Efficient Interactive Proofs for Non-Deterministic Bounded Space

Joshua Cook (UT Austin) and
Ron Rothblum (Technion)

Background

Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

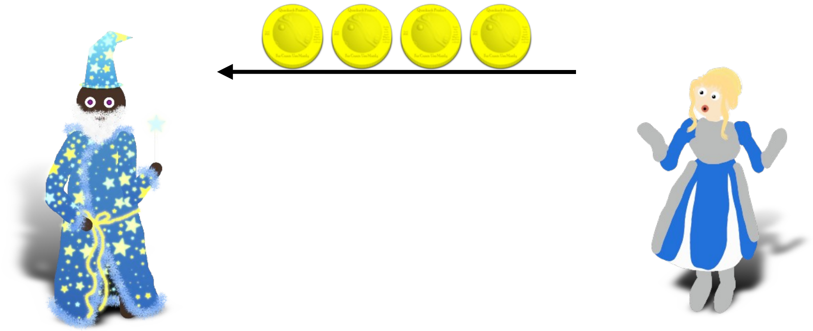
Randomized Verifier (Arthur)

Many Questions

Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

Randomized Verifier (Arthur)

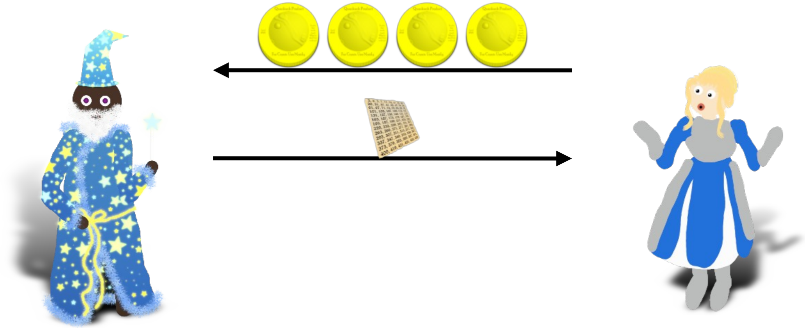


Many Questions

Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

Randomized Verifier (Arthur)

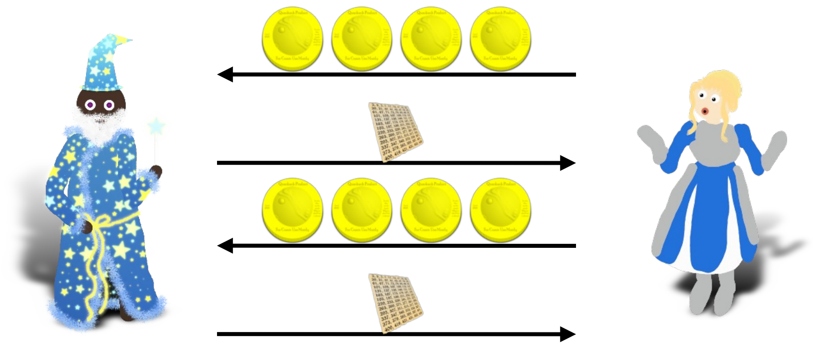


Many Questions

Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

Randomized Verifier (Arthur)



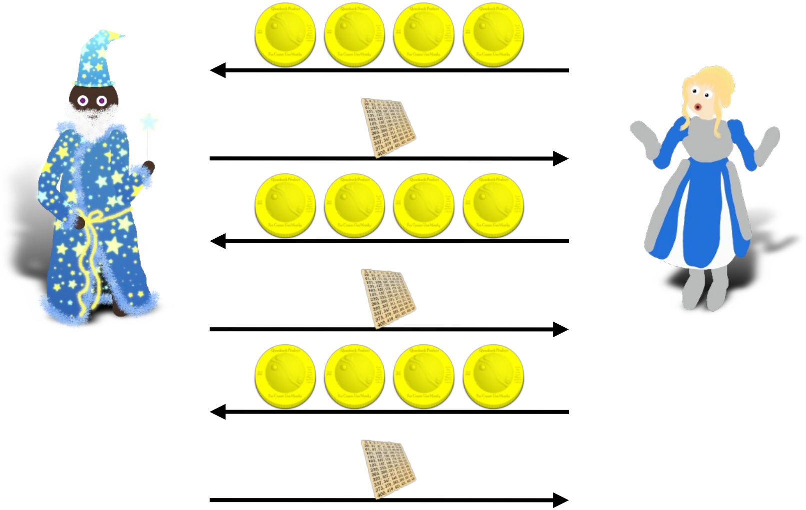
Many Questions

Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

Randomized Verifier (Arthur)

