Nova

Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli (CMU), Srinath Setty (MSR), Ioanna Tzialla (NYU)

eprint.iacr.org/2021/370
github.com/Microsoft/Nova
Goal: Practical zkSNARKs for Recursive Computation

Prove that (non-deterministic) function $F$ applied $n$ times to initial input $z_0$ results in $z_n$
Goal: Practical zkSNARKs for Recursive Computation

Prove that (non-deterministic) function $F$ applied $n$ times to initial input $z_0$ results in $z_n$

Applications

• Verifiable Delay Function: Let $F$ be a delay function [BBBF19]
• ZK Virtual Machines: Let $F$ be a step of the VM [BCTV14, GPR21]
• ZK-rollups: Let $F$ validate new blockchain transactions
Naive Approach

Use a SNARK to monolithically prove the unrolled statement

\[ z_0 \quad W_0, \ldots, W_n \]

\[
\begin{array}{cccccccc}
F & F & F & F & F & F & \cdots & F & F \\
\end{array}
\]

\[ z_n \]
Naive Approach

Use a SNARK to monolithically prove the unrolled statement

\[
\begin{array}{c}
  z_0 \\
  \vdots \\
  z_n
\end{array}
\]

\[
W_0, \ldots, W_n
\]

**Arithmetic Circuit**

\[
F \ F \ F \ F \ F \ F \ \ldots \ F \ F
\]

**Drawbacks**

- Fixes \( n \) ahead of time
- \( O(n \cdot |F|) \) prover memory and verifier preprocessing time
- Verifier time may depend on \( n \)
Approach: Incrementally Verifiable Computation (IVC) [Val08]

Incrementally update a proof of $i$ applications to a proof of $i + 1$ applications with the same size.
Approach: Incrementally Verifiable Computation (IVC) [Val08]

Incrementally update a proof of $i$ applications to a proof of $i + 1$ applications with the same size.
Approach: Incrementally Verifiable Computation (IVC) [Val08]

Incrementally update a proof of $i$ applications to a proof of $i + 1$ applications with the same size
Approach: Incrementally Verifiable Computation (IVC) [Val08]

Incrementally update a proof of $i$ applications to a proof of $i + 1$ applications with the same size.
Nova: A Built zkSNARK for Recursive Computation
Nova: A Built zkSNARK for Recursive Computation

- Relaxed R1CS: A new NP-complete relation
- Folding Scheme for Relaxed R1CS
- IVC Scheme
- Nova
Nova: A Built zkSNARK for Recursive Computation

Relaxed R1CS: A new NP-complete relation

Folding Scheme for Relaxed R1CS

IVC Scheme

Efficient zkSNARK of IVC Proof

Succinctness and ZK Layer

Novo
Nova: A Built zkSNARK for Recursive Computation

Implementation and Evaluation

- Implemented in Rust [6000 LOC][github.com/Microsoft/Nova]
- Smallest per step prover time (roughly two $|F| + c$ size multi-exps) [1M gates: 1.1s]
- Smallest recursion overhead $c$ (roughly two scalar multiplications) [20k gates]
- $O(\log |F|)$ size compressed proofs [1M gates: 8 KB]
Presented in this Talk

Relaxed R1CS: A new NP-complete relation

Folding Scheme for Relaxed R1CS

IVC Scheme

Succinctness and ZK Layer

zkSNARK for Relaxed R1CS

Efficient zkSNARK of IVC Proof

Nova

Outline

1. Show that a folding scheme for NP implies IVC
2. Develop a folding scheme for Relaxed R1CS
Valiant’s Original IVC Approach [Val08, BCTV14]
Valiant’s Original IVC Approach [Val08, BCTV14]

\[
(F')^i(z_0) = z_i
\]

SNARK Proof of

\[
Verify \Pi_i
\]

Computation \( F' \)

\[
F
\]

\[
F
\]

\[
\pi_i
\]

\[
z_i
\]

\[
z_{i+1}
\]
Valiant's Original IVC Approach [Val08, BCTV14]

\[ F(z_0) = z_i \]

\[ (F')^i(z_0) = z_i \]

\[ (F')^{i+1}(z_0) = z_{i+1} \]
Valiant’s Original IVC Approach \cite{Val08,BCTV14}

\[
\text{SNARK Proof of } (F')^i(z_0) = z_i
\]

\[
\text{Computation } F'
\]

\[
\pi_i \rightarrow F \
\rightarrow z_{i+1}
\]

\[
\text{Verify } \Pi_i
\]

\[
\pi_{i+1} \quad \text{SNARK Proof of } (F')^{i+1}(z_0) = z_{i+1}
\]

