FLI: Folding Lookup Instances

Albert Garreta, Nethermind Research
Joint work with Ignacio Manzur

NETHERMIND RESEARCH

Folding

Folding

- Fix a relation R, consisting of pairs $(x; w)$

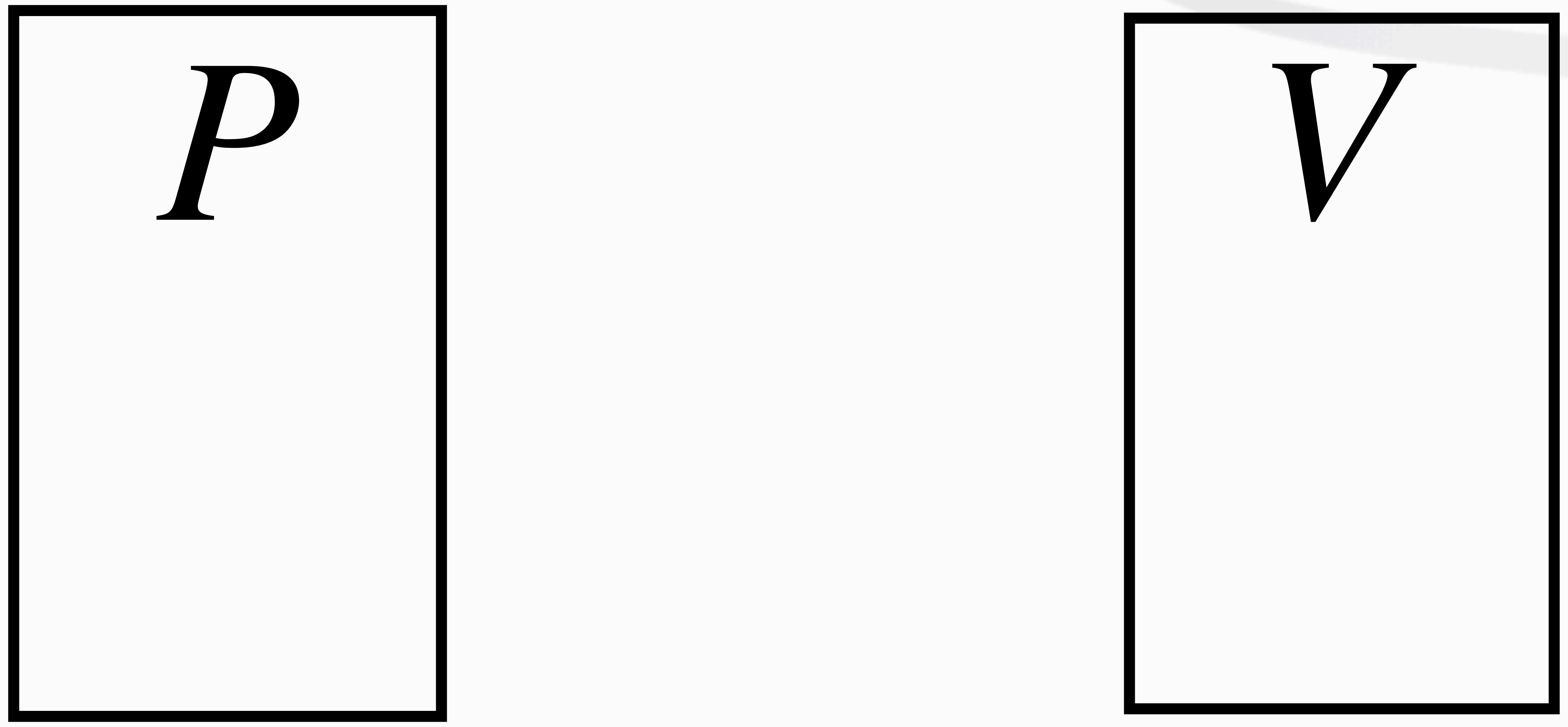
Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

A folding scheme is an interactive protocol between P and V where:



Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances x_1, x_2.
- P also has witnesses w_1, w_2 such that $(x_1; w_1), (x_2; w_2) \in R$.

P	V
$(x_1; w_1)$	(x_1)
$(x_2; w_2)$	(x_2)

