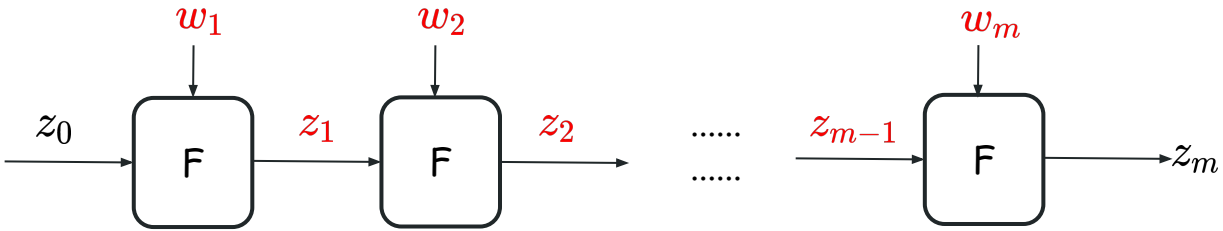
ProtoStar: Generic Efficient IVC Schemes

Benedikt Bünz Binyi Chen

Espresso Systems

Incrementally Verifiable Computation (IVC)

Goal: Prove correctness of (non-deterministic) iterative computation



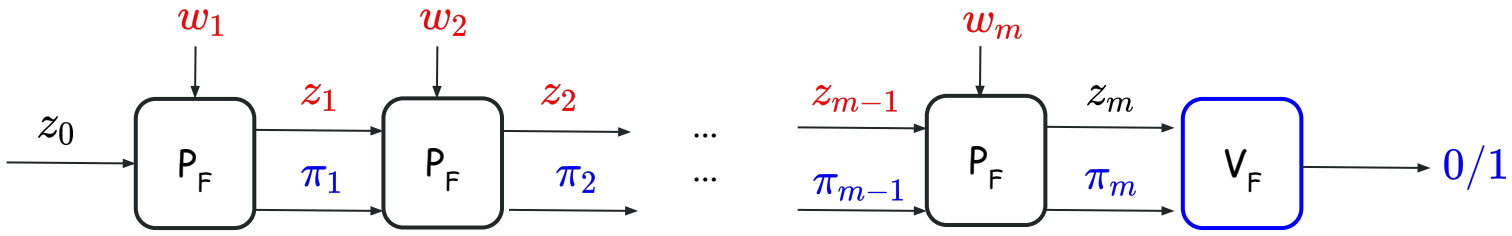
Prove that $\exists w_1, \ldots w_m, z_1, \ldots z_{m-1} : \forall i \in [m] F(z_{i-1}, w_i) = z_i$

Examples

- Verifiable delay functions: $\text{VDF}(z_0) = F^T(z_0)$
- Succinct blockchain (e.g. Mina): $\mathbf{z_i}$ = ledger state; $\mathbf{w_i}$ = txs in block i
- zkVM, zkEVM: instruction set $\mathcal{F} = \{\mathbf{op_1}, \ldots, \mathbf{op_k}\}$

Incrementally Verifiable Computation (IVC)

IVC [Valiant08]: Prover P_F and Verifier V_F such that



Completeness:
Given accepted proof π_{i-1} for input z_{i-1}, can generate valid proof π_i for $z_i = F(z_{i-1}, w_i)$

Knowledge soundness:
Can extract the witness (i.e. intermediate w and z) from the **final** output and the **final** IVC proof

Generalization: Proof-Carrying Data (PCD)
Prove chain of computations -> prove tree/DAG of computations

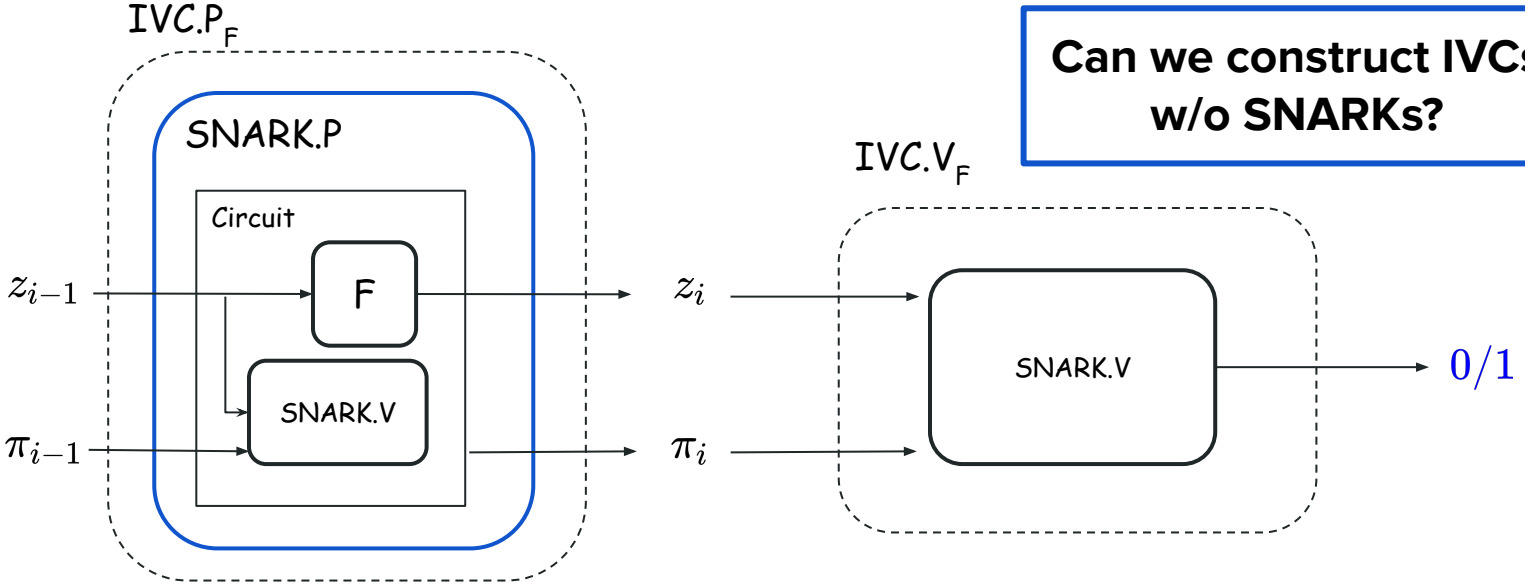
Why not directly using SNARKs?

