{V}$ *on witnesses* (w_1, w_2)*, prover key* pk*, verifier key* vk *and instances* (u_1, u_2)*. Likewise, let*

$$\mathsf{tr} = \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk}) \rangle (u_1, u_2)$$

denote the corresponding interaction transcript. We call a transcript an accepting transcript if $\mathcal{P}$ outputs a satisfying folded witness w for the folded instance u. A folding scheme satisfies the following requirements.

(i) Perfect Completeness: For all PPT adversaries $\mathcal{A}$

$$\Pr\left[(\mathsf{pp},\mathsf{S},u,w)\in\mathcal{R}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda),\\(\mathsf{S},(u_1,w_1),(u_2,w_2))\leftarrow\mathcal{A}(\mathsf{pp}),\\(\mathsf{pp},\mathsf{S},u_1,w_1),(\mathsf{pp},\mathsf{S},u_2,w_2)\in\mathcal{R},\\(\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},\mathsf{S}),\\(u,w)\leftarrow\langle\mathcal{P}(\mathsf{pk},w_1,w_2),\mathcal{V}(\mathsf{vk})\rangle(u_1,u_2)\end{array}\right]=1.$$

(ii) Knowledge Soundness: For any expected polynomial-time adversary $\mathcal{P}^*$ there is an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l}(\mathsf{pp},\mathsf{S},u_1,w_1)\in\mathcal{R},\\(\mathsf{pp},\mathsf{S},u_2,w_2)\in\mathcal{R}\end{array}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda),\\(\mathsf{S},(u_1,u_2))\leftarrow\mathcal{P}^*(\mathsf{pp}),\\(w_1,w_2)\leftarrow\mathcal{E}(\mathsf{pp})\end{array}\right]\geq$$
$$\Pr\left[(\mathsf{pp},\mathsf{S},u,w)\in\mathcal{R}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda),\\(\mathsf{S},(u_1,u_2))\leftarrow\mathcal{P}^*(\mathsf{pp}),\\(\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},\mathsf{S}),\\(u,w)\leftarrow\langle\mathcal{P}^*(\mathsf{pk}),\mathcal{V}(\mathsf{vk})\rangle(u_1,u_2)\end{array}\right]-\mathsf{negl}(\lambda)$$

(iii) Efficiency: The communication costs and $\mathcal{V}$'s computation are lower in the case where $\mathcal{V}$ participates in the 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 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 6 (Non-interactive). *A 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 an output of $\mathcal{P}$ and as an input to $\mathcal{V}$.*

Definition 7 (Public coin). *A 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.*

R1CS is an NP-complete problem implicit in the work of GGPR [14]. Below, we recall its definition and its folding-friendly variant, committed relaxed R1CS.

Definition 8 (R1CS). *Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m,n,\ell\in\mathbb{N}$ where $m>\ell$. The R1CS structure consists of sparse matrices $A,B,C\in\mathbb{F}^{m\times m}$ with at most $n=\Omega(m)$ non-zero entries in each matrix. An instance $\mathsf{x}\in\mathbb{F}^\ell$ consists of public inputs and outputs and is satisfied by a witness $W\in\mathbb{F}^{m-\ell-1}$ if $(A\cdot Z)\circ(B\cdot Z)=C\cdot Z$, where $Z=(W,\mathsf{x},1)$.*

Definition 9 (Committed relaxed R1CS). *Consider a finite field $\mathbb{F}$ and a commitment scheme* Com *over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m-\ell-1$ respectively. The committed relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m\times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. A committed relaxed R1CS instance is a tuple $(\overline{E}, u, \overline{W}, \mathsf{x})$, where $\overline{E}$ and $\overline{W}$ are commitments, $u \in \mathbb{F}$, and $\mathsf{x} \in \mathbb{F}^{\ell}$ are public inputs and outputs. An instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ is satisfied by a witness $(E, r_E, W, r_W) \in (\mathbb{F}^m, \mathbb{F}, \mathbb{F}^{m-\ell-1}, \mathbb{F})$ if $\overline{E} = \mathsf{Com}(\mathsf{pp}_E, E, r_E)$, $\overline{W} = \mathsf{Com}(\mathsf{pp}_W, W, r_W)$, and $(A\cdot Z)\circ(B\cdot Z) = u\cdot(C\cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.*

Below, we state a key prior result on folding schemes for NP. We first describe the result in the random oracle model, and then describe the result that is heuristically secure in the plain model.

Lemma 1 (A folding scheme for committed relaxed R1CS [21]). *Consider $n \in \mathbb{N}$ and security parameter λ. Consider an additively homomorphic commitment scheme* Com *over $\mathbb{F}^n$ with $O_\lambda(1)$-sized commitments. There exists a non-interactive folding scheme for committed relaxed R1CS with respect to* Com *in the random oracle model. For an n-sized committed relaxed R1CS instances over a finite field $\mathbb{F}$, the prover's time complexity is $O_\lambda(n)$, the verifier's time complexity is $O_\lambda(1)$, and the communication complexity is $O_\lambda(1)$.*

Assumption 1 (Random oracle instantiation [21]). *Consider $n \in \mathbb{N}$ and security parameter λ. Consider an additively homomorphic commitment scheme* Com *over $\mathbb{F}^n$ with $O_\lambda(1)$-sized commitments. There exists a non-interactive folding scheme for committed relaxed R1CS with respect to* Com *in the plain model. For n-sized committed relaxed R1CS instances over a finite field $\mathbb{F}$, the prover's time complexity is $O_\lambda(n)$, the verifier's time complexity is $O_\lambda(1)$, and the communication complexity is $O_\lambda(1)$.*

Justification. We apply the Fiat-Shamir transformation [13]. In particular, we heuristically instantiate the random oracle in Lemma 1 with a concrete cryptographic hash function. $\square$

3 Non-uniform incrementally verifiable computation

This section introduces *non-uniform IVC (NIVC)*, a generalization of IVC [33], where at each step of an incremental computation, the prover proves the satisfiability of a relation chosen from a set of possible relations (the choice of which relation to use is made by an additional designated relation), whereas in the standard IVC, there is only one possible relation. As a result of this generalization, the overall relation proven by non-uniform IVC can be a non-uniform circuit, which motivates its name.