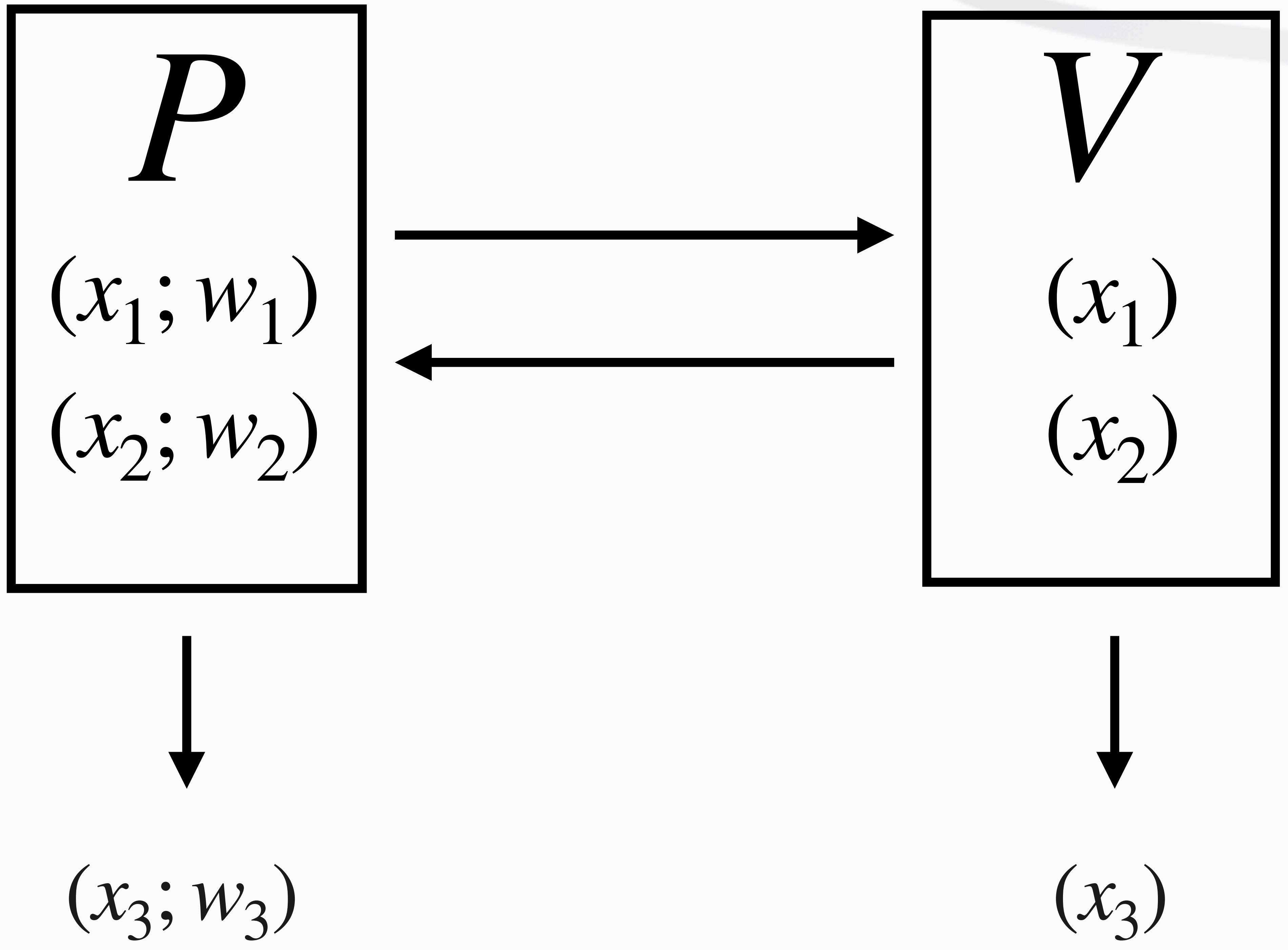
NETHERMIND RESEARCH

Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances x_1, x_2.
- P also has witnesses w_1, w_2 such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:



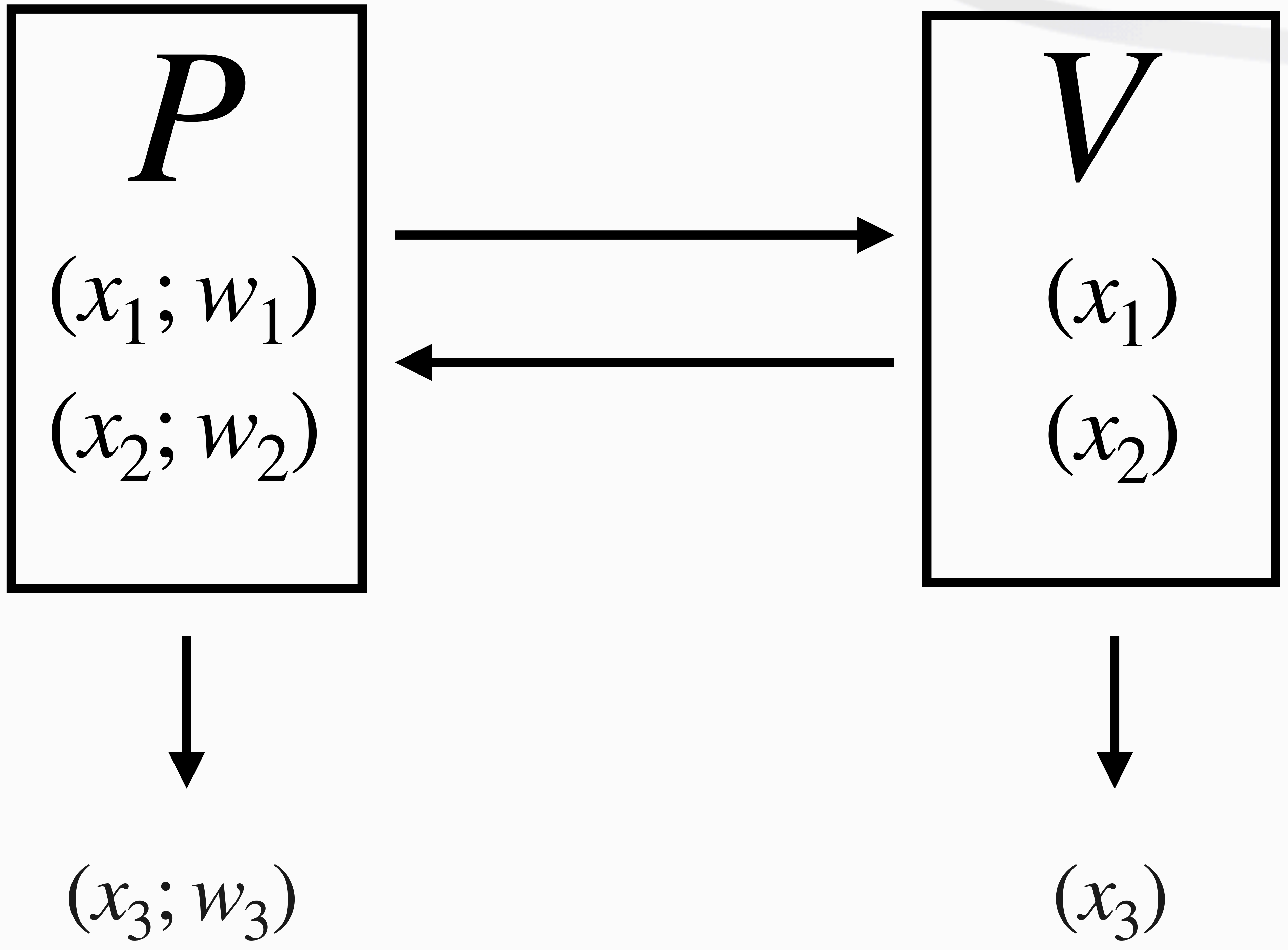
NETHERMIND RESEARCH

Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances x_1, x_2.
- P also has witnesses w_1, w_2 such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:
- If $(x_3; w_3) \in R$, then $(x_1; w_1), (x_2; w_2) \in R$, e.w.n.p.