Why IVC?
Succinct Non-interactive Arg-of-Knowledge (SNARKs) are also succinct

IVC's Advantages over SNARKs

- Small prover memory and preprocessing overhead (step func F is small)
- Enable proving dynamic number of calls to step function F
- Easier parallel proving (e.g., with PCD)
- and more.....

IVC from SNARKs [Val08, BCCT13, BCTV14, COS20]



Can we construct IVCs w/o SNARKs?

Caveats:

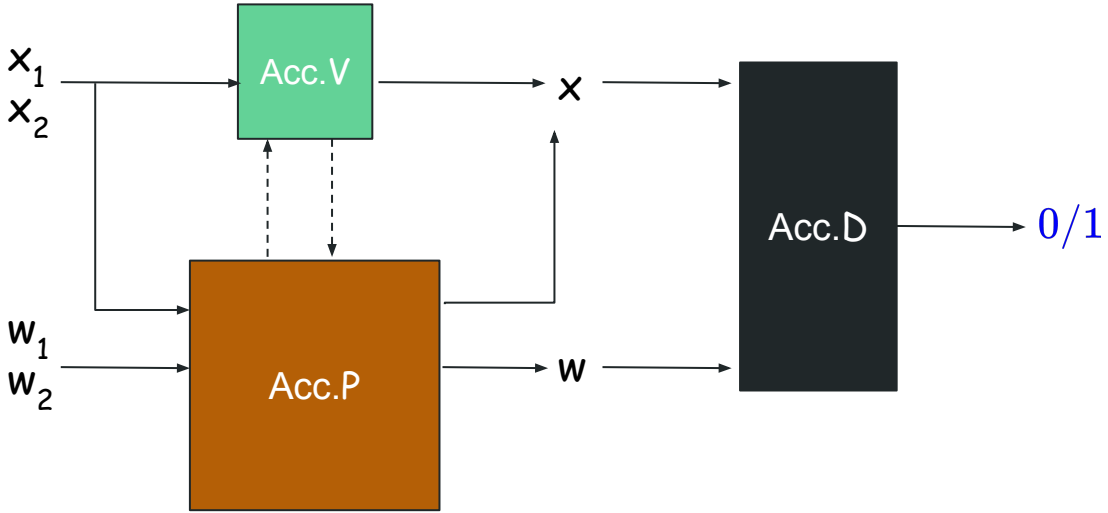
- expensive recursive verifier circuit
- expensive pairing-friendly cycles/trusted setup

Split Accumulation/Folding [BCLMS20, KST21]

NP Relation R: $(x, w) \in R$

Small Claim — Large Witness

More efficient than SNARK.V!



Acc/Folding: reduce the task of proving **two** NP instances into proving a **single** instance

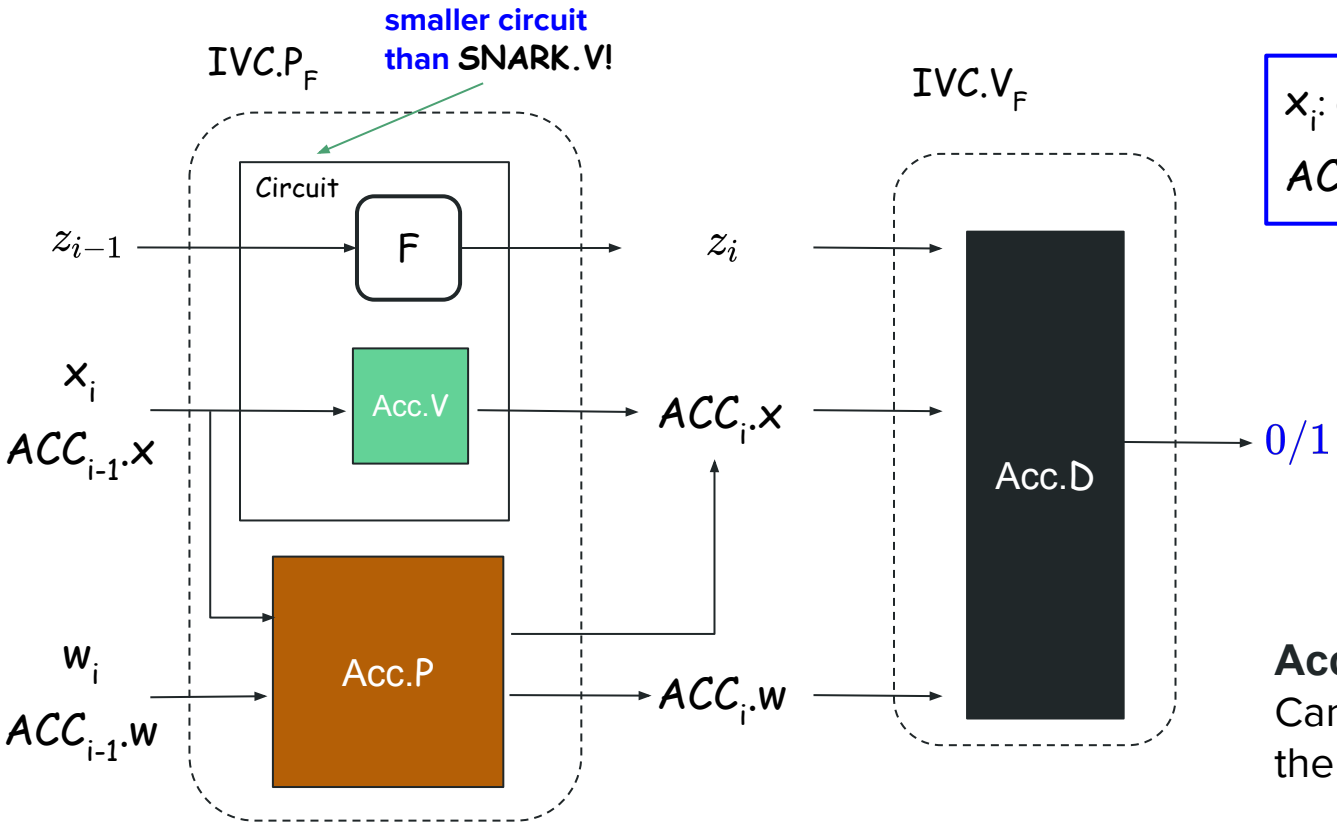
Completeness:

If x_1, x_2 satisfiable, then x satisfiable

Knowledge Soundness:

If prover knows valid w, then it also knows valid w_1, w_2

IVC from Split Accumulation/Folding [BCLMS20, KST21]



x_i: claim that $C(z_{i-1})=z_i$

$ACC_i.x$: claim that $C^i(z_0)=z_i$

Acc.D may not be succinct:
Can use SNARK to delegate the computation of Acc.D

Limitations of State-of-the-Arts

Existing Acc/Folding schemes

- Use R1CS constraint systems
  - Less expressive per gate (e.g., no **high-degree/lookup** gate support)
  - Nova follow-ups (e.g. Sangria, Origami, etc) support Plonk/lookup gates, but not **efficient** enough
- Do not support efficient circuit branching (except [KS22])
  - Can't select step **F** (e.g. EVM opcode) **at runtime** without a circuit that is **linear** in all possible **F**s (e.g. the EVM opcode set)

Essential for zkEVM applications!

- Ad-hoc constructions with different proofs
  - No unified and general framework

Our Contributions

- **General recipe for IVC/PCD schemes**
  - Can fold **any** special-sound multi-round protocols **efficiently** (Nova is a special case)
- **Efficient folding/IVC with highly expressive gates**
  - high-degree gates + lookup + non-uniform computation
    - Easily extendable to support Customizable Constraint System (CCS) [STW23]
  - Dominant IVC proving cost: (**no dependence on gate degree and table size**)
    - 1 MSM of size |w|
    - Recursive circuit: 3 G-exps

	Protostar	HyperNova	SuperNova
Language	Degree d Plonk/CCS	Degree d CCS	R1CS (degree 2)
Non-uniform	yes	no	yes
P native	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$
extra P native w/ lookup	$O(\lvert\ell_{\mathsf{lk}}\rvert)\,\mathbb{G}$	$O(T)\,\mathbb{F}$	N/A
P recursive	$3\mathbb{G}$ $(d+O(1))\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $(d+O(1))\,\mathbb{F}$	$1\mathbb{G}$ $d\log n\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $O(d\log n)\mathbb{F}$	$2\mathbb{G}$ $\mathsf{H}_{\mathsf{in}}+O_\lambda(1)\mathsf{H}+1\mathsf{H}_{\mathbb{G}}$
extra P recursive w/ lookup	$1\mathsf{H}$	$O(\log T)\,\mathsf{H}$ $O(\ell_{\mathsf{lk}}\log T)\,\mathbb{F}$	N/A

Significantly fewer hashes/F-ops in the circuit

General Recipe for Folding/Accumulation

General Recipe for Folding/Acc

Goal: Fold instances for NP Relation $\mathcal{R}$

Step 1: Build a multi-round special-sound protocol **Π** for relation $\mathcal{R}$ **usually easy**

Step 2: Transform **Π** to a non-interactive argument **NARK(Π)**

Innovation: generic & efficient transform

Step 3: Build a folding/acc scheme for the verifier check of **NARK(Π)**

General Recipe for Folding/Acc

Step 1: Build a multi-round special-sound protocol **Π** for relation $\mathcal{R}$ **usually easy**

Step 1.a: Transform **Π** to **CV[Π]** with a compressed verifier for relation $\mathcal{R}$

Step 2: Transform **Π** to a non-interactive argument **NARK(Π)**

Innovation: generic & efficient transform

Step 3: Build a folding/acc scheme for the verifier check of **NARK(Π)**

General Recipe for Folding/Acc

Step 1: Build a multi-round special-sound protocol **Π** for relation $\mathcal{R}$ **usually easy**

Step 1.a: Transform **Π** to **CV[Π]** with a compressed verifier for relation $\mathcal{R}$

Step 2: Transform **CV[Π]** to a non-interactive argument **NARK(CV[Π])**

Innovation: generic & efficient transform

Step 3: Build a folding/acc scheme for the verifier check of **NARK(CV[Π])**

# of G-ops independent of the degree of the verifier check!

Special-Sound Protocols: Example

Check: P knows **a**, **b**, **c** s.t. $\mathbf{a}_i * \mathbf{b}_i = \mathbf{c}_i$ for all $i \in [n]$

k-special-sound:
can extract valid witness from **k** accepting transcripts

$(k_1, \ldots, k_u)$-special-soundness
for multi-round protocols

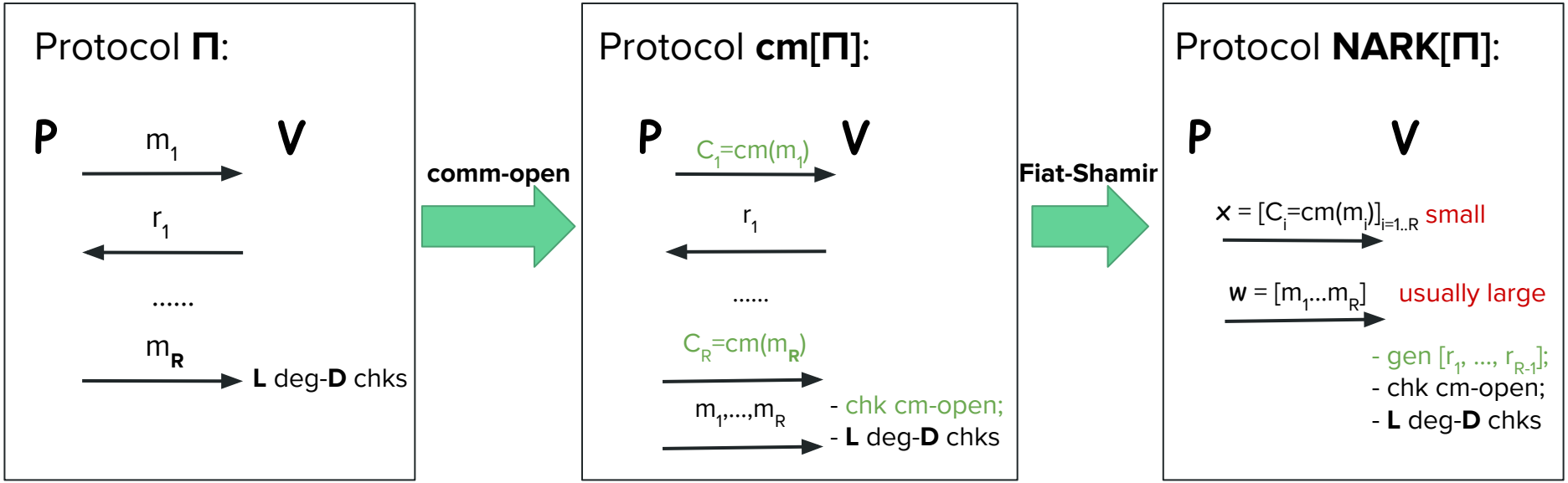
R: # of prover moves = **1**
D: max degree of each check = **2**
L: # of verifier checks = **n**