As detailed in the introduction, non-uniform IVC implies proofs of program executions on machines with a pre-defined custom instruction set. In the next section, we construct SuperNova, an efficient NIVC scheme.

Overview. To formally define NIVC, we extend standard definitions of IVC. In particular, we extend the definitions provided by Kothapalli et al. [21] in a recent work. Below, we review the definition of IVC (Definition 1) before introducing NIVC.

In the standard IVC, for some polynomial-time function F, the prover takes as input a claim/statement (i, z_0, z) and a corresponding proof $\mathit{\Pi}_i$ that proves the knowledge of witnesses $(\omega_0, \ldots, \omega_{i-1})$ such that by computing

$$z_{j+1} \leftarrow F(z_j, \omega_j)$$

for all $j \in \{0, \ldots, i-1\}$ we have that $z = z_i$. To incrementally update a proof, the prover additionally takes as input a new witness ω_i and computes a new proof $\mathit{\Pi}_{i+1}$ which attests to the statement $(i+1, z_0, z_{i+1})$ for $z_{i+1} = F(z_i, \omega_i)$. A key requirement is that proofs are succinct, that is, they do not grow in size with each incremental update.

Informally, completeness holds if given an accepting proof $\mathit{\Pi}_i$ for a statement (i, z_0, z_i) and a witness ω_i such that $z_{i+1} = F(z_i, \omega_i)$, the prover is guaranteed to produce an accepting proof $\mathit{\Pi}_{i+1}$ for statement $(i+1, z_0, z_{i+1})$. Similarly, knowledge soundness holds if for any malicious prover $\mathcal{P}^*$ that is able to produce an accepting proof $\mathit{\Pi}_i$ for statement (i, z_0, z_i), there exists a corresponding extractor $\mathcal{E}$ that can produce the corresponding witnesses $(\omega_0, \ldots, \omega_{i+1})$.

In the setting of NIVC, we extend IVC to handle a number of arbitrary polynomial-time functions $(F_1, \ldots, F_\ell)$. The choice of which function F_j for $j \in [\ell]$ is executed at a particular step in the incremental computation is handled by an additional polynomial-time function φ. In more detail, NIVC captures an incremental proof system for the following augmented statement: There exists $(\omega_0, \ldots, \omega_{i-1})$ such that on initial input z_0 and claimed output z, by computing

$$z_{j+1} \leftarrow F_{\varphi(z_j, \omega_j)}(z_j, \omega_j)$$

for all $j \in \{0, \ldots, i-1\}$, we have that $z = z_i$.

We adapt the above succinctness, completeness and knowledge soundness definitions of IVC for the setting of NIVC. Moreover, for NIVC to be a meaningful notion, we stipulate an additional efficiency requirement: the prover's work at each step scales only with the size of the function executed at that step. Without such a requirement, IVC immediately implies NIVC with the use of a single universal circuit that embeds all functions $(F_1, \ldots, F_\ell)$.

Below, we formally define NIVC in the common reference string (CRS) with preprocessing model. We consider an adaptive adversary that can pick functions $(F_1, \ldots, F_\ell)$ and φ as well as the statement after seeing the CRS.

Definition 10 (Non-uniform IVC). *A non-uniform incrementally verifiable computation (NIVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and a 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.

- $\mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))) \rightarrow (\mathsf{pk}, \mathsf{vk})$*: on input public parameters* pp*, a control function* φ*, and functions* $F_1, \ldots, F_\ell$ *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*, initial input* z_0*, claimed output after* i *applications* z_i*, a non-deterministic advice* ω_i*, and an NIVC proof* Π_i *attesting to* z_i*, produces a new proof* Π_{i+1} *attesting to* $z_{i+1} = F_{\varphi(z_i,\omega_i)}(z_i, \omega_i)$.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \rightarrow \{0, 1\}$*: on input a verifier key* vk*, a counter* i*, an initial input* z_0*, a claimed output after* i *applications* z_i*, and an NIVC proof* Π_i *attesting to* z_i*, outputs* 1 *if* Π_i *is accepting,* 0 *otherwise.*

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

(i) Perfect completeness: for any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, (F_1, \ldots, F_\ell), (i, z_0, z_i, \omega_i, \Pi_i)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))), \\ \mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1, \\ z_{i+1} \leftarrow F_{\varphi(z_i,\omega_i)}(z_i, \omega_i), \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \end{array}\right] = 1$$

where $\ell \geq 1$ *and* φ *produces an element in* $\mathbb{Z}^*_{\ell+1}$*. Moreover,* φ *and each* F_j *for* $j \in \{1, \ldots, \ell\}$ *are a polynomial-time computable function.*

(ii) Knowledge soundness: For any constant $n \in \mathbb{N}$*, and expected polynomial time adversaries* $\mathcal{P}^*$ *there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that*

$$\Pr\left[\begin{array}{l} z_n \neq z, \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\varphi, (F_1, \ldots, F_\ell), (z_0, z, \Pi)) \leftarrow \mathcal{P}^*(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))), \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}), \\ z_{i+1} \leftarrow F_{\varphi(z_i,\omega_i)}(z_i, \omega_i) \quad \forall i \in \{0, \ldots, n-1\} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

(iii) Succinctness: The size of an NIVC proof Π *is independent of the number of iterations* n.

(iv) Efficiency: The prover's space and time complexity at any step i *is linear in the size of the function applied at step* i*, i.e.,* $O_\lambda(|F_{\varphi(z_{i-1},\omega_{i-1})}|)$.

Remark 2 (NIVC implies IVC). Observe that if one restricts the above definition to the setting where $\ell = 1$ and φ outputs 1, one recovers the definition of IVC (Section 2.1). This means that any NIVC scheme is also an IVC scheme.

4 SuperNova: Non-uniform IVC from folding schemes

This section describes SuperNova, a non-uniform IVC scheme designed from a non-interactive folding scheme. When instantiated with any additively-homomorphic commitment scheme with succinct commitments, the scheme achieves the claimed efficiency (Lemma 4). In addition, like Nova [21], SuperNova can incorporate a zkSNARK (e.g., Spartan [29]) to prove the knowledge of valid NIVC proofs, providing a succinct, zero-knowledge proof of knowledge of a valid NIVC proof.