**Drawbacks**

- SNARKs in \cite{BCTV14} require expensive cycles of pairing-friendly curves and trusted-setup
- Utilizing SNARKs without trusted setup require much larger verifier circuits
Halo’s Approach: Partially Verify Proofs \cite{BGH19,BCLMS20,BDFG20}

\[ F'(z_i - 1) = z_i \]

Proof of \( F'(z_{i-1}) = z_i \)

Computation \( F' \)

\( z_i \) \rightarrow \( F \) \rightarrow \( z_{i+1} \)

\( \pi_i \) \rightarrow \text{Partially Verify } \Pi_i
Halo’s Approach: Partially Verify Proofs \cite{BGH19,BCLMS20,BDFG20}

Proof of $F'(z_{i-1}) = z_i$

Computation $F'$

\begin{align*}
F(z_i - 1) & = z_{i+1} \\
\text{Partially Verify } \Pi_i & \\
\text{Defe the rest into } \text{acc}_i & \\
\text{acc}_i & = \text{acc}_{i+1}
\end{align*}
Halo’s Approach: Partially Verify Proofs \([\text{BGH19, BCLMS20, BDFG20}]\)
Halo’s Approach: Partially Verify Proofs \cite{BGH19, BCLMS20, BDFG20}

**Drawbacks**

- Partially checking $\pi_i$ in-circuit is still expensive
- Generating $\pi_i$ is concretely and asymptotically expensive
Nova: Reduce Claims rather than Verify Proofs

Fold reduces the task of checking two instances to the task of checking a single instance

\[
F'(z_{i-1}) = z_i
\]

\[
(F')^{i-1}(z_0) = z_{i-1}
\]
Nova: Reduce Claims rather than Verify Proofs

Fold reduces the task of checking two instances to the task of checking a single instance.

Claim of $F'(z_{i-1}) = z_i$
Claim of $(F')^{-1}(z_0) = z_{i-1}$

Claim of $(F')^i(z_0) = z_i$

Fold reduces the task of checking two instances to the task of checking a single instance.
How do we implement Fold?

$$z_i \rightarrow F \rightarrow z_{i+1}$$

$$u_i \rightarrow \text{Fold} \rightarrow U_{i+1}$$

$$U_i \rightarrow \text{Computation } F' \rightarrow U_{i+1}$$
Solution: Folding Schemes

A folding scheme interactively reduces the claims \((u_1, w_1), (u_2, w_2) \in R\) to a claim \((u, w) \in R\)
Solution: Folding Schemes

A folding scheme interactively reduces the claims \((u_1, w_1), (u_2, w_2) \in R\) to a claim \((u, w) \in R\)

### Completeness

If \(u_1, u_2\) are satisfiable then \(u\) is satisfiable
Solution: Folding Schemes

A folding scheme interactively reduces the claims \((u_1, w_1), (u_2, w_2) \in R\) to a claim \((u, w) \in R\)

**Completeness**

If \(u_1, u_2\) are satisfiable then \(u\) is satisfiable

**Knowledge Soundness**

If prover outputs satisfying \(w\) then it must know satisfying \(w_1, w_2\)
Solution: Folding Schemes

A folding scheme interactively reduces the claims \((u_1, w_1), (u_2, w_2) \in R\) to a claim \((u, w) \in R\)

### Completeness
If \(u_1, u_2\) are satisfiable then \(u\) is satisfiable

### Efficiency
Folding should be much cheaper for the verifier than checking an instance

### Knowledge Soundness
If prover outputs satisfying \(w\) then it must know satisfying \(w_1, w_2\)
Non-Interactive Folding Schemes

A public-coin folding scheme can be made non-interactive via the Fiat-Shamir Transform.
Given a non-interactive folding scheme for NP, $F'$ can verifiably fold $u_i$ and $U_i$ by running the folding verifier.
Given a non-interactive folding scheme for NP, $F'$ can verifiably fold $u_i$ and $U_i$ by running the folding verifier.