check $\mathbf{a}_i * \mathbf{b}_i = \mathbf{c}_i$ for all $i \in [n]$

Why easy to design?
No need for *succinct* communication/verifier
No need to be *non-interactive*

1-special-sound!

Transform to Non-Interactive Argument of Knowledge



[AFK22]: Special-soundness of Π ⇒ Knowledge-soundness of NARK[Π]

Folding/Acc for **NARK[Π]**'s verifier checks

ignore public input for simplicity

R: # of prover moves
d: max degree of each check
L: # of verifier checks

err-vec cm

$ACC.x = \left\{\vec{\mathbf{C}}', \vec{\mathbf{r}}';\quad \mathbf{E}\right\}$

$ACC.w = \left\{\vec{\mathbf{m}}'\right\}$

Acc's chk: $\vec{f}_d\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

err-vec

commitments **chals**

$\boldsymbol{\pi}.x = \left\{\vec{\mathbf{C}} = [C_i]_{i=1}^{R}, \vec{\mathbf{r}} = [r_i]_{i=1}^{R-1}\right\}$

$\boldsymbol{\pi}.w = \left\{\vec{\mathbf{m}} = [m_i]_{i=1}^{R}\right\}$ **prover msgs**

NARK vfy chk: $\vec{f}_d(\vec{\mathbf{m}}, \vec{\mathbf{r}}) \stackrel{?}{=} 0^L$

terms in $\mathbf{f}_d$ all have deg **d** for simplicity

Goal: Fold *ACC* and **π** into a new *ACC*

Can generalize to fold two *ACCs*

For inhomogeneous $\mathbf{f}_d$: add a slack variable **u = 1**

Folding/Acc for **NARK[Π]**'s verifier checks

d: max degree of each check
L: # of verifier checks

ACC.x = $\left\{\left(\vec{\mathbf{C}}', \vec{\mathbf{r}}'\right); \mathbf{E}\right\}$

π.x = $\left\{\left(\vec{\mathbf{C}}, \vec{\mathbf{r}}\right)\right\}$

ACC.w = $\left\{\vec{\mathbf{m}}'\right\}$

π.w = $\left\{\vec{\mathbf{m}}\right\}$

Acc's chk: $\vec{f}_d\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

NARK vfy chk: $\vec{f}_d\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) \stackrel{?}{=} 0^L$

$\vec{f}_d\left(X \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}', X \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right)$

const coeff: *Acc*'s checks
X^d coeff: NARK vfy checks

Example:

$L = 1,\ d = 2$

$f_2\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) = \vec{\mathbf{r}}_1^2$

$$\vec{f}_2\left(X \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}', X \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right) = \left(X \cdot \vec{\mathbf{r}}_1 + \vec{\mathbf{r}}'_1\right)^2 = \vec{\mathbf{r}}'^2_1 + (\ldots)X + \vec{\mathbf{r}}_1^2 X^2$$

Acc chk

NARK vfy chk

Folding/Acc for **NARK[Π]**'s verifier checks

d: max degree of each check
L: # of verifier checks

ACC.x = $\left\{\left(\vec{\mathbf{C}}', \vec{\mathbf{r}}'\right); \mathbf{E}\right\}$ ACC.w = $\left\{\vec{\mathbf{m}}'\right\}$ **Acc's chk:** $\vec{f}_d\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

π.x = $\left\{\left(\vec{\mathbf{C}}, \vec{\mathbf{r}}\right)\right\}$ π.w = $\left\{\vec{\mathbf{m}}\right\}$ **NARK vfy chk:** $\vec{f}_d\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) \stackrel{?}{=} 0^L$

$\left[\vec{e}_j \in \mathbb{F}^L\right]_{j=1}^{d-1}$ s.t. $\vec{e} + \sum_{j=1}^{d-1} \vec{e}_j X^j \stackrel{?}{=} \vec{f}_d\left(X \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}', X \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right)$

const coeff: *Acc*'s checks
X^d coeff: NARK vfy checks

New **Acc's chk** w/ folded msgs/errs! $\vec{f}_d\left(\alpha \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}', \alpha \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right) = \vec{e} + \sum_{j=1}^{d-1} \vec{e}_j \alpha^j$

P$_{(Acc,\pi)}$

O(dL) **G-exps** to commit $\mathbf{E_j}$

$Acc.w \leftarrow Acc.w + \alpha \cdot \pi.w$

$\left[\mathbf{E_j} \leftarrow cm(\vec{e}_j)\right]_{j=1}^{d-1}$ →

$\alpha \leftarrow \mathbb{F}$ **(d+1)-special-sound** ←

$Acc.x \leftarrow Acc.x + \alpha \cdot \pi.x$ ↔

$Acc.x.\mathbf{E} \leftarrow Acc.x.\mathbf{E} + \sum_{j=1}^{d-1} \alpha^j \mathbf{E_j}$ ↔

V$_{(Acc.x,\pi.x)}$

O(d) **G-exps** to fold $\mathbf{C_i}$ and **E**

Expensive for large d : (

Compressing Verification Checks for High-deg Gates

O(dL) **G-ops** to commit $\mathbf{E_j}$

$$[\mathbf{E_j} \leftarrow cm(\vec{e}_j)]_{j=1}^{d-1}, \text{ where } \vec{e}_j \in \mathbb{F}^L$$

L: # of verifier checks

Idea: Can we get a new SS protocol with single chk (i.e. L = 1)?
The folding can use identity commitment without any G-ops!