Many Questions

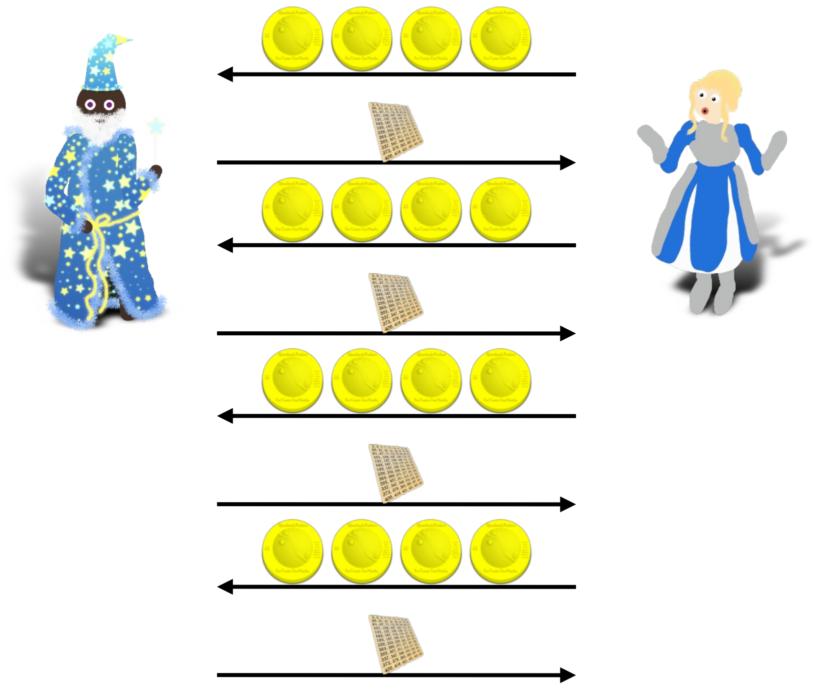


Interactive Proofs (IPs)?

Untrusted Prover (Merlin)

Randomized Verifier (Arthur)

Many Questions



Goal

- Perfect Completeness: an honest Prover always succeeds
- Statistical Soundness: any Prover is unlikely to trick Verifier
- Primary goal: minimize Verifier time.
- Secondary goal: minimize Prover time, Verifier space, generalize to alternating algorithms

Goal

- Perfect Completeness: an honest Prover always succeeds
- Statistical Soundness: any Prover is unlikely to trick Verifier
- Primary goal: minimize Verifier time.
- Secondary goal: minimize Prover time, Verifier space,
  generalize to alternating algorithms

Other interactive protocols have much faster provers **[RRR16]**.

IPs for Deterministic Algorithms

TISP[T, S]: time T and space S algorithms.

ITIME[T]: IP with T Verifier time.

[LFKN90, Shamir90, GKR08, HMS13, Thaler20, Cook22]

$$\mathrm{TISP}[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S \log(T))]$$

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

IP verifiers are programs with nondeterminism AND randomness.

- Using Nisan's [Nisan90] PRG, IPs speed up randomized algorithms as much as known for deterministic [Cook22].
- Should IPs speed up nondeterministic algorithms as much?

Randomness Time T, Space S

Nondeterminism
Time T, Space S

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

IP verifiers are programs with nondeterminism AND randomness.

- Using Nisan's [Nisan90] PRG, IPs speed up randomized algorithms as much as known for deterministic [Cook22].
- Should IPs speed up nondeterministic algorithms as much?

Randomness Time T, Space S + Nondeterminism

Nondeterminism
Time T, Space S

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

IP verifiers are programs with nondeterminism AND randomness.

- Using Nisan's [Nisan90] PRG, IPs speed up randomized algorithms as much as known for deterministic [Cook22].
- Should IPs speed up nondeterministic algorithms as much?

Randomness Time T, Space S + Nondeterminism → Randomness Nondeterminism Time $\tilde{O}(S\log(T))$, Space $\tilde{O}(S)$

Nondeterminism
Time T, Space S

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

IP verifiers are programs with nondeterminism AND randomness.

- Using Nisan's [Nisan90] PRG, IPs speed up randomized algorithms as much as known for deterministic [Cook22].
- Should IPs speed up nondeterministic algorithms as much?

Randomness Time T, Space S + Nondeterminism → Randomness Nondeterminism Time Õ(Slog(T)), Space Õ(S)

Nondeterminism Time T, Space S + Randomness

Why IPs for nondeterministic algorithms?