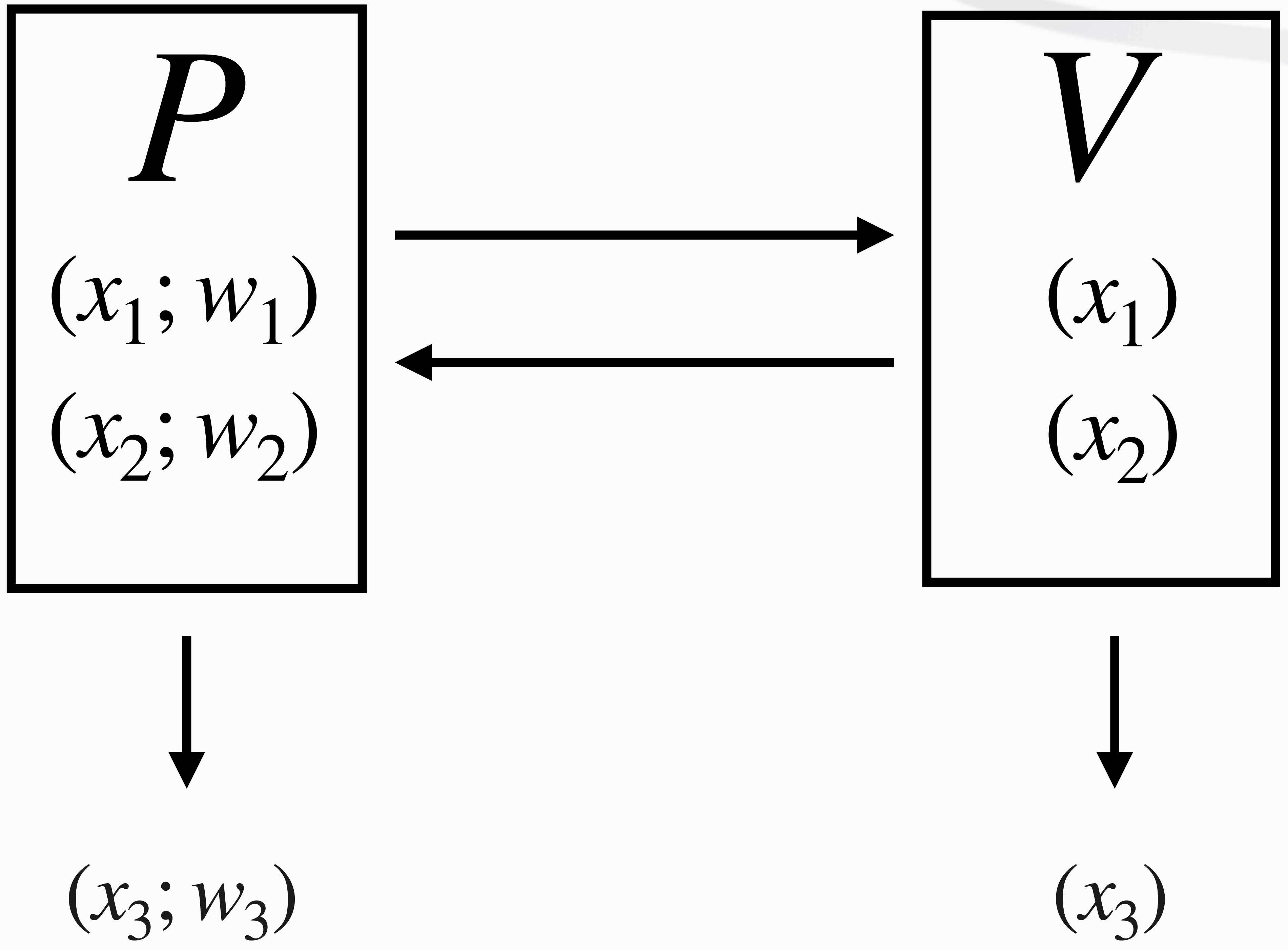
NETHERMIND RESEARCH

Folding

- Fix a relation R, consisting of pairs $(x; w)$
- x is a public instance, w is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances x_1, x_2.
- P also has witnesses w_1, w_2 such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:
- If $(x_3; w_3) \in R$, then $(x_1; w_1), (x_2; w_2) \in R$, e.w.n.p.



$$(x_3; w_3) \in R \quad \Leftrightarrow_{(e.w.n.p)} \quad \begin{cases} (x_1; w_1) & \in R, \\ (x_2; w_2) & \in R \end{cases}$$

NETHERMIND RESEARCH

Folding

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

$(x_1; w_1) \in R$ $(x_2; w_2) \in R$ —Fold→ $(x_3; w_3) \in R$

NETHERMIND RESEARCH

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

$(x_1; w_1) \in R$ $\quad$ $(x_2; w_2) \in R$ $\quad \xrightarrow{\text{Fold}} \quad$ $(x_3; w_3) \in R$

- Commitments play a crucial role in folding schemes.

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

- Commitments play a crucial role in folding schemes.
- The instances x_1, x_2, x_3 all contain a commitment to w_1, w_2, w_3, respectively. I.e.