Below, we first provide an overview of SuperNova's NIVC scheme. We then describe SuperNova's NIVC scheme formally and optimizations. We defer formal details of zkSNARK of valid NIVC proofs to near-term work.

4.1 Overview of SuperNova

We intentionally overlook certain minor complications. We then address these complications before providing a formal construction.

Consider polynomial-time computable functions $F_1, \ldots, F_\ell$ and φ. Recall that the NIVC statement (i, z_0, z_i) claims the knowledge of $(\omega_0, \ldots, \omega_{i-1})$ such that by computing for all $k \in \{0, \ldots, i-1\}$

$$z'_{k+1} \leftarrow F(z'_k, \omega_k)$$

for $z'_0 = z_0$ we have that $z'_i = z_i$.

We now describe a single iterative step of the prover's work. That is, we explain how the prover can take a proof Π_i for the NIVC statement (i, z_0, z_i) and efficiently produce an updated proof Π_{i+1} for the NIVC statement $(i+1, z_0, z_{i+1})$.

At a high level, instead of directly proving the knowledge of satisfying witness to some prescribed F_j for $j \in \{1, \ldots, \ell\}$ in each step, the prover proves the knowledge of a satisfying witness to an augmented function F'_j. The augmented function F'_j, in addition to running F_j, performs additional bookkeeping using a folding scheme to help verifiably update the NIVC proof.

At first glance, a straw-man approach is to have each F'_j take as input a relaxed R1CS instance that claims the correct execution of all prior iterations and then fold that instance into a running instance using the folding scheme as stated in Lemma 1 (this is the approach taken by Nova [21]). However, the folding scheme for relaxed R1CS requires that both instances have the same structure (i.e., represent the same computation). In the case of standard IVC, as there is only one function that can be applied at each iterative step, this holds naturally. However, this is not the case for non-uniform IVC.

To address this, F'_j instead takes a list U_i of running instances, where $\mathsf{U}_i[j]$ attests to all prior iterations of F'_j up to $i-1$ steps. As such, checking all of U_i is equivalent to checking $i-1$ steps. In addition, F'_j takes as input a new instance u_i, which claims the correctness of the i'th step. Instead of directly checking this instance (which would be concretely expensive), F'_j folds u_i into the appropriate instance in U_i according to φ to produce a new list of running instances U_{i+1}. To claim the correctness of F'_j itself, the prover produces a new instance u_{i+1}.

We let the NIVC proof Π_i contain the list U_i, the fresh instance u_i, and the corresponding witnesses. Thus, the prover can use parts of Π_i as input to the appropriate function F'_j to produce U_{i+1} and u_{i+1}, and separately compute the corresponding witnesses. These terms together define Π_{i+1}. At the end of the iterative computation (or at any intermediate step, if necessary), the verifier can check i steps by checking proof Π_i directly.

We now provide additional details.

The augmented function. The function F'_j takes as non-deterministic input the statement so far (i, z_0, z_i), the auxiliary witness ω_i, the index of the previously executed function pc_i, a relaxed R1CS instance that claims that the step i was executed correctly u_i, and a list of ℓ running relaxed R1CS instances U_i, where for $j \in \{1, \ldots, \ell\}$, $\mathsf{U}_i[j]$ is a relaxed R1CS instance attesting to all prior executions of of F'_j. Function F'_j first runs F_j on input (z_i, ω_i) to compute z_{i+1}. As additional bookkeeping, F'_j runs a verifier circuit that does the following.

1. Checks that U_i and pc_i are contained in the public output of the instance u_i. This enforces that U_i and pc_i are indeed produced by the prior step.
2. Runs the non-interactive folding scheme's verifier to fold an instance that claims the correct execution of the previous step, u_i, into $\mathsf{U}_i[\mathsf{pc}_i]$ to produce an updated list of running instances U_{i+1}. This ensures that checking U_{i+1} implies checking U_i and u_i while maintaining that U_{i+1} does not grow in size with respect to U_i.
3. Invokes the function φ on input (z_i, ω_i) to compute pc_{i+1}, which represents the index of the function F_j currently being run. pc_{i+1} is then sent to the next invocation of an augmented circuit (which contains a verifier circuit).

F'_j produces as public output the new statement $(i + 1, z_0, z_{i+1})$, the updated list of running instances U_{i+1}, and the updated index pc_{i+1}.

Structure of a SuperNova proof. We now discuss the structure of an NIVC proof and how it can be checked. Consider an NIVC statement (i, z_0, z_i). Let the corresponding NIVC proof be Π_i, which consists of a vector of ℓ instances U_i, the corresponding vector of ℓ witnesses W_i, an instance that claims the correct execution of the latest iteration u_i, the corresponding witness w_i, and pc_i.

Suppose, we have the following: So long as $(\mathsf{u}_i, \mathsf{w}_i)$ is a satisfying instance-witness pair with respect to augmented function F'_{pc_i} and contains U_i and pc_i in the public output we have that checking all instances in U_i implies checking all prior iterations and correct sequencing. So, the verifier can check the NIVC statement (i, z_0, z_i) by checking the following: $(\mathsf{u}_i, \mathsf{w}_i)$ is a satisfying instance-witness pair with respect to function F'_{pc_i}, the public IO of u_i contains U_i and pc_i, and for each $j \in \{1, \ldots, \ell\}$ check that $(\mathsf{U}_i[j], \mathsf{W}_i[j])$ is a satisfying instance-witness pair with respect to function F'_j.

Updating a SuperNova proof. Given a proof Π_i of i steps, the prover can efficiently produce a proof Π_{i+1} of $i + 1$ steps. The core invariant we maintain is as follows: If checking Π_i indeed attests to i steps we must have that Π_{i+1} attests to $i + 1$ steps while maintaining that Π_{i+1} does not grow in size. Indeed, assume that checking $\Pi_i = (\mathsf{U}_i, \mathsf{W}_i, \mathsf{u}_i, \mathsf{w}_i, \mathsf{pc}_i)$ is sufficient to verify the NIVC claim (i, z_0, z_i). Suppose the prover is provided as input proof Π_i, a claim (i, z_0, z_i), and an auxiliary witness ω_i.