**Witness** $w_i$ consists of a trace of the execution of $F(z_{i-1}) = z_i$.

**Diagram:**
- **Computation $F'$**
  - Input: $z_i$
  - Output: $z_{i+1}$
  - $u_i$ is input to the folding verifier.
- **Folding Verifier**
  - Input: $u_i, U_i$
  - Output: $U_{i+1}$
- **Folding Prover**
  - Input: $w_i, W_i$
  - Output: $W_{i+1}$

The diagram illustrates the flow of information through the folding scheme, with the folding verifier and prover exchanging information to verify the folding process.
Folding an NP-Complete Relation

We start with R1CS a popular algebraic constraint system for NP

An R1CS statement consists of constraint matrices $A, B, C,$ and vector $x$. A witness vector $W$ is satisfying if for $Z = (W, x, 1)$
Attempt to Fold R1CS Instances

\[ W_1 \quad (A, B, C, x_1) \]
\[ W_2 \quad (A, B, C, x_2) \]
Attempt to Fold R1CS Instances

\[ W_1, W_2 \]
\[ (A, B, C, x_1) \]
\[ (A, B, C, x_2) \]

Challenge \( r \)
Attempt to Fold R1CS Instances

\[ W_1 \quad (A, B, C, x_1) \]
\[ W_2 \quad (A, B, C, x_2) \]

\[ x \leftarrow x_1 + r \cdot x_2 \]

Take a random linear combination

\[ x \leftarrow x_1 + r \cdot x_2 \]
Attempt to Fold R1CS Instances

$$W_1 \leftarrow (A, B, C, x_1)$$

$$W_2 \leftarrow (A, B, C, x_2)$$

$$x \leftarrow x_1 + r \cdot x_2$$

$$W \leftarrow W_1 + r \cdot W_2$$

Challenge $r$

Take a random linear combination

$x \leftarrow x_1 + r \cdot x_2$
Attempt to Fold R1CS Instances

\[
\begin{align*}
W_1 & \leftarrow (A, B, C, x_1) \\
W_2 & \leftarrow (A, B, C, x_2) \\
P & \leftarrow \text{Challenge } r \\
V & \leftarrow \text{Take a random linear combination} \\
x & \leftarrow x_1 + r \cdot x_2 \\
W & \leftarrow W_1 + r \cdot W_2 \\
W & \leftarrow (A, B, C, x)
\end{align*}
\]
Attempt to Fold R1CS Instances

Unfortunately, letting $Z_i = (W_i, x_i, 1)$ and $Z = Z_1 + r \cdot Z_2$, we have that $AZ \cdot BZ \neq CZ$. 
Attempt to Fold R1CS Instances

Unfortunately, letting $Z_i = (W_i, x_i, 1)$ and $Z = Z_1 + r \cdot Z_2$, we have that $AZ \cdot BZ \neq CZ$

\[ CZ = AZ_1 \circ BZ_1 + r \cdot AZ_2 \circ BZ_2 \]
Attempt to Fold R1CS Instances

Unfortunately, letting $Z_i = (W_i, x_i, 1)$ and $Z = Z_1 + r \cdot Z_2$, we have that $AZ \cdot BZ \neq CZ$

$$CZ = AZ_1 \circ BZ_1 + r \cdot AZ_2 \circ BZ_2$$

$$AZ \cdot BZ = AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2)$$
Attempt to Fold R1CS Instances

Unfortunately, letting $Z_i = (W_i, x_i, 1)$ and $Z = Z_1 + r \cdot Z_2$, we have that $AZ \cdot BZ \neq CZ$

$$CZ = AZ_1 \cdot BZ_1 + r \cdot AZ_2 \cdot BZ_2$$

$$AZ \cdot BZ = AZ_1 \cdot BZ_1 + r \cdot (AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1) + r^2 \cdot (AZ_2 \cdot BZ_2)$$

To absorb the error terms we introduce an error vector $E$ in the statement
Attempt to Fold R1CS Instances

Unfortunately, letting $Z_i = (W_i, x_i, 1)$ and $Z = Z_1 + r \cdot Z_2$, we have that $AZ \cdot BZ \neq CZ$

\[ CZ = AZ_1 \cdot BZ_1 + r \cdot AZ_2 \cdot BZ_2 \]

To absorb the extra $r$ factor we introduce scalar $u$ in the statement

\[ AZ \cdot BZ = AZ_1 \cdot BZ_1 + r \cdot (AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1) + r^2 \cdot (AZ_2 \cdot BZ_2) \]