Example:

$$V_{sps}^{(1)}(x_1, x_2) = x_1 + x_2 \stackrel{?}{=} 0$$

$$V_{sps}^{(2)}(x_1, x_2) = x_1 \cdot x_2 \stackrel{?}{=} 0$$

$$V_{sps}^{(1)}(x_1, x_2) + \beta \cdot V_{sps}^{(2)}(x_1, x_2) = (x_1 + x_2) + \beta x_1 x_2 \stackrel{?}{=} 0$$

for random β

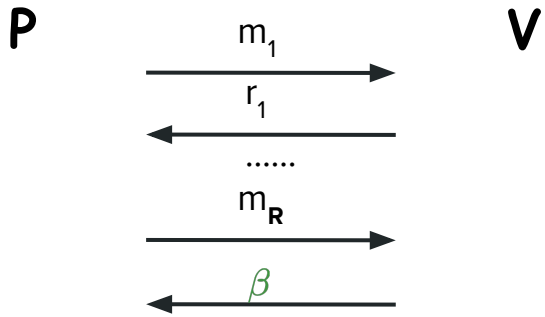
In general: Acc's chk:

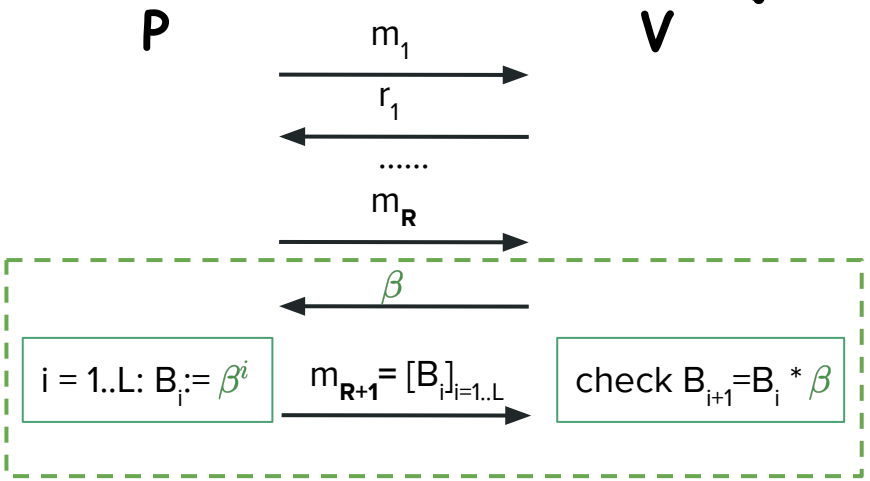
$$V_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e} \in \mathbb{F}^L$$

$$V'_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}', \beta\right) = \sum_{j=1}^{L} \beta^j \cdot V_{sps}^{(j)}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} e \in \mathbb{F}$$

Compressing Verification Checks for High-deg Gates

Idea: Can we make powers of β different vars/msgs sent by the prover?





The new verifier check:

$$V'_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}', \beta\right) = \sum_{j=1}^{L} \beta^j \cdot V^{(j)}_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} e \in \mathbb{F}$$

- **Gate-deg** = d+L **: (**

The new verifier check:

homogenous deg = d+1

$$V'_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}', [B_j]\right) = \sum_{j=1}^{L} B_j \cdot V^{(j)}_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} e \in \mathbb{F}$$

Caveat: **Acc prover commits to O(L) terms**

Compressing Verification Checks for High-deg Gates

The NARK check:

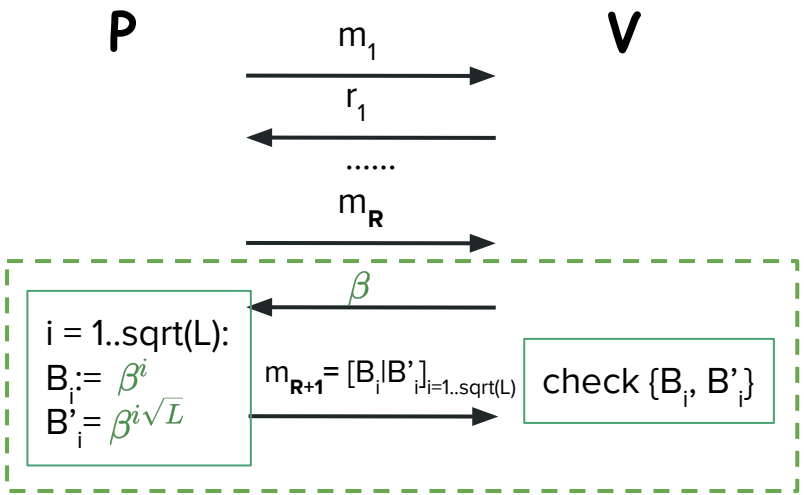
Caveat: Acc prover needs to commit to O(L) terms

deg = d+2

$$V'_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}', [B_j]\right) = \sum_{j=1}^{L} B_j \cdot V^{(j)}_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right)$$

$$V'_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}', [B_j, B'_j]\right) = \sum_{i=1}^{\sqrt{L}} \sum_{j=1}^{\sqrt{L}} B_i B'_j \cdot V^{(i+j\sqrt{L})}_{sps}\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right)$$

Fixed SS proto:



Acc Verifier: R+2 **G-exps** to fold $\mathbf{C_i}$ and **E**

no overhead w/ lookup!