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

$$(x_1; w_1) \in R \qquad (x_2; w_2) \in R \quad \xrightarrow{\text{Fold}} \quad (x_3; w_3) \in R$$

- Commitments play a crucial role in folding schemes.
- The instances x_1, x_2, x_3 all contain a commitment to w_1, w_2, w_3, respectively. I.e.

$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$

NETHERMIND RESEARCH

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

$(x_1; w_1) \in R$ $\quad$ $(x_2; w_2) \in R$ $\quad \xrightarrow{\text{Fold}} \quad$ $(x_3; w_3) \in R$

- Commitments play a crucial role in folding schemes.
- The instances x_1, x_2, x_3 all contain a commitment to w_1, w_2, w_3, respectively. I.e.

$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$

$\left(x_1, \mathrm{cm}_{w_1}; w_1\right) \in R$ $\quad$ $\left(x_2, \mathrm{cm}_{w_2}; w_2\right) \in R$ $\quad \xrightarrow{\text{Fold}} \quad$ $\left(x_3, \mathrm{cm}_{w_3}; w_3\right) \in R$

Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.

$$(x_1; w_1) \in R \qquad (x_2; w_2) \in R \quad \xrightarrow{\text{Fold}} \quad (x_3; w_3) \in R$$

- Commitments play a crucial role in folding schemes.
- The instances x_1, x_2, x_3 all contain a commitment to w_1, w_2, w_3, respectively. I.e.

$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$

$$\left(x_1, \text{cm}_{w_1}; w_1\right) \in R \qquad \left(x_2, \text{cm}_{w_2}; w_2\right) \in R \quad \xrightarrow{\text{Fold}} \quad \left(x_3, \text{cm}_{w_3}; w_3\right) \in R$$

- Usually the commitment is homomorphic: $\text{cm}_{w_1+w_2} = \text{cm}_{w_1} + \text{cm}_{w_2}$

Folding from 5000 km

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \qquad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

NETHERMIND RESEARCH

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R$, $(x_2, \mathsf{cm}_{w_2}; w_2) \in R$.

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

V

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

P	V
$(x_1, \mathsf{cm}_{w_1}; w_1)$	(x_1, cm_{w_1})
$(x_2, \mathsf{cm}_{w_2}; w_2)$	(x_2, cm_{w_2})

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages ⟷

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages

Uniformly sampled challenge $\alpha \in \mathbb{F}$

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R$, $(x_2, \mathsf{cm}_{w_2}; w_2) \in R$.

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages

Uniformly sampled challenge $\alpha \in \mathbb{F}$

$(x_3, \mathsf{cm}_{w_3}; w_3)$

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages ↔

Uniformly sampled challenge $\alpha \in \mathbb{F}$ ←

⇩

$(x_3, \mathsf{cm}_{w_3}; w_3)$

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

- Where $x_3 = x_1 + \alpha x_2 \qquad w_3 = w_1 + \alpha w_2$

NETHERMIND RESEARCH

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R, \quad (x_2, \mathsf{cm}_{w_2}; w_2) \in R.$

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages ⟷

Uniformly sampled challenge $\alpha \in \mathbb{F}$ ⟵

⇩

$(x_3, \mathsf{cm}_{w_3}; w_3)$

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

- Where $x_3 = x_1 + \alpha x_2 \qquad w_3 = w_1 + \alpha w_2$
- V computes cm_{w_3} using that $\mathsf{cm}_{w_1} + \alpha \mathsf{cm}_{w_2} = cm_{w_1 + \alpha w_2}$

Folding from 5000 km

- Fix $(x_1, \mathsf{cm}_{w_1}; w_1) \in R$, $(x_2, \mathsf{cm}_{w_2}; w_2) \in R$.

P

$(x_1, \mathsf{cm}_{w_1}; w_1)$

$(x_2, \mathsf{cm}_{w_2}; w_2)$

Exchange messages ⟷

Uniformly sampled challenge $\alpha \in \mathbb{F}$ ⟵

⇩

$(x_3, \mathsf{cm}_{w_3}; w_3)$

V

(x_1, cm_{w_1})

(x_2, cm_{w_2})

- Where $x_3 = x_1 + \alpha x_2$ $w_3 = w_1 + \alpha w_2$
- V computes cm_{w_3} using that $\mathsf{cm}_{w_1} + \alpha \mathsf{cm}_{w_2} = cm_{w_1 + \alpha w_2}$
- (Disclaimer: this is an extremely simplified, technically incorrect, blueprint)

NETHERMIND RESEARCH

Nova, R1CS, and other relations

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.
- One can design folding schemes for many other relations: HyperNova, ProtoStar, ProtoGalaxy, NeutronNova, etc.

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.
- One can design folding schemes for many other relations: HyperNova, ProtoStar, ProtoGalaxy, NeutronNova, etc.
- For example, for lookup relations.

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.
- One can design folding schemes for many other relations: HyperNova, ProtoStar, ProtoGalaxy, NeutronNova, etc.
- For example, for lookup relations.
- A lookup relation R for a set S consists of pairs $(\text{Com}(v); v)$ where

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.
- One can design folding schemes for many other relations: HyperNova, ProtoStar, ProtoGalaxy, NeutronNova, etc.
- For example, for lookup relations.
- A lookup relation R for a set S consists of pairs $(\mathrm{Com}(v); v)$ where
  - $v = (v_1, \ldots, v_m) \in \mathbb{F}^m$

Nova, R1CS, and other relations

- Folding schemes have become very popular since Nova (2021)
- In Nova, R are relaxed R1CS constraints of the form $A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}$. Here $A, B, C \in \mathbb{F}^{n \times n}$ are public matrices, $u \in \mathbb{F}$ is public, and $\mathbf{z}, \mathbf{e} \in \mathbb{F}^n$ is the witness.
- One can design folding schemes for many other relations: HyperNova, ProtoStar, ProtoGalaxy, NeutronNova, etc.
- For example, for lookup relations.
- A lookup relation R for a set S consists of pairs $(\text{Com}(v); v)$ where
  - $v = (v_1, \ldots, v_m) \in \mathbb{F}^m$
  - $v_i \in S$ for all $i = 1, \ldots, n$.