To absorb the error terms we introduce an error vector $E$ in the statement
A relaxed R1CS statement additionally contains an error vector $E$ and scalar $u$. A witness vector $W$ is satisfying if for $Z = (W, x, u)$

\[
A Z \odot B Z = u \cdot C Z + E
\]
Folding Scheme for Relaxed R1CS (Almost)

$W_1$ $(A, B, C, E_1, u_1, x_1)$

$W_2$ $(A, B, C, E_2, u_2, x_2)$
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow A Z_1 \cdot B Z_2 + A Z_2 \cdot B Z_1 - u_1 \cdot C Z_2 - u_2 \cdot C Z_1 \]

\[ W_1 \quad (A, B, C, E_1, u_1, x_1) \]

\[ W_2 \quad (A, B, C, E_2, u_2, x_2) \]
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 \]
\[ -u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]

\[ W_1 \quad \text{(A, B, C, } E_1, u_1, x_1) \]
\[ W_2 \quad \text{(A, B, C, } E_2, u_2, x_2) \]

\[ T \]
\[ u \leftarrow u_1 + r \cdot u_2 \]
\[ x \leftarrow x_1 + r \cdot x_2 \]
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]

\[ W \leftarrow W_1 + r \cdot W_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]

\[ (A, B, C, E_1, u_1, x_1) \]

\[ (A, B, C, E_2, u_2, x_2) \]

\[ u \leftarrow u_1 + r \cdot u_2 \]

\[ x \leftarrow x_1 + r \cdot x_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]
Folding Scheme for Relaxed R1CS (Almost)

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]

\[ W \leftarrow W_1 + r \cdot W_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]

\[ T \leftarrow \begin{array}{c}
W_1 \\
W_2
\end{array} \quad (A, B, C, E_1, u_1, x_1) \]

\[ (A, B, C, E_2, u_2, x_2) \]

\[ u \leftarrow u_1 + r \cdot u_2 \]

\[ x \leftarrow x_1 + r \cdot x_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]

\[ (A, B, C, E, u, x) \]
Folding Scheme for Relaxed R1CS (Almost)

- $T \leftarrow A \mathbf{Z}_1 \cdot B \mathbf{Z}_2 + A \mathbf{Z}_2 \cdot B \mathbf{Z}_1 - u_1 \cdot C \mathbf{Z}_2 - u_2 \cdot C \mathbf{Z}_1$
- $W \leftarrow W_1 + r \cdot W_2$
- $E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2$

Problem: Verifier cannot enforce that prover folds $W$ correctly.

$u \leftarrow u_1 + r \cdot u_2$
$x \leftarrow x_1 + r \cdot x_2$

$E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2$
Folding Scheme for Relaxed R1CS (Almost)

Problem: Verifier cannot enforce that prover folds correctly

\[ T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \]

\[ W \leftarrow W_1 + r \cdot W_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]

Problem: Because \( |E| = O(|W|) \) the verifier is not succinct

\[ u \leftarrow u_1 + r \cdot u_2 \]

\[ x \leftarrow x_1 + r \cdot x_2 \]

\[ E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \]

\[ W_1 \]

\[ (A, B, C, E_1, u_1, x_1) \]

\[ W_2 \]

\[ (A, B, C, E_2, u_2, x_2) \]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement:

\[\begin{align*}
(E_1, W_1) & \quad (A, B, C, \overline{E}_1, u_1, \overline{W}_1, x_1) \\
(E_2, W_2) & \quad (A, B, C, \overline{E}_2, u_2, \overline{W}_2, x_2)
\end{align*}\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement

\[
T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1
\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement

\[
T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1
\]

\[
\begin{align*}
(E_1, W_1) &\quad (A, B, C, E_1, u_1, W_1, x_1) \\
(E_2, W_2) &\quad (A, B, C, E_2, u_2, W_2, x_2)
\end{align*}
\]

\[
\bar{T} = \text{com}(T)
\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement

\[
E_1, W_1, (A, B, C, E_2, u_1, W_1, x_1)
\]

\[
E_2, W_2, (A, B, C, E_2, u_2, W_2, x_2)
\]

\[
T = \text{com}(T)
\]

\[
T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1
\]

\[
- u_1 \cdot CZ_2 - u_2 \cdot CZ_1
\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat $(E, W)$ as part of the witness and store their commitments in the statement

$$(E_1, W_1) \quad (A, B, C, E_1, u_1, W_1, x_1)$$
$$(E_2, W_2) \quad (A, B, C, E_2, u_2, W_2, x_2)$$

$$T \leftarrow AZ_1 \odot BZ_2 + AZ_2 \odot BZ_1$$
$$-u_1 \cdot CZ_2 - u_2 \cdot CZ_1$$