The prover proceeds as follows: Using the non-interactive folding scheme, the prover first folds the instance-witness pair $(\mathsf{u}_i, \mathsf{w}_i)$, which attests to the correctness of the last step into index pc_i of U_i and W_i respectively. Let U_{i+1} and W_{i+1} denote the updated list of running instances and witnesses respectively. Now, by assumption, so long as u_i and contains U_i and pc_i, we have that that checking $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ is equivalent to checking Π_i while maintaining that $|(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})| = |(\mathsf{U}_i, \mathsf{W}_i)|$. To account for the next step of execution, the prover first computes the updated index $\mathsf{pc}_{i+1} \leftarrow \varphi(z_i, \omega_i)$. The prover then computes

$$((i+1, z_0, z_{i+1}), \mathsf{U}_{i+1}, \mathsf{pc}_{i+1}) \leftarrow F'_{\mathsf{pc}_{i+1}}(\mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i)$$

and computes the corresponding claim of correct execution u_{i+1} and witness w_{i+1}. Now we have that checking u_{i+1} attests to the following.

1. $F_{\mathsf{pc}_{i+1}}$ produces z_{i+1} on input (z_i, ω_i).
2. The public IO of u_i contains U_i and pc_i, and therefore U_i indeed attests to i steps so long as u_i is valid and u_i is designated to be folded into $\mathsf{U}_i[\mathsf{pc}_i]$.
3. U_{i+1} was computed by folding u_i into $\mathsf{U}_i[\mathsf{pc}_i]$ and therefore checking U_{i+1} is equivalent to checking Π_i.
4. pc_{i+1} was computed correctly and therefore u_{i+1} is indeed designated to be checked against $F'_{\mathsf{pc}_{i+1}}$

Therefore, so long as u_{i+1} is valid, we have that checking U_{i+1} attests to i steps. Moreover, because u_{i+1} attests to the correctness of the latest step, checking u_{i+1} against $F'_{\mathsf{pc}_{i+1}}$ is sufficient to attest to $i+1$ iterations. This means that checking $\Pi_{i+1} = (\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1}, \mathsf{pc}_{i+1})$ is sufficient to check the NIVC statement $(i+1, z_0, z_{i+1})$.

Fixing minor complications. The prior description overlooks the following minor issues. Prior work [21] addresses these, and we now provide an overview of these in light of the above overview.

First, we describe how to update a proof Π_i to produce a proof Π_{i+1}. However, we did not define a base case proof Π_0 and how the prover, verifier, and each function F'_j handles the base case. At a high level, we have F'_j produce trivial running instances in the base case.

Second, the non-interactive folding scheme's verifier run by F'_j needs additional advice generated by the non-interactive folding scheme's prover. To address this, the prover provides additional non-deterministic input to F'_j.

Finally, there is a subtle sizing issue in the above description: in each step, because U_{i+1} is produced as the public IO of $F'_{\mathsf{pc}_{i+1}}$, it must be contained in the public IO of instance u_{i+1}. In the next iteration, because u_{i+1} is folded into $\mathsf{U}_{i+1}[\mathsf{pc}_{i+1}]$, this means that $\mathsf{U}_{i+1}[\mathsf{pc}_{i+1}]$ is at least as large as U_i by the properties of the folding scheme. This means that the list of running instances grows in each step. To alleviate this issue, we have each F'_j only produce a hash of its outputs as public output. In the subsequent step, the next augmented function takes as non-deterministic input a preimage to this hash.

4.2 Core construction

We formally describe our construction below and then prove that it meets the requirements of an NIVC scheme.

Construction 1 (Non-uniform IVC (NIVC)). Let $\mathsf{NIFS} = (\mathsf{G}, \mathsf{K}, \mathsf{P}, \mathsf{V})$ be the non-interactive folding scheme for committed relaxed R1CS implied by Assumption 1. Let hash denote a cryptographic hash function.

Let φ denote a polynomial time function, and let $(F_1, \dots F_\ell)$ denote a collection of ℓ polynomial time functions. Without loss of generality, assume that these functions have identical input/output sizes and circuit sizes. We define a collection of augmented functions as follows (all arguments to an augmented function are provided as non-deterministic advice). There are ℓ augmented functions, where F'_j is hardwired with F_j, $j \in [\ell]$. Furthermore, F'_j can be computed in polynomial time because F_j, φ, and the additional bookkeeping in F'_j can be computed in polynomial time. So, each F'_j can be represented with committed relaxed R1CS and let $\mathsf{S}(F'_j)$ denote its structure. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ denote a trivial instance-witness pair for any of these committed relaxed R1CS structures.

$\underline{F'_j(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \overline{T}) \to \mathsf{x}}$:

```
(1) Compute the next program counter $\mathsf{pc}_{i+1} \in \mathbb{Z}^*_{\ell+1} \leftarrow \varphi(z_i, \omega_i)$.
(2) Compute the next output $z_{i+1} \leftarrow F_j(z_i, \omega_i)$.
(3) If $i$ is 0:
    (a) Instantiate the running instance $\mathsf{U}_{i+1} \leftarrow [\mathsf{u}_\perp]^\ell$.
    (b) Check that $z_0 = z_i$ to ensure the statement holds in the base case.
    Otherwise:
    (a) Check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathsf{pc}_i, z_0, z_i, \mathsf{U}_i)$ where $\mathsf{u}_i.\mathsf{x}$ is $\mathsf{u}_i$'s public IO.
    (b) Check that $1 \le \mathsf{pc}_i \le \ell$.
    (c) Check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$ to ensure that $\mathsf{u}_i$ is an R1CS instance.
    (d) Set $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i$ and update $\mathsf{U}_{i+1}[\mathsf{pc}_i] \leftarrow \mathsf{NIFS.V}(\mathsf{vk}[\mathsf{pc}_i], \mathsf{U}_i[\mathsf{pc}_i], \mathsf{u}_i, \overline{T})$.
(4) Output $\mathsf{x} \leftarrow \mathsf{hash}(\mathsf{vk}, i+1, \mathsf{pc}_{i+1}, z_0, z_{i+1}, \mathsf{U}_{i+1})$.
```

Let $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F'_j, (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \overline{T}))$ denote a committed relaxed R1CS instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ (such that $\mathsf{u}_{i+1}.u = 1$, $\mathsf{w}_{i+1}.E = \mathbf{0}$, and $\mathsf{w}_{i+1}.r_E = 0$) for the execution of F'_j on non-deterministic advice $(\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \overline{T})$. Note that the committed relaxed R1CS structure for this instance-witness pair is $\mathsf{S}(F'_j)$ where $j = \varphi(z_i, \omega_i)$.