Natural generalization, extremely well studied.

IP verifiers are programs with nondeterminism AND randomness.

- Using Nisan's [Nisan90] PRG, IPs speed up randomized algorithms as much as known for deterministic [Cook22].
- Should IPs speed up nondeterministic algorithms as much?

Randomness Time T, Space S	+	Nondeterminism	→	Randomness Nondeterminism Time Õ(Slog(T)), Space Õ(S)
Nondeterminism Time T, Space S	+	Randomness	→	Randomness Nondeterminism Time ?, Space ?

Results

Result

Prior verifier for nondeterministic algorithms [Cook22]:

$$\mathrm{NTISP}[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S \log(T)^2)]$$

Our results:

$$\mathrm{NTISP}[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S \log(T))]$$

***Our verifiers also have space Õ(S)**

IPs for NTISP as fast as known for TISP, up to a log(S) factor.

Nondeterminism Time T, Space S	+	Randomness	→	Randomness Nondeterminism Time $S\log(T)^2$, Space $S\log(T)$

Result

Prior verifier for nondeterministic algorithms [Cook22]:

$$\mathrm{NTISP}[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S \log(T)^2)]$$

Our results:

$$\mathrm{NTISP}[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S \log(T))]$$

***Our verifiers also have space $\tilde{O}(S)$**

IPs for NTISP as fast as known for TISP, up to a log(S) factor.

Nondeterminism
Time T, Space S + Randomness → Randomness Nondeterminism
Time $\tilde{O}(S\log(T))$, Space $\tilde{O}(S)$

Generalization

Generalizes to alternating algorithms with few alternations.

$$\mathrm{ATISP}_d[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S\log(T) + Sd)]$$

Generalization

Generalizes to alternating algorithms with few alternations.

$$\mathrm{ATISP}_d[T, S] \subseteq \mathrm{ITIME}[\tilde{O}(n + S\log(T) + Sd)]$$

- Closely related to depth d circuits **[Ruz81]**.
- Non-trivial to reduce to Alternating algorithms to nondeterministic ones.

Techniques for Prior Results

Computation Graph

View space S program as

2^S state graph, G

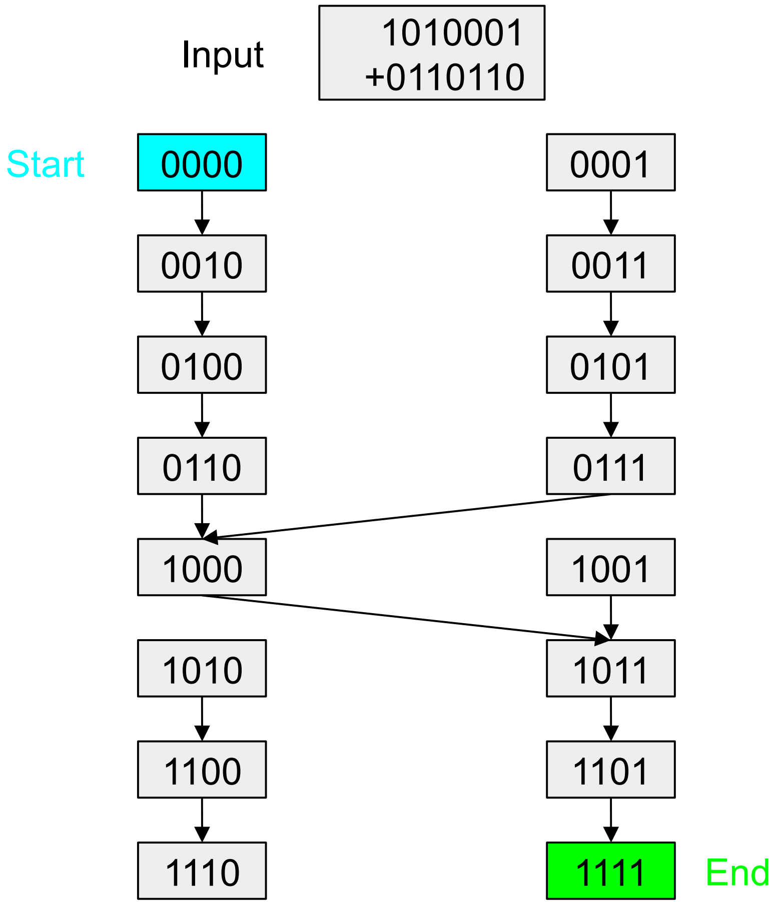
Edges are state transitions

Graph is a function of

Input, Program

Accepts IFF there is a length T path from start to end.