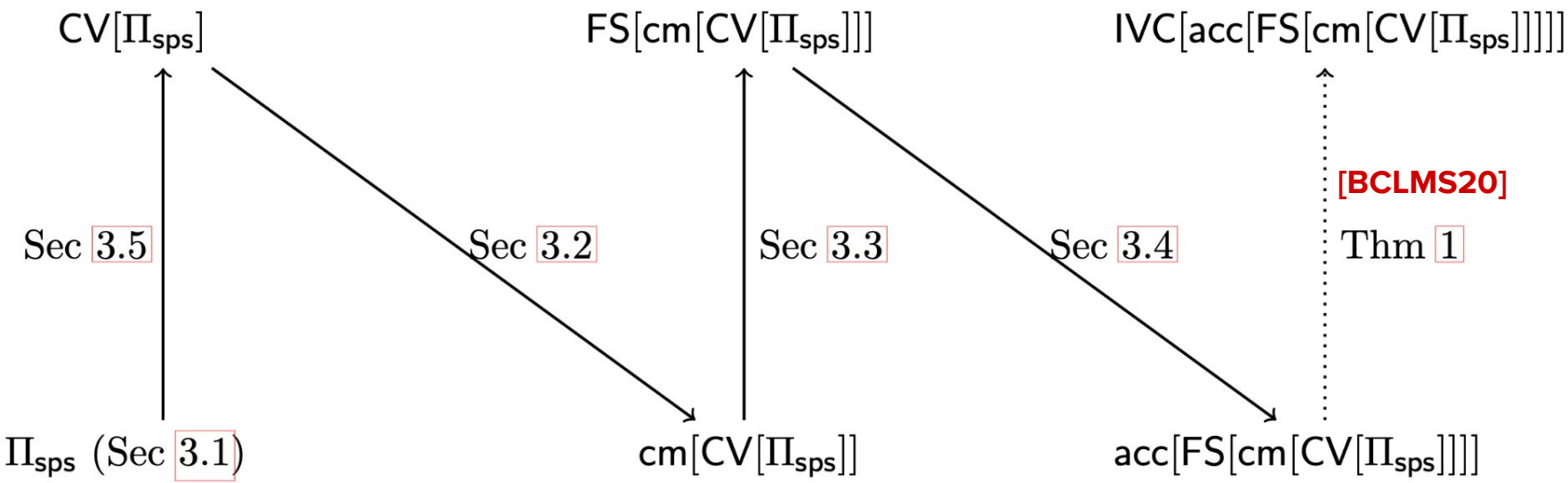
Acc Prover: $O\left(\sqrt{\ell}\right)$ **extra G-exps** to commit $\mathbf{m_{R+1}}$

Acc Chks: $O\left(\sqrt{\ell}\right)$ deg-2 checks for correctness of $[B_i, B'_i]$

Sol: add **1** sep comm for the err vec w/ len $O\left(\sqrt{\ell}\right)$

Summary of the General Recipe

Can use [NguBonSet23] for efficient IVC compilation



How to construct Π_{sps} for expressive constraint systems?

Efficient Special-sound Protocols for Expressive Relations

Permutation Check/High-deg Gates/Circuit Selection

- We obtain 1-move special-sound protocols for
  - Permutation relation/High-degree gate-check relation
  - Non-uniform circuit selection relation
  - Customizable constraint system (CCS) [STW23]
- The schemes are trivial as the constraints are algebraic
  - P sends witness, V checks the algebraic equations

A justification that

Step 1: Build a multi-round special-sound protocol **Π** for relation R **usually easy**

What if the constraints are not algebraic?

Lookup

lookup vals $\{w_i\}_{i=1}^{\ell} \subseteq \{t_i\}_{i=1}^{T}$ **table vals**

[Hab22]

$$\exists [m_i]_{i=1}^{T} : \sum_{i=1}^{\ell} \frac{1}{X + w_i} = \sum_{i=1}^{T} \frac{m_i}{X + t_i}$$

Great Feature:
Prover messages are **sparse!**

Efficient **IVC** prover w/ cheap prv-msg comms computation?

Challenges:
Acc msg/err-vec are not sparse

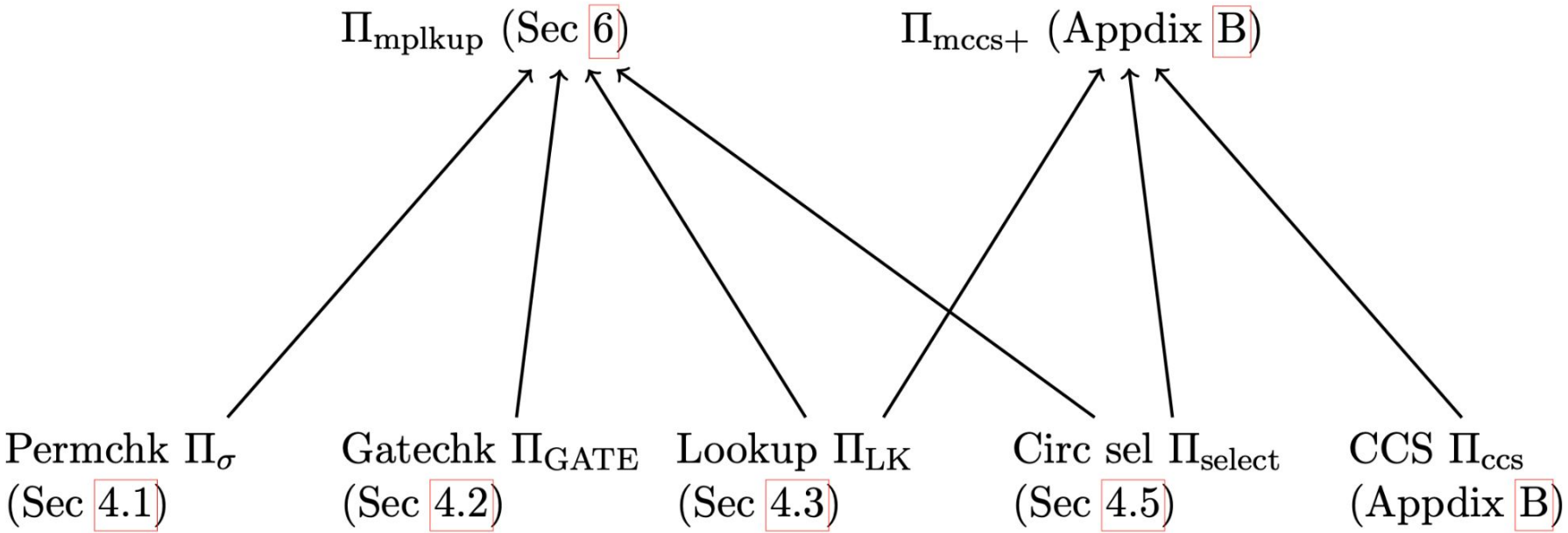
Solution:
Update msg/err-vec **comms** *homomorphically*