Lookup argument

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
  - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
  - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.
  - $S = \{(x || y) \mid x \in \{0,1\}^n,\ y = \text{SHA256}(x)\}$

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
  - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.
  - $S = \{(x || y) \mid x \in \{0,1\}^n,\ y = \text{SHA256}(x)\}$
- Why do we need lookup arguments?

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
  - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.
  - $S = \{(x || y) \mid x \in \{0,1\}^n, y = \text{SHA256}(x)\}$
- Why do we need lookup arguments?
  - The previous statements can be proved with a regular SNARK.

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
  - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.
  - $S = \{(x || y) \mid x \in \{0,1\}^n,\ y = \text{SHA256}(x)\}$
- Why do we need lookup arguments?
  - The previous statements can be proved with a regular SNARK.
  - However, arithmetizing it (i.e. writing it in Plonkish, R1CS, CCS, AIR constraints) is really expensive.

Lookup argument

- A lookup argument is a special proof system (e.g. a SNARK) for lookup relations.
- Examples:
    - When $S = [0, 2^{128} - 1]$, the lookup proves all entries in v are between 0 and 2^{128}.
    - $S = \{(x || y) \mid x \in \{0,1\}^n, \ y = \text{SHA256}(x)\}$
- Why do we need lookup arguments?
    - The previous statements can be proved with a regular SNARK.
    - However, arithmetizing it (i.e. writing it in Plonkish, R1CS, CCS, AIR constraints) is really expensive.
    - Here "expensive" means that a huge circuit is required. E.g. SHA-256 requires $\approx 2^{20}$ constraints as R1CS.

The FLI scheme

The FLI scheme

- Let R_S be a lookup relation for a set S, so

The FLI scheme

- Let R_S be a lookup relation for a set S, so

$$R_S = \{(\mathsf{cm}_v; v) \mid v \in \mathbb{F}^m,\ v_i \in S\ \forall i\}$$

The FLI scheme

- Let R_S be a lookup relation for a set S, so

$$R_S = \{(\text{cm}_v; v) \mid v \in \mathbb{F}^m,\ v_i \in S\ \ \forall i\}$$

- To simplify exposition, from now on we forget about commitments. We write $(\text{cm}_v; v) \in R_S$ as $v \subseteq S$.

The FLI scheme

- Let R_S be a lookup relation for a set S, so

$$R_S = \{(\mathsf{cm}_v; v) \mid v \in \mathbb{F}^m,\ v_i \in S\ \forall i\}$$

- To simplify exposition, from now on we forget about commitments. We write $(\mathsf{cm}_v; v) \in R_S$ as $v \subseteq S$.
- We have two lookup instances $v_1 \subseteq S,\ v_2 \subseteq S$.

The FLI scheme

- Let R_S be a lookup relation for a set S, so

$$R_S = \{(\mathrm{cm}_v; v) \mid v \in \mathbb{F}^m,\ v_i \in S\ \forall i\}$$

- To simplify exposition, from now on we forget about commitments. We write $(\mathrm{cm}_v; v) \in R_S$ as $v \subseteq S$.
- We have two lookup instances $v_1 \subseteq S$, $v_2 \subseteq S$.
- We want P and V to create a new instance $v_3 \subseteq S$ so that

The FLI scheme

- Let R_S be a lookup relation for a set S, so

$$R_S = \{(\mathrm{cm}_v; v) \mid v \in \mathbb{F}^m,\ v_i \in S\ \forall i\}$$

- To simplify exposition, from now on we forget about commitments. We write $(\mathrm{cm}_v; v) \in R_S$ as $v \subseteq S$.
- We have two lookup instances $v_1 \subseteq S$, $v_2 \subseteq S$.
- We want P and V to create a new instance $v_3 \subseteq S$ so that

$$v_3 \subseteq S \Leftrightarrow_{(e.w.n.p)} v_1 \subseteq S, v_2 \subseteq S$$

The FLI scheme

The FLI scheme

- Note $v \subseteq S$ if and only if:

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

Notation: $M \in R_{elem}$.

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

    Notation: $M \in R_{elem}$.
  - $M \cdot S^T = v^T$

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

    Notation: $M \in R_{elem}$.
  - $M \cdot S^T = v^T$
- Elementary vector = all entries are 0 except for one entry, which is 1.

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

    Notation: $M \in R_{elem}$.
  - $M \cdot S^T = v^T$
- Elementary vector = all entries are 0 except for one entry, which is 1.
- Example: $S = (1,2,3,4)$, $v = (4,2)$, then

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

    Notation: $M \in R_{elem}$.
  - $M \cdot S^T = v^T$
- Elementary vector = all entries are 0 except for one entry, which is 1.
- Example: $S = (1,2,3,4)$, $v = (4,2)$, then

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} \qquad M \cdot S^T = v^T$$

The FLI scheme

- Note $v \subseteq S$ if and only if:
  - There is a $|v| \times |S|$ matrix M such that all its rows are elementary vectors.

    Notation: $M \in R_{elem}$.
  - $M \cdot S^T = v^T$
- Elementary vector = all entries are 0 except for one entry, which is 1.
- Example: $S = (1,2,3,4)$, $v = (4,2)$, then

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} \qquad M \cdot S^T = v^T$$

- Simply, the $i-th$ row of M indicates a position of S that equals v_i

The FLI scheme 🛫

The FLI scheme 🛫

$$R_{elem} = \{\, |v| \times |S| \text{ matrices all whose rows are elementary vectors} \}$$

The FLI scheme 🛬

$$R_{elem} = \{\, |v| \times |S| \text{ matrices all whose rows are elementary vectors} \}$$

- We can replace the claim $v \subseteq S$ by “there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.