We construct a non-uniform IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

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

$\underline{\mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \dots, F_\ell))) \to (\mathsf{pk}, \mathsf{vk})}$:

```
(1) Compute $(\mathsf{pk}_{\mathsf{fs}_i}, \mathsf{vk}_{\mathsf{fs}_i}) \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, \mathsf{S}(F'_i))$ for all $i \in [\ell]$.
(2) Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow (((F_1, \mathsf{pk}_{\mathsf{fs}_1}), \dots, (F_\ell, \mathsf{pk}_{\mathsf{fs}_\ell})), ((F_1, \mathsf{vk}_{\mathsf{fs}_1}), \dots, (F_\ell, \mathsf{vk}_{\mathsf{fs}_\ell})))$.
```

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

```
(1) Parse $\Pi_i$ as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
(2) if $i$ is 0:
      (a) Compute $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \overline{T}) \leftarrow ([\mathsf{u}_\perp]^\ell, [\mathsf{w}_\perp]^\ell, \mathsf{u}_\perp.\overline{E})$.
    Otherwise:
      (a) Set $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i, \mathsf{W}_{i+1} \leftarrow \mathsf{W}_i$.
      (b) Update $(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i], \overline{T}) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}[\mathsf{pc}_i], (\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i]), (\mathsf{u}_i, \mathsf{w}_i))$.
(3) Compute $\mathsf{pc}_{i+1} \in \mathbb{Z}^*_{\ell+1} \leftarrow \varphi(z_i, \omega_i)$.
(4) Compute $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{trace}(F'_{\mathsf{pc}_{i+1}}, (\mathsf{vk}, \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \overline{T}))$.
(5) Output $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \mathsf{pc}_{i+1})$.
```

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

```
If $i$ is 0:
(a) Check that $z_i = z_0$.
Otherwise:
(a) Parse $\Pi_i$ as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
(b) Check that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathsf{pc}_i, z_0, z_i, \mathsf{U}_i)$.
(c) Check that $1 \leq \mathsf{pc}_i \leq \ell$.
(d) Check that $(\mathsf{u}_i.\overline{E}, \mathsf{u}.u) = (\mathsf{u}_\perp.\overline{E}, 1)$.
(e) $\forall j \in [\ell]$, check that $(\mathsf{U}_i[j], \mathsf{W}_i[j])$ are satisfying instance-witness pairs with
    structure $\mathsf{S}(F'_j)$.
(f) check that $\mathsf{w}_i$ is a satisfying witness to instance $\mathsf{u}_i$ with structure $\mathsf{S}(F'_{\mathsf{pc}_i})$.
```

4.3 Proofs of SuperNova's properties

Lemma 2 (Completeness). *Construction 1 is an NIVC scheme that satisfies perfect completeness.*

Proof. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ denote the public parameters. Consider arbitrary polynomial-time functions $(\varphi, (F_1, \ldots, F_\ell))$, and let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, \{F_1, \ldots, F_\ell\}))$. Consider an arbitrary statement (i, z_0, z_i), a witness ω_i, and a proof Π_i such that

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

Let $\mathsf{pc}_{i+1} \leftarrow \varphi(z_i, \omega_i)$ and $z_{i+1} \leftarrow F_{\mathsf{pc}_{i+1}}(z_i, \omega_i)$, and let

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

We must show that

$$\mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 \tag{2}$$

with probability 1.

We show this by considering the case when $i = 0$ and the case when $i > 0$.

Suppose that $i = 0$ (we refer to this case as the "base case" for ease of reference). First, we show that the prover *can* compute a proof $\mathit{\Pi}_{i+1}$. By Equation (1), we have that $\mathcal{V}$ accepts $\mathit{\Pi}_i$ on input (i, z_0, z_i). Because $i = 0$, by the base case check of $\mathcal{V}$, we have that $z_0 = z_i$. Thus, we have that the base case check of $F'_{\mathsf{pc}_{i+1}}$ passes on input $(\mathsf{vk}, \_, \_, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \_)$, where $\mathsf{pc}_{i+1} = \varphi(z_i, \omega_i)$, and $\_$ denotes an arbitrary argument. This ensures that $\mathcal{P}$ *can* compute satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ with respect to committed relaxed R1CS structure $\mathsf{S}(F'_{\mathsf{pc}_{i+1}})$. Then, by the base case specification of $\mathcal{P}$, we have:

$$\mathit{\Pi}_{i+1} = ((\mathsf{u}_\perp, \mathsf{w}_\perp)^\ell, (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \mathsf{pc}_{i+1}).$$

Next, we show that that the verifier accepts $\mathit{\Pi}_{i+1}$ by demonstrating that all of its checks pass. First, by the construction of $\mathcal{P}$ and $F'_{\mathsf{pc}_{i+1}}$, we have that

$$\mathsf{u}_{i+1}.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i+1, \mathsf{pc}_{i+1}, z_0, F_{\mathsf{pc}_{i+1}}(z_i, \omega_i), [\mathsf{u}_\perp]^\ell).$$

Second, by the definition of φ, we have that $1 \le \mathsf{pc}_{i+1} \le \ell$. Third, by the construction of $\mathcal{P}$ (specifically, the definition of trace), we have $\mathsf{w}_{i+1}.E = \mathbf{0}$ and $\mathsf{w}_{i+1}.r_E = 0$ (implying that $\mathsf{u}_{i+1}.\overline{E} = 0$), and $\mathsf{u}_{i+1}.u = 1$. Fourth, by definition, the k'th instance-witness pair in $(\mathsf{u}_\perp, \mathsf{w}_\perp)^\ell$ satisfies F'_k, for all $k \in \{1, \ldots, \ell\}$. Finally, we have that $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ is satisfying with respect to $\mathsf{S}(F'_{\mathsf{pc}_{i+1}})$ from above. Therefore, we have that Equation (2) holds.