Special-sound protocol Π_{LK} for $\mathcal{R}_{\mathrm{LK}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{LK}}, \mathbf{w} \in \mathbb{F}^{\ell})$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{LK}})$
Compute $\mathbf{m} \in \mathbb{F}^{T}$ such that $\mathbf{m}_i := \sum_{j=1}^{\ell} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$	$\mathbf{w}, \mathbf{m}$ →	
	← r	$r \leftarrow\$ \mathbb{F}$
Compute $\mathbf{h} \in \mathbb{F}^{\ell}$, $\mathbf{g} \in \mathbb{F}^{T}$ $\mathbf{h}_i := \frac{1}{\mathbf{w}_i + r} \forall i \in [\ell]$ $\mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T]$	$\mathbf{h}, \mathbf{g}$ →	$\sum_{i=1}^{\ell} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$ $\mathbf{h}_i \cdot (\mathbf{w}_i + r) \stackrel{?}{=} 1 \forall i \in [\ell]$ $\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \forall i \in [T]$

Extend to vector-lookups

Putting the Pieces Together



Special-sound protocols for ProtoStar/ProtoStar$_{\text{ccs}}$:
Compositions of the building block SS protocols

Key Takeaway

- **General** recipe for **efficient** folding/accumulation schemes
- **Simple** Special-sound protocols for **expressive** relations
- ProtoStar IVC: **3 G-exps** in the recursive circuit for supporting
  - arbitrary high-deg gates
  - arbitrarily large lookup tables
  - arbitrarily many opcodes in the VM

	PROTOSTAR	HyperNova	SuperNova
Language	Degree d Plonk/CCS	Degree d CCS	R1CS (degree 2)
Non-uniform	yes	no	yes
P native	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$
extra P native w/ lookup	$O(\lvert\ell_{\mathsf{lk}}\rvert)\,\mathbb{G}$	$O(T)\,\mathbb{F}$	N/A
P recursive	$3\mathbb{G}$ $(d+O(1))\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $(d+O(1))\,\mathbb{F}$	$1\mathbb{G}$ $d\log n\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $O(d\log n)\mathbb{F}$	$2\mathbb{G}$ $\mathsf{H}_{\mathsf{in}}+O_\lambda(1)\mathsf{H}+1\mathsf{H}_{\mathbb{G}}$
extra P recursive w/ lookup	1H	$O(\log T)\,\mathsf{H}$ $O(\ell_{\mathsf{lk}}\log T)\,\mathbb{F}$	N/A

Highly practical with [NguBonSet23] IVC compiler

Significantly fewer hashes & non-native F-ops in the circuit

Thanks!

https://eprint.iacr.org/2023/620.pdf

Non-uniform Circuit Selection

Special-sound protocol Π_{select} for circuit selecting relation $\mathcal{R}_{\text{select}}$

Prover $\mathsf{P}(\mathbf{b} \in \mathbb{F}^n, pc \in \mathbb{F})$		**Verifier** V
	$\mathbf{b}, pc \longrightarrow$	
		$b_i \cdot (pc - i) \stackrel{?}{=} 0 \forall i \in [n]$
		$b_i \cdot (b_i - 1) \stackrel{?}{=} 0 \forall i \in [n]$
		$\sum_{i \in [n]} b_i \stackrel{?}{=} 1$

Definition 14. *For an integer n the relation $\mathcal{R}_{select}$ is the set of tuples $(\mathbf{b}, pc) \in \mathbb{F}^n \times \mathbb{F}$ such that $b_i = 0 \forall i \in [n] \setminus \{pc\}$ and if $pc \in [n]$ then $b_{pc} = 1$.*

Protocol $\Pi_{\mathbf{mccs+}} = \langle \mathsf{P}(\mathcal{C}_{\mathbf{mccs+}}, \mathsf{pi}, \mathbf{w}), \mathsf{V}(\mathcal{C}_{\mathbf{mccs+}}, \mathsf{pi} = (pc \in [I], \mathsf{pi}')) \rangle$**:**

1. P sends V vector $\mathbf{b} = (0, \ldots, 0, b_{pc} = 1, 0, \ldots, 0) \in \mathbb{F}^I$.
2. V checks that $b_i \cdot (1 - b_i) \stackrel{?}{=} 0$ and $b_i \cdot (i - pc) \stackrel{?}{=} 0$ for all $i \in [I]$, and $\sum_{i \in [I]} b_i \stackrel{?}{=} 1$.
3. P computes vector $\mathbf{m} \in \mathbb{F}^T$ such that $\mathbf{m}_i := \sum_{j \in L_{pc}} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$.
4. P sends V vectors $\mathbf{w}, \mathbf{m}$.
5. V computes $\mathbf{z} = (1, \mathsf{pi}, \mathbf{w}) \in \mathbb{F}^n$ and checks that
$$\sum_{k=1}^{I} b_k \cdot \sum_{i=1}^{q} c_{i,k} \cdot \bigcirc_{j \in S_{i,k}} M_{j,k} \cdot \mathbf{z} = \mathbf{0}^m .$$
6. V samples and sends P random challenge $r \leftarrow\!\!\$\, \mathbb{F}$.
7. P computes vectors $\mathbf{h} \in \mathbb{F}^{\ell_{\mathsf{lk}}}$, $\mathbf{g} \in \mathbb{F}^T$ such that
$$\mathbf{h}_i := \frac{1}{\mathbf{w}_{L_{pc}[i]} + r} \forall i \in [\ell_{\mathsf{lk}}] , \qquad \mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T] .$$
8. V checks that $\sum_{i=1}^{\ell_{\mathsf{lk}}} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$ and
$$\sum_{j=1}^{I} b_j \cdot \left[\mathbf{h}_i \cdot (\mathbf{w}_{L_j[i]} + r) \right] \stackrel{?}{=} 1 \quad \forall i \in [\ell_{\mathsf{lk}}] ,$$
$$\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \quad \forall i \in [T] .$$