The FLI scheme 🛬

$$R_{elem} = \{\, |v| \times |S| \text{ matrices all whose rows are elementary vectors}\}$$

- We can replace the claim $v \subseteq S$ by "there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.
- Then to fold two claims

The FLI scheme 🛬

$$R_{elem} = \{\, |v| \times |S| \text{ matrices all whose rows are elementary vectors} \}$$

- We can replace the claim $v \subseteq S$ by “there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.
- Then to fold two claims

$$v_1 \subseteq S,\ v_2 \subseteq S$$
$$\left(\Leftrightarrow \ \exists M_1, M_2;\ M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem} \right) \qquad (*)$$

The FLI scheme 🛬

$$R_{elem} = \{ |v| \times |S| \text{ matrices all whose rows are elementary vectors} \}$$

- We can replace the claim $v \subseteq S$ by "there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.
- Then to fold two claims

$$v_1 \subseteq S,\ v_2 \subseteq S$$
$$\left(\Leftrightarrow \ \exists M_1, M_2;\ M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem} \right) \qquad (*)$$

- V sends a random element $\alpha \in \mathbb{F}$, and the claim (*) is reduced to

The FLI scheme 🛬

$$R_{elem} = \{\, |v| \times |S| \text{ matrices all whose rows are elementary vectors} \}$$

- We can replace the claim $v \subseteq S$ by “there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.
- Then to fold two claims

$$v_1 \subseteq S,\ v_2 \subseteq S$$
$$\left(\Leftrightarrow \ \ \exists M_1, M_2;\ \ M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem} \right) \qquad (*)$$

- V sends a random element $\alpha \in \mathbb{F}$, and the claim (*) is reduced to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

The FLI scheme 🛬

$$R_{elem} = \{\,|v| \times |S| \text{ matrices all whose rows are elementary vectors}\}$$

- We can replace the claim $v \subseteq S$ by "there exists $M \in R_{elem}$ s.t. $M \cdot S^T = v^T$.
- Then to fold two claims

$$v_1 \subseteq S,\ v_2 \subseteq S$$
$$\left(\Leftrightarrow\ \ \exists M_1, M_2;\ \ M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem} \right) \qquad (*)$$

- V sends a random element $\alpha \in \mathbb{F}$, and the claim (*) is reduced to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- It can be seen that, with high probability (over the choice of α), (*) holds if and only if (**) holds.

The FLI scheme 🛫

The FLI scheme 🛫

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

The FLI scheme 🛬

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- (**) is barely better than our initial claim (*). The issue is that we have two hanging claims $M_1 \in R_{elem}$ and $M_2 \in R_{elem}$.

The FLI scheme 🛬

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- (**) is barely better than our initial claim (*). The issue is that we have two hanging claims $M_1 \in R_{elem}$ and $M_2 \in R_{elem}$.
- The next step consists in folding these two claims.

The FLI scheme 🛬

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- (**) is barely better than our initial claim (*). The issue is that we have two hanging claims $M_1 \in R_{elem}$ and $M_2 \in R_{elem}$.
- The next step consists in folding these two claims.
- The idea is to define R_{elem} algebraically as follows:
  A matrix M belongs to R_{elem} if and only:

The FLI scheme 🛬

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- (**) is barely better than our initial claim (*). The issue is that we have two hanging claims $M_1 \in R_{elem}$ and $M_2 \in R_{elem}$.
- The next step consists in folding these two claims.
- The idea is to define R_{elem} algebraically as follows:
  A matrix M belongs to R_{elem} if and only:
  - $M_{ij}^2 = M_{ij}$ for all entries M_{ij} of M. This ensures M contains only 0 or 1's.

The FLI scheme 🛬

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad M_1, M_2 \in R_{elem} \qquad (**)$$

- (**) is barely better than our initial claim (*). The issue is that we have two hanging claims $M_1 \in R_{elem}$ and $M_2 \in R_{elem}$.
- The next step consists in folding these two claims.
- The idea is to define R_{elem} algebraically as follows:
  A matrix M belongs to R_{elem} if and only:
  - $M_{ij}^2 = M_{ij}$ for all entries M_{ij} of M. This ensures M contains only 0 or 1's.
  - $M \cdot \mathbf{1}^T = \mathbf{1}^T$. With the above, this ensures each row contains exactly one 1.

The FLI scheme 🛫

The FLI scheme 🛫

- $M \in R_{elem}$ iff $M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T$.

The FLI scheme 🛬

- $M \in R_{elem} \quad \text{iff} \quad M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T.$
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.

The FLI scheme 🛬

- $M \in R_{elem}$ iff $M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T$.
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.
- Accordingly, we can fold the statements $M_1 \in R_{elem}, M_2 \in R_{elem}$ using a Nova-type approach.

The FLI scheme 🛬

- $M \in R_{elem} \quad \text{iff} \quad M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T.$
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.
- Accordingly, we can fold the statements $M_1 \in R_{elem}, M_2 \in R_{elem}$ using a Nova-type approach.
- The final folded instance has the form :

The FLI scheme 🛬

- $M \in R_{elem} \quad \text{iff} \quad M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T.$
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.
- Accordingly, we can fold the statements $M_1 \in R_{elem}, M_2 \in R_{elem}$ using a Nova-type approach.
- The final folded instance has the form :

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem-relaxed}$$

The FLI scheme 🛬

- $M \in R_{elem}$ iff $M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T$.
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.
- Accordingly, we can fold the statements $M_1 \in R_{elem}, M_2 \in R_{elem}$ using a Nova-type approach.
- The final folded instance has the form :

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem-relaxed}$$

- $R_{elem-relaxed}$ is a “relaxed version” of R_{elem}, similar to a relaxed R1CS.