Now, suppose $i > 0$. Let $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$ be the result of parsing $\mathit{\Pi}_i$. First, we show that the prover can compute a proof $\mathit{\Pi}_{i+1}$. Because the verifier accepts $\mathit{\Pi}_i$ (Equation 1), we have that $\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathsf{pc}_i, z_0, z_i)$ and $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\mathsf{u}_\perp.\overline{E}, 1)$. As these are precisely the checks performed by $F'_{\mathsf{pc}_{i+1}}$ for $\mathsf{pc}_{i+1} = \varphi(z_i, \omega_i)$, we have that $\mathcal{P}$ can compute satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ with respect to committed relaxed R1CS structure $\mathsf{S}(F'_{\mathsf{pc}_{i+1}})$. Then, by the construction of $\mathcal{P}$, we have

$$\mathit{\Pi}_{i+1} = ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \mathsf{pc}_{i+1})$$

for some $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$.

Next, we show that the verifier accepts $\mathit{\Pi}_{i+1}$. First, by the construction of $\mathcal{P}$ and $F'_{\mathsf{pc}_{i+1}}$ we have that

$$\mathsf{u}_{i+1}.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i+1, \mathsf{pc}_{i+1}, z_0, z_{i+1}, \mathsf{U}_{i+1}) \tag{3}$$

Second, by definition of φ we have that $1 \le \mathsf{pc}_{i+1} \le \ell$. Third, as in the base case, by the construction of $\mathcal{P}$, we have $\mathsf{u}_{i+1}.\overline{E} = 0$ and $\mathsf{u}_{i+1}.u = 1$. Fourth, by the construction of $\mathcal{P}$, we have for all $k \in [\ell]$ (for notational convenience ignoring that $\mathsf{NIFS.P}$ additionally outputs a cross-term):

$$(\mathsf{U}_{i+1}[k], \mathsf{W}_{i+1}[k]) = \begin{cases} \mathsf{NIFS.P}(\mathsf{pk}[k], (\mathsf{U}_i[k], \mathsf{W}_i[k]), (\mathsf{u}_i, \mathsf{w}_i)) & \text{if } k = \mathsf{pc}_i \\ (\mathsf{U}_i[k], \mathsf{W}_i[k]) & \text{otherwise} \end{cases}$$

Then, by the premise that the verifier accepts Π_i, we already have that $(\mathsf{U}_{i+1}[k], \mathsf{W}_{i+1}[k])$ is a satisfying instance-witness pair for structure $\mathsf{S}(F_k)$ for all $k \in \{1, \ldots, \mathsf{pc}_i - 1, \mathsf{pc}_i + 1, \ldots, \ell\}$. Moreover, by the premise, $(\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i])$ and $(\mathsf{u}_i, \mathsf{w}_i)$ are satisfying instance-witness pairs for committed relaxed R1CS structure $\mathsf{S}(F_{\mathsf{pc}_i})$. Thus, by the completeness of the underlying non-interactive folding scheme, we have that $(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i])$ is also a satisfying instance-witness pair for structure $\mathsf{S}(F_{\mathsf{pc}_i})$. Fourth, we have that $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ is satisfying with respect to $\mathsf{S}(F'_{\mathsf{pc}_{i+1}})$ from above. Therefore, we have that Equation (2) holds. □

Lemma 3 (Knowledge soundness). *Construction 1 is an NIVC scheme that satisfies knowledge soundness.*

Proof. Let n be a global constant. Consider an expected polynomial-time adversary $\mathcal{P}^*$. Suppose that $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$. Suppose that on input pp, $\mathcal{P}^*$ produces, for some $\ell \geq 1$, a polynomial-time control function φ that maps into $\mathbb{Z}^*_{\ell+1}$, polynomial-time functions $(F_1, \ldots, F_\ell)$, an input z_0, an output z, and an NIVC proof Π. Suppose that on input $(\mathsf{pp}, (\varphi, \{F_1, \ldots, F_\ell\}))$, $\mathcal{K}$ produces prover key pk, and verifier key vk.

Suppose that

$$\mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1 \tag{4}$$

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

$$\begin{aligned} \mathsf{pc}_{i+1} &\leftarrow \varphi(z_i, \omega_i) \\ z_{i+1} &\leftarrow F_{\mathsf{pc}_{i+1}}(z_i, \omega_i) \end{aligned}$$

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

We show inductively that we can construct an expected polynomial-time extractor $\mathcal{E}_i$ that on input pp outputs $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i)$ such for all $j \in \{i, \ldots, n-1\}$, given $\mathsf{pc}_{j+1} \leftarrow \varphi(z_j, \omega_j)$, we have that

$$z_{j+1} = F_{\mathsf{pc}_{j+1}}(z_j, \omega_j)$$

and $z_n = z$, and that

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

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 $\mathcal{E}_0$ are such that computing $\mathsf{pc}_{i+1} \leftarrow \varphi(z_i, \omega_i)$ and $z_{i+1} \leftarrow F_{\mathsf{pc}_{i+1}}(z_i, \omega_i)$ for all $i \geq 0$ gives $z_n = z$. Thus, by setting $\mathcal{E} = \mathcal{E}_0$ we are done.

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 $\mathcal{E}_i$ to construct an adversary for the non-interactive folding scheme, which we denote as $\widetilde{\mathcal{P}}_{i-1}$. This in turn guarantees a corresponding extractor $\widetilde{\mathcal{E}}_{i-1}$ by the knowledge soundness of the non-interactive folding scheme. 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 $\mathcal{E}_n(\mathsf{pp})$ output $(\perp, \perp, \Pi_n)$ where $\perp$ represents the empty list and Π_n is the output of $\mathcal{P}^*(\mathsf{pp})$. By the precondition (Equation (4)), $\mathcal{E}_n$ succeeds with probability ϵ in expected polynomial-time.

For $i \geq 1$, suppose that we 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 $\mathcal{E}_{i-1}$, we first construct an adversary $\widetilde{\mathcal{P}}_{i-1}$ for the non-interactive folding scheme that outputs an adversarially chosen structure and instances to be folded followed by a folded instance-witness pair and a folding proof:

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

(1) Let $((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i) \leftarrow \mathcal{E}_i(\mathsf{pp})$.
(2) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$.
(3) Parse w_i to retrieve $\mathsf{U}_{i-1}, \mathsf{u}_{i-1}, \overline{T}_{i-1}, \mathsf{pc}_{i-1}$.
(4) Output $(\mathsf{S}(F'_{\mathsf{pc}_{i-1}}), (\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{u}_{i-1}))$ and $((\mathsf{U}_i[\mathsf{pc}_{i-1}], \mathsf{W}_i[\mathsf{pc}_{i-1}]), \overline{T}_{i-1})$.