Folding/Acc for **NARK[Π]**'s verifier checks

ignore public input for simplicity

R: # of prover moves
d: max degree of each check
L: # of verifier checks

err-vec cm

$ACC.x = \left\{\vec{\mathbf{C}}', \vec{\mathbf{r}}', \quad u; \quad \mathbf{E}\right\}$

slack

$ACC.w = \left\{\vec{\mathbf{m}}'\right\}$

Acc's chk: $\sum_{j=0}^{d} u^{d-j} \cdot \vec{f}_j\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

commitments chals

$\boldsymbol{\pi}.x = \left\{\vec{\mathbf{C}} = [C_i]_{i=1}^{R}, \vec{\mathbf{r}} = [r_i]_{i=1}^{R-1}, \; u = 1\right\}$

$\boldsymbol{\pi}.w = \left\{\vec{\mathbf{m}} = [m_i]_{i=1}^{R}\right\}$ prover msgs

NARK vfy chk: $\sum_{j=0}^{d} \vec{f}_j(\vec{\mathbf{m}}, \vec{\mathbf{r}}) \stackrel{?}{=} 0^L$

terms in $\mathbf{f}_j$ all have deg **j**

Goal: Fold *ACC* and **π** into a new *ACC*

Can generalize to fold two *ACCs*

Folding/Acc for **NARK[Π]**'s verifier checks

d: max degree of each check
L: # of verifier checks

ACC.x = $\left\{\left(\vec{\mathbf{C}}', \vec{\mathbf{r}}', u\right); \mathbf{E}\right\}$

π.x = $\left\{\left(\vec{\mathbf{C}}, \vec{\mathbf{r}}, 1\right)\right\}$

ACC.w = $\left\{\vec{\mathbf{m}}'\right\}$

π.w = $\left\{\vec{\mathbf{m}}\right\}$

Acc's chk: $\sum_{j=0}^{d} u^{d-j} \cdot \vec{f}_j\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

NARK vfy chk: $\sum_{j=0}^{d} \vec{f}_j\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) \stackrel{?}{=} 0^L$

$$\sum_{j=0}^{d} (X+u)^{d-j} \vec{f}_j\left(X \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}',\ X \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right)$$

const coeff: Acc's checks
X^d coeff: NARK vfy checks

Example:

$L = 1,\ d = 2$
$f_0 = 0$
$f_1\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) = \vec{\mathbf{r}}_1$
$f_2\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) = \vec{\mathbf{r}}_1^2$
$V_{sps} = \vec{\mathbf{r}}_1 + \vec{\mathbf{r}}_1^2$

$$\begin{aligned}
&(X+u)^2 f_0 + (X+u) f_1 + f_2 \\
&= (X+u)^2 \cdot 0 + (X+u)\left(X \cdot \vec{\mathbf{r}}_1 + \vec{\mathbf{r}}'_1\right) + \left(X \cdot \vec{\mathbf{r}}_1 + \vec{\mathbf{r}}'_1\right)^2 \\
&= \left(u\vec{\mathbf{r}}'_1 + \vec{\mathbf{r}}'^2_1\right) + (\ldots)X + \left(\vec{\mathbf{r}}_1 + \vec{\mathbf{r}}_1^2\right)X^2
\end{aligned}$$

Acc's chk **NARK vfy chk**

Folding/Acc for **NARK[Π]**'s verifier checks

d: max degree of each check
L: # of verifier checks

$ACC.x = \left\{\left(\vec{\mathbf{C}}', \vec{\mathbf{r}}', u\right); \mathbf{E}\right\}$ $\quad ACC.w = \left\{\vec{\mathbf{m}}'\right\}$ $\quad$ **Acc's chk:** $\sum_{j=0}^{d} u^{d-j} \cdot \vec{f}_j\left(\vec{\mathbf{m}}', \vec{\mathbf{r}}'\right) \stackrel{?}{=} \vec{e}$

$\pi.x = \left\{\left(\vec{\mathbf{C}}, \vec{\mathbf{r}}, 1\right)\right\}$ $\quad \pi.w = \left\{\vec{\mathbf{m}}\right\}$ $\quad$ **NARK vfy chk:** $\sum_{j=0}^{d} \vec{f}_j\left(\vec{\mathbf{m}}, \vec{\mathbf{r}}\right) \stackrel{?}{=} 0^L$

$$\left[\vec{e}_j \in \mathbb{F}^L\right]_{j=1}^{d-1} \text{ s.t. } \vec{e} + \sum_{j=1}^{d-1} \vec{e}_j X^j \stackrel{?}{=} \sum_{j=0}^{d} (X+u)^{d-j} \vec{f}_j\left(X \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}',\ X \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right)$$

const coeff: Acc's checks
X^d coeff: NARK vfy checks

New **Acc's chk** w/ folded msgs/errs!

$$\sum_{j=0}^{d} (\alpha+u)^{d-j} \vec{f}_j\left(\alpha \cdot \vec{\mathbf{m}} + \vec{\mathbf{m}}',\ \alpha \cdot \vec{\mathbf{r}} + \vec{\mathbf{r}}'\right) = \vec{e} + \sum_{j=1}^{d-1} \vec{e}_j \alpha^j$$

$P_{(Acc,\pi)}$

O(dL) **G-ops** to commit $\mathbf{E_j}$

$Acc.w \leftarrow Acc.w + \alpha \cdot \pi.w$

$[\mathbf{E_j} \leftarrow cm(\vec{e}_j)]_{j=1}^{d-1}$

$\alpha \leftarrow \mathbb{F}$ **(d+1)-special-sound**

$Acc.x \leftarrow Acc.x + \alpha \cdot \pi.x$

$Acc.x.\mathbf{E} \leftarrow Acc.x.\mathbf{E} + \sum_{j=1}^{d-1} \alpha^j \mathbf{E_j}$

$V_{(Acc.x,\pi.x)}$

O(d) **G-ops** to fold $\mathbf{C_i}$ and **E**

Expensive for large d : (