The FLI scheme 🛬

- $M \in R_{elem}$ iff $M_{ij}^2 = M_{ij}, \quad M \cdot \mathbf{1}^T = \mathbf{1}^T$.
- The RHS above is roughly a R1CS constraint where the witness vector is the matrix M.
- Accordingly, we can fold the statements $M_1 \in R_{elem}, M_2 \in R_{elem}$ using a Nova-type approach.
- The final folded instance has the form :

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem-relaxed}$$

- $R_{elem-relaxed}$ is a “relaxed version” of R_{elem}, similar to a relaxed R1CS.
- Leveraging the sparseness of M_i the overall cost for P and V is similar to Nova on witnesses of size $|v_i|$

A caveat

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

$$M_{fold} \cdot S^T = v_{fold}^T, \quad M_{fold} \in R_{elem}$$

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

$$M_{fold} \cdot S^T = v_{fold}^T, \quad M_{fold} \in R_{elem}$$

for M_{fold} a dense $|v| \cdot |S|$ matrix.

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

$$M_{fold} \cdot S^T = v_{fold}^T, \quad M_{fold} \in R_{elem}$$

for M_{fold} a dense $|v| \cdot |S|$ matrix.

- Hence, proving the folded statement could be very expensive.

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

$$M_{fold} \cdot S^T = v_{fold}^T, \quad M_{fold} \in R_{elem}$$

for M_{fold} a dense $|v| \cdot |S|$ matrix.

- Hence, proving the folded statement could be very expensive.
- We use the concept of SOS-decomposability (Lasso) to reduce $|S|$ to $|S|^{1/c}$ in exchange for doing "c small folds per folding step".

A caveat

- We reduced from $M_1 \cdot S^T = v_1^T, \quad M_2 \cdot S^T = v_2^T, \quad M_1, M_2 \in R_{elem}$ to

$$(M_1 + \alpha M_2) \cdot S^T = (v_1 + \alpha v_2)^T, \quad (M_1 + \alpha M_2) \in R_{elem}$$

- While M_1, M_2 are highly sparse matrices, $M_1 + \alpha M_2$ loses a bit of sparsity. Iterating this folding procedure can lead to a folded statement of the form

$$M_{fold} \cdot S^T = v_{fold}^T, \quad M_{fold} \in R_{elem}$$

for M_{fold} a dense $|v| \cdot |S|$ matrix.

- Hence, proving the folded statement could be very expensive.
- We use the concept of SOS-decomposability (Lasso) to reduce $|S|$ to $|S|^{1/c}$ in exchange for doing "c small folds per folding step".
- Note: Any other scheme working with huge SOS-dec. tables needs also to increase the number of folds per step by c (though c can be taken smaller), and FLI can make this step with less commitment costs.

Large tables and SOS decomposability

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{32} - 1\}$.

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{32} - 1\}$.
- Hence, to prove $x \in S$, one can prove (*) and then prove $x_i \in S'$, $i = 1,2,3,4$

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{32} - 1\}$.

- Hence, to prove $x \in S$, one can prove (*) and then prove $x_i \in S'$, $i = 1,2,3,4$
- This is good because: (*) is very simple; and S' is small: $|S'| = 2^{32}$

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{32} - 1\}$.

- Hence, to prove $x \in S$, one can prove (*) and then prove $x_i \in S'$, $i = 1,2,3,4$
- This is good because: (*) is very simple; and S' is small: $|S'| = 2^{32}$
- We can actually make the S' as small as wanted by making (*) longer.

Large tables and SOS decomposability

- From the Lasso paper (Setty, Thaler, 2023): A set/table of elements S is SOS decomposable if its elements can be written as algebraic expressions involving smaller sets.
- Example: $S = \{0,1,\ldots,2^{128} - 1\}$. Note we can't even store S in memory.
- An element x belongs to S if and only if

$$x = x_1 + 2^{32}x_2 + 2^{64}x_3 + 2^{96}x_4 \quad (*)$$

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{32} - 1\}$.
- Hence, to prove $x \in S$, one can prove (*) and then prove $x_i \in S'$, $i = 1,2,3,4$
- This is good because: (*) is very simple; and S' is small: $|S'| = 2^{32}$
- We can actually make the S' as small as wanted by making (*) longer.
- Jolt (Arun et al. 2023): Many S's of interest are SOS-dec. E.g. RISC-V instructions

Thanks!

FLI and SOS decomposition

FLI and SOS decomposition

- Let’s take $S = \{0, 1, \ldots, 2^{32} - 1\}$

FLI and SOS decomposition

- Let's take $S = \{0, 1, \ldots, 2^{32} - 1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16} x_2$ (*)

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32} - 1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16} - 1\}$.

FLI and SOS decomposition

- Let’s take $S = \{0, 1, \ldots, 2^{32} - 1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16} x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0, \ldots, 2^{16} - 1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32} - 1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16} - 1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.
- Following (*), we can write $v \subseteq S$ as

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32} - 1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16} - 1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.
- Following (*), we can write $v \subseteq S$ as
  - $v = v^{(1)} + 2^{16}v^{(2)}$ for vectors $v^{(1)}, v^{(2)}$.

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32}-1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16}-1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.
- Following (*), we can write $v \subseteq S$ as
  - $v = v^{(1)} + 2^{16}v^{(2)}$ for vectors $v^{(1)}, v^{(2)}$.
  - $v^{(i)} \subseteq S' \Leftrightarrow M_i \cdot S'^T = v^{(i)T}$, $M_i \in R_{elem}$, for $i = 1,2$.