We now analyze the success probability of $\widetilde{\mathcal{P}}_{i-1}$. Suppose that $\widetilde{\mathcal{P}}_{i-1}$ retrieves proof $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$ as a result of running $\mathcal{E}_i$ internally and retrieves $(\mathsf{U}_{i-1}, \mathsf{u}_{i-1}, \overline{T}_{i-1}, \mathsf{pc}_{i-1})$ from parsing w_i. By the inductive hypothesis, we have that $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. This implies that $1 \leq \mathsf{pc}_i \leq \ell$ and that $(\mathsf{u}_i, \mathsf{w}_i)$ is a satisfying instance-witness pair for committed relaxed R1CS structure $\mathsf{S}(F'_{\mathsf{pc}_i})$. Because $\mathcal{V}$ ensures that $(\mathsf{u}_i.\overline{E}, \mathsf{u}_i.u) = (\overline{0}, 1)$ we have that w_i is indeed a satisfying assignment for F'_{pc_i} (and not just a trivially satisfying witness). Therefore, because pc_{i-1} was parsed from w_i, by the construction of $\mathsf{S}(F'_{\mathsf{pc}_i})$, this implies that $1 \leq \mathsf{pc}_{i-1} \leq \ell$. Then, by the the verifier's checks on U_i and W_i, we have that $(\mathsf{U}_i[\mathsf{pc}_{i-1}], \mathsf{W}_i[\mathsf{pc}_{i-1}])$ is a satisfying instance-witness pair for committed relaxed R1CS structure $\mathsf{S}(F'_{\mathsf{pc}_{i-1}})$. By the verifier's checks, we additionally have that

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

Then, by the construction of F'_{pc_i} and the binding property of the hash function, we have that

$$\mathsf{U}_i[\mathsf{pc}_{i-1}] = \mathsf{NIFS.V}(\mathsf{vk}[\mathsf{pc}_{i-1}], \mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{u}_{i-1}, \overline{T}_{i-1})$$

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

Then, by the knowledge soundness of the underlying non-interactive folding scheme, there exists an extractor $\widetilde{\mathcal{E}}_{i-1}$ that outputs satisfying witnesses for instances $\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}]$ and u_{i-1} with respect to $\mathsf{S}(F'_{\mathsf{pc}_{i-1}})$ with probability $\epsilon - \mathsf{negl}(\lambda)$ in expected polynomial-time.

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

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

(1) Run $\widetilde{\mathcal{P}}_{i-1}$ to retrieve the output $(\mathsf{u}'_{i-1}, \mathsf{u}_{i-1})$ and retrieve

$$((z_i, \ldots, z_{n-1}), (\omega_i, \ldots, \omega_{n-1}), \Pi_i)$$

from its internal state.

(2) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$ and parse w_i to retrieve z_{i-1}, ω_{i-1}, and pc_{i-1}.

(3) Let $(\mathsf{w}'_{i-1}, \mathsf{w}_{i-1}) \leftarrow \widetilde{\mathcal{E}}_{i-1}(\mathsf{pp})$.

(4) Compute $(\mathsf{U}_{i-1}, \mathsf{W}_{i-1}) \leftarrow (\mathsf{U}_i, \mathsf{W}_i)$ and update

$$(\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{W}_{i-1}[\mathsf{pc}_{i-1}]) \leftarrow (\mathsf{u}'_{i-1}, \mathsf{w}'_{i-1})$$

(5) Let $\Pi_{i-1} \leftarrow ((\mathsf{U}_{i-1}, \mathsf{W}_{i-1}), (\mathsf{u}_{i-1}, \mathsf{w}_{i-1}), \mathsf{pc}_{i-1})$.

(6) Output $((z_{i-1}, \ldots, z_{n-1}), (\omega_{i-1}, \ldots, \omega_{n-1}), \Pi_{i-1})$.

We now analyze the success probability of $\mathcal{E}_{i-1}$. Suppose that $\mathcal{E}_{i-1}$ retrieves $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$ from the internal state of $\widetilde{P}_{i-1}$ and outputs

$$((z_{i-1}, \ldots, z_{n-1}), (\omega_{i-1}, \ldots, \omega_{n-1}), \Pi_{i-1}).$$

Suppose that Π_{i-1} is parsed as $((\mathsf{U}_{i-1}, \mathsf{W}_{i-1}), (\mathsf{u}_{i-1}, \mathsf{w}_{i-1}), \mathsf{pc}_{i-1})$. 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 $j \in \{i, \ldots, n-1\}$, given $\mathsf{pc}_{j+1} \leftarrow \varphi(z_j, \omega_j)$,

$$z_{j+1} = F_{\mathsf{pc}_{j+1}}(z_j, \omega_j),$$

and $z_n = z$, and that $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$. By the verifier's checks, we have that $1 \leq \mathsf{pc}_i \leq \ell$, that $(\mathsf{u}_i, \mathsf{w}_i)$ is a non-trivial satisfying instance witness pair for F'_{pc_i}, and that

$$\mathsf{u}_i.\mathsf{x} = \mathsf{hash}(\mathsf{vk}, i, \mathsf{pc}_i, z_0, z_i, \mathsf{U}_i). \tag{5}$$

Then, because z_{i-1} and ω_{i-1} were parsed from w_i, by the construction of F'_{pc_i} and the binding property of the hash function, we have

$$\begin{aligned} \mathsf{pc}_i &= \varphi(z_{i-1}, \omega_{i-1}) \\ z_i &= F_{\mathsf{pc}_i}(z_{i-1}, \omega_{i-1}) \end{aligned}$$

Therefore we have that $(z_{i-1}, \ldots, z_{n-1})$ and $(\omega_{i-1}, \ldots, \omega_{n-1})$ satisfy the inductive hypothesis with probability $\epsilon - \mathsf{negl}(\lambda)$.