$u \leftarrow u_1 + r \cdot u_2$
$x \leftarrow x_1 + r \cdot x_2$
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement:

\[
\begin{align*}
(E_1, W_1) & \quad \quad \quad \quad (A, B, C, \bar{E}_1, u_1, \bar{W}_1, x_1) \\
(E_2, W_2) & \quad \quad \quad \quad (A, B, C, \bar{E}_2, u_2, \bar{W}_2, x_2)
\end{align*}
\]

\[
T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 \\
- u_1 \cdot CZ_2 - u_2 \cdot CZ_1
\]

\[
\begin{align*}
\bar{W} & \leftarrow \bar{W}_1 + r \cdot \bar{W}_2 \\
\bar{E} & \leftarrow \bar{E}_1 + r \cdot \bar{T} + r^2 \cdot \bar{E}_2 \\
u & \leftarrow u_1 + r \cdot u_2 \\
x & \leftarrow x_1 + r \cdot x_2
\end{align*}
\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement

\[
\begin{align*}
(E_1, W_1) &\quad (A, B, C, E_1, u_1, W_1, x_1) \\
(E_2, W_2) &\quad (A, B, C, E_2, u_2, W_2, x_2)
\end{align*}
\]

\[
T \leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \\
W \leftarrow W_1 + r \cdot W_2 \\
E \leftarrow E_1 + r \cdot T + r^2 \cdot E_2
\]

Challenge \(r\)

\[
E \leftarrow \overline{E}_1 + r \cdot \overline{T} + r^2 \cdot \overline{E}_2 \\
W \leftarrow \overline{W}_1 + r \cdot \overline{W}_2 \\
u \leftarrow u_1 + r \cdot u_2 \\
x \leftarrow x_1 + r \cdot x_2
\]
Folding Scheme for Relaxed R1CS from Additive Commitments

Treat \((E, W)\) as part of the witness and store their commitments in the statement

\[
\begin{align*}
T &\leftarrow AZ_1 \cdot BZ_2 + AZ_2 \cdot BZ_1 \\
&\quad - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 \\
W &\leftarrow W_1 + r \cdot W_2 \\
E &\leftarrow E_1 + r \cdot T + r^2 \cdot E_2
\end{align*}
\]

\[
\begin{align*}
(E, W) &\leftarrow (E_1, W_1) \quad (A, B, C, E_2, u_1, W_1, x_1) \\
(E, W) &\leftarrow (E_2, W_2) \quad (A, B, C, E_2, u_2, W_2, x_2)
\end{align*}
\]

\[
\begin{align*}
\overline{E} &\leftarrow \overline{E}_1 + r \cdot \overline{T} + r^2 \cdot \overline{E}_2 \\
\overline{W} &\leftarrow \overline{W}_1 + r \cdot \overline{W}_2 \\
u &\leftarrow u_1 + r \cdot u_2 \\
x &\leftarrow x_1 + r \cdot x_2
\end{align*}
\]
Summary

We design a folding-friendly variant of R1CS, Relaxed R1CS

We construct a folding scheme for Relaxed R1CS

We construct IVC using our folding scheme for Relaxed R1CS

Our techniques result in a recursive zkSNARK with state-of-the-art efficiency

eprint.iacr.org/2021/370

github.com/Microsoft/Nova

akothapalli@cmu.edu
References

[Val08] Valiant. Incrementally Verifiable Computation or Proofs of Knowledge imply Time/Space Efficiency


[BGH19] Bowe, Green, Hopwood. Halo: Recursive Proof Composition without a Trusted Setup


[BDFG20] Boneh, Drake, Fisch, Gabizon. Halo Infinite: Recursive zkSNARKs from any Additive Polynomial Commitment Scheme


[BBBF19] Boneh, Bonneau, Bünz, Fisch. Verifiable Delay Functions