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32}-1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16}-1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.
- Following (*), we can write $v \subseteq S$ as
  - $v = v^{(1)} + 2^{16}v^{(2)}$ for vectors $v^{(1)}, v^{(2)}$.
  - $v^{(i)} \subseteq S' \Leftrightarrow M_i \cdot S'^T = v^{(i)T},\ M_i \in R_{elem}$, for $i = 1,2$.
- These conditions are equivalent to $v = (M_1 \cdot S') + 2^{16}(M_2 \cdot S')$ and $M_1, M_2 \in R_{elem}$

FLI and SOS decomposition

- Let's take $S = \{0,1,\ldots,2^{32}-1\}$
- An element x belongs to S if and only if $x = x_1 + 2^{16}x_2$ (*)

  And $x_i \in S'$ for all i, where $S' = \{0,\ldots,2^{16}-1\}$.
- Say we want to prove $v \subseteq S$. Equivalenty that $M \cdot S^T = v^T$ for some $M \in R_{elem}$.
- Following (*), we can write $v \subseteq S$ as
  - $v = v^{(1)} + 2^{16}v^{(2)}$ for vectors $v^{(1)}, v^{(2)}$.
  - $v^{(i)} \subseteq S' \Leftrightarrow M_i \cdot S'^T = v^{(i)T}$, $M_i \in R_{elem}$, for $i = 1,2$.
- These conditions are equivalent to $v = (M_1 \cdot S') + 2^{16}(M_2 \cdot S')$ and $M_1, M_2 \in R_{elem}$
- We now use a Hypernova-style sumcheck to reduce the equality to two linear equalities, plus $M_1, M_2 \in R_{elem}$. Then we perform folding similarly as before.

FLI and other folding schemes

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.
- However, when doing many foldings, it works with a dense $|v| \cdot |S|$ witness.

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.
- However, when doing many foldings, it works with a dense $|v| \cdot |S|$ witness.
- If S is SOS decomposable, roughly:

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.
- However, when doing many foldings, it works with a dense $|v| \cdot |S|$ witness.
- If S is SOS decomposable, roughly:
  - We turn each folding step into c folding steps.

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.
- However, when doing many foldings, it works with a dense $|v| \cdot |S|$ witness.
- If S is SOS decomposable, roughly:
  - We turn each folding step into c folding steps.
  - Then when proving a folded instance, we work with $|v| \cdot |S|^{1/c}$-sized witness.

FLI and other folding schemes

- Overall, FLI has the cheapest folding Prover and Verfier we are aware of.
- However, when doing many foldings, it works with a dense $|v| \cdot |S|$ witness.
- If S is SOS decomposable, roughly:
  - We turn each folding step into c folding steps.
  - Then when proving a folded instance, we work with $|v| \cdot |S|^{1/c}$-sized witness.
- FLI can leverage SOS decomposability of S with much less field operations and commitments than other schemes: Protostar (Bünz, Biny Chen, 2023) , Proofs for Deep Thougth (Bünz, Jessica Chen, 2024), NeutronNova (Kothapally, Setty, 2024)

New trends in the zk world: SOS decomposition

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.
- Namely, those capturing the RISC-V instructions.

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.
- Namely, those capturing the RISC-V instructions.
- They propose building a zkVM that proves computations using the Lasso lookup, exploiting the SOS-decomposability of the RISC-V tables.

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.
- Namely, those capturing the RISC-V instructions.
- They propose building a zkVM that proves computations using the Lasso lookup, exploiting the SOS-decomposability of the RISC-V tables.
- Roughly, a RISC-V table has the form

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.
- Namely, those capturing the RISC-V instructions.
- They propose building a zkVM that proves computations using the Lasso lookup, exploiting the SOS-decomposability of the RISC-V tables.
- Roughly, a RISC-V table has the form

$$S = \{(x||y||z) \mid (x, y) \text{ input to an instruction, } z \text{ output}\}$$

New trends in the zk world: SOS decomposition

- In Jolt (Arun, Setty, Thaler, 2023) it is observed that many natural sets are SOS decomposable.
- Namely, those capturing the RISC-V instructions.
- They propose building a zkVM that proves computations using the Lasso lookup, exploiting the SOS-decomposability of the RISC-V tables.
- Roughly, a RISC-V table has the form

$$S = \{(x||y||z) \mid (x, y) \text{ input to an instruction, } z \text{ output}\}$$

- Example: An instruction could be bitwise XOR of 64-bit strings. Then $|S| = 2^{3\cdot 64} = 2^{192}$.

Handling very large computations in Jolt ⚡

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.
- Because of this, the Jolt team proposes to:

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.
- Because of this, the Jolt team proposes to:

  1. Split the lookup into, say, 2^5 lookups $v_1 \subseteq S, \ldots, v_{2^5} \subseteq S$, where $|v_i| = 2^{19}$

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.
- Because of this, the Jolt team proposes to:
  1. Split the lookup into, say, 2^5 lookups $v_1 \subseteq S, \ldots, v_{2^5} \subseteq S$, where $|v_i| = 2^{19}$
  2. Either:

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.
- Because of this, the Jolt team proposes to:
  1. Split the lookup into, say, 2^5 lookups $v_1 \subseteq S, \ldots, v_{2^5} \subseteq S$, where $|v_i| = 2^{19}$
  2. Either:
     - Prove each lookup and then create a recursive proof.

Handling very large computations in Jolt ⚡

- Jolt targets proving computations with 2^{24} instructions.
- This means Jolt must prove a lookup instance $v \subseteq S$ where S is gigantic (concatenation of all instruction tables) and SOS decomposable.
- It's currently unfeasible to do that in a single shot, due to memory constraints.
- Because of this, the Jolt team proposes to:
    1. Split the lookup into, say, 2^5 lookups $v_1 \subseteq S, \ldots, v_{2^5} \subseteq S$, where $|v_i| = 2^{19}$
    2. Either:
        - Prove each lookup and then create a recursive proof.
        - Fold the 2^5 lookups and then prove the folded claim.