Next, we argue that Π_{i-1} is valid. Because $(\mathsf{u}_i, \mathsf{w}_i)$ satisfies F'_{pc_i}, 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 (5), we have that

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

$$(\mathsf{u}_{i-1}.\overline{E}, \mathsf{u}_{i-1}.u) = (\overline{0}, 1)$$

By the inductive hypothesis that Π_i is accepting, by construction, we have that $(\mathsf{U}_{i-1}[k], \mathsf{W}_{i-1}[k])$ is a satisfying instance-witness pair for all $k \in \{1, \ldots, \mathsf{pc}_{i-1} - 1, \mathsf{pc}_{i-1}+1, \ldots, \ell\}$. Moreover, by the construction of F'_{pc_i} we have that $1 \leq \mathsf{pc}_{i-1} \leq \ell$. Thus, we have that $(\mathsf{U}_{i-1}[\mathsf{pc}_{i-1}], \mathsf{W}_{i-1}[\mathsf{pc}_{i-1}])$ and $(\mathsf{u}_{i-1}, \mathsf{w}_{i-1})$ are accepting instance-witness pairs with respect to structure $\mathsf{S}(F'_{\mathsf{pc}_{i-1}})$ with probability $\epsilon - \mathsf{negl}(\lambda)$ due to the success probability of $\widetilde{\mathcal{E}}_{i-1}$. Therefore, we have that

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

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

□

Lemma 4 (Efficiency). *When instantiated with the Pedersen commitment scheme, we have that for each $j \in \{1, \ldots, \ell\}$, $|F'_j| = |\varphi| + |F_j| + o(2 \cdot \mathsf{G} + 2 \cdot \mathsf{H} + \mathsf{R})$, where $|F_j|$ and $|\varphi|$ denote the number of R1CS constraints to encode functions F_j and φ respectively, G is the number of constraints required to encode a group scalar multiplication, H is the number of constraints required to encode* hash*, and R is the number of constraints to encode the RO ρ.*

Proof. On input instances U and u, $\mathsf{NIFS.V}$ computes $\overline{E} \leftarrow \mathsf{U}.\overline{E} + r \cdot \overline{T} + r^2 \cdot \mathsf{u}.\overline{E}$ and $\overline{W} \leftarrow \mathsf{U}.\overline{W} + r \cdot \mathsf{u}.\overline{W}$. However, by construction, $\mathsf{u}.\overline{E} = \mathsf{u}_{\perp}.\overline{E} = \overline{0}$. So, $\mathsf{NIFS.V}$ computes two group scalar multiplications, as it does not need to compute $r^2 \cdot \mathsf{u}.\overline{E}$. $\mathsf{NIFS.V}$ additionally invokes the RO once to obtain a random scalar. Finally, F'_j makes two additional calls to hash and a call to φ (details are in the description of F'_j). □

4.4 Optimizations

In the construction of F'_j from the previous subsection, the cost of hash scales with ℓ since the circuit takes as non-deterministic input ℓ running instances. However, at each invocation, the circuit updates only one of them. Thus, one can employ standard memory-checking techniques [7]. More specifically, by encoding Merkle proofs in a circuit [10], one can reduce dependence of F'_j's circuit size on ℓ from $O_\lambda(\ell)$ to $O_\lambda(\log \ell)$. This can be reduced to $O_\lambda(1)$ constraints by using multiset-based offline memory checking inside a circuit [22, 30].

Acknowledgments

We thank Justin Drake, Chhi'mèd Künzang, Bryan Parno, Justin Thaler, Ioanna Tzialla, and Michael Walfish for helpful conversations and comments on a prior version of this paper. Abhiram Kothapalli was supported by a gift from Bosch, NSF Grant No. 1801369, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: Extended abstract. In: ITCS (2013)

[2] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: CRYPTO (Aug 2013)

[3] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: CRYPTO (2014)

[4] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: USENIX Security (2014)

[5] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)

[6] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: STOC (2013)

[7] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: FOCS (1991)

[8] Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018)

[9] Braun, B.: Compiling computations to constraints for verified computation. Tech. rep., UT Austin Honors thesis HR-12-10 (Dec 2012)

[10] Braun, B., Feldman, A.J., Ren, Z., Setty, S., Blumberg, A.J., Walfish, M.: Verifying computations with state. In: SOSP (2013)

[11] Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: ITCS (2012)

[12] Drake, J.: ZK Whiteboard Sessions – Module Fourteen: Nova Crash Course with Justin Drake. https://www.youtube.com/watch?v=SwonTtOQzAk

[13] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)

[14] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT (2013)

[15] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC. pp. 99–108 (2011)

[16] Goldberg, L., Papini, S., Riabzev, M.: Cairo – a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive (2021)

[17] Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)

[18] Khovratovich, D., Maller, M., Tiwari, P.R.: Minroot: Candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626 (2022)

[19] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: STOC (1992)

[20] Kosba, A., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In: USENIX Security (2020)

[21] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: CRYPTO (2022)

[22] Lee, J., Nikitin, K., Setty, S.: Replicated state machines without replicated execution. In: S&P (2020)

[23] Lurk: https://github.com/lurk-lang

[24] Micali, S.: CS proofs. In: FOCS (1994)

[25] Ozdemir, A., Wahby, R.S., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: USENIX Security (2020)

[26] Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P (May 2013)

[27] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO (1991)

[28] RISC ZERO: https://www.risczero.com/

[29] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)

[30] Setty, S., Angel, S., Gupta, T., Lee, J.: Proving the correct execution of concurrent services in zero-knowledge. In: OSDI (Oct 2018)

[31] Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: EuroSys (Apr 2013)

[32] Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: USENIX Security (Aug 2012)

[33] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC. pp. 552–576 (2008)

[34] Wahby, R.S., Setty, S., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: NDSS (2015)

[35] Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT. pp. 379–407 (2019)

[36] WhiteHat, B., Gluchowski, A., HarryR, Fu, Y., Castonguay, P.: Roll_up / roll_back snark side chain ~17000 tps. https://ethresear.ch/t/roll-up-roll-back-snark-side-chain-17000-tps/3675 (Oct 2018)

[37] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster verifiable RAM with program-independent preprocessing. In: S&P (2018)