Edges are fast to compute



Adjacency Matrix For Deterministic Algorithms

$$M = \begin{bmatrix} & & 1 & & & & & \\ 1 & & & & & & & \\ & & & & & 1 & & \\ & & & & 1 & & & \\ & & & 1 & & & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix}$$

Represent G as an adjacency, M

Algorithm accepts in time T iff

$$M^T_{\text{start, end}} = 1$$

For deterministic algorithms, for all i, M^i is a binary matrix.

With sum check [Thaler14, HMS13]
can reduce $\mathbf{M}^2(x_0, x_1) = \alpha$

to $\mathbf{M}(y_0, y_1) = \beta$

In time $\tilde{O}(S)$.

Adjacency Matrix For Deterministic Algorithms

$$M = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & 1 & & & & & \\ \hline 1 & & & & & & & \\ \hline & & & & & 1 & & \\ \hline & & & & 1 & & & \\ \hline & & & 1 & & & & \\ \hline & & & & & & 1 & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array} \qquad M^2 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & 1 & & \\ \hline & & 1 & & & & & \\ \hline & & & & & & 1 & \\ \hline & & & 1 & & & & \\ \hline & & & & 1 & & & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array}$$

Represent G as an adjacency, M

Algorithm accepts in time T iff

$$M^T_{\text{start, end}} = 1$$

For deterministic algorithms, for all i,
M^i is a binary matrix.

With sum check [Thaler14, HMS13]
can reduce $\mathbf{M}^2(x_0, x_1) = \alpha$

to $\mathbf{M}(y_0,y_1) = \beta$

In time $\tilde{O}(S)$.

Adjacency Matrix For Deterministic Algorithms

Represent G as an adjacency, M

Algorithm accepts in time T iff

$$M^{T}_{\text{start, end}} = 1$$

For deterministic algorithms, for all i, M^i is a binary matrix.

With sum check [Thaler14, HMS13] can reduce $\mathbf{M}^2(x_0, x_1) = \alpha$

to $\mathbf{M}(y_0, y_1) = \beta$

In time $\tilde{O}(S)$.

$$M = \begin{bmatrix} & & 1 & & & & & \\ 1 & & & & & & & \\ & & & & & 1 & & \\ & & & & 1 & & & \\ & & & 1 & & & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} & & & & & 1 & & \\ & & 1 & & & & & \\ & & & & & & 1 & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix}$$

$$M^4 = \begin{bmatrix} & & & & & & & 1 \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix}$$

Adjacency Matrix For Deterministic Algorithms

Represent G as an adjacency, M

Algorithm accepts in time T iff

$$M^T_{\text{start, end}} = 1$$

For deterministic algorithms, for all i, M^i is a binary matrix.

With sum check [Thaler14, HMS13] can reduce $\mathbf{M}^2(x_0, x_1) = \alpha$

to $\mathbf{M}(y_0, y_1) = \beta$

In time Õ(S).

$$M = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & 1 & & & & & \\ \hline 1 & & & & & & & \\ \hline & & & & & 1 & & \\ \hline & & & & 1 & & & \\ \hline & & & 1 & & & & \\ \hline & & & & & & 1 & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array}
\qquad
M^2 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & 1 & & \\ \hline & & 1 & & & & & \\ \hline & & & & & & 1 & \\ \hline & & & 1 & & & & \\ \hline & & & & 1 & & & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array}$$

$$M^4 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & & & 1 \\ \hline & & & & & & 1 & \\ \hline & & & & & & & 1 \\ \hline & & & 1 & & & & \\ \hline & & & & 1 & & & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array}
\qquad
M^8 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & 1 & & & & \\ \hline & & & & 1 & & & \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline & & & & & & & 1 \\ \hline \end{array}$$

Nondeterministic Algorithm Matrix

$$M = \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
 & 1 & 1 & & & & & \\ \hline
 & & 1 & 1 & & & & \\ \hline
 & & & 1 & 1 & & & \\ \hline
 & & & 1 & & & & \\ \hline
 & & & & & 1 & 1 & \\ \hline
 & & & & & & 1 & 1 \\ \hline
 & & & 1 & & & & 1 \\ \hline
 & & & & & & & 1 \\ \hline
\end{array}$$

For nondeterministic algorithms:

$M^T_{\text{start, end}}$ = #valid proofs

Matrix entries could be very large.

Random field size can help, but gives log(T) overhead of [Cook22].

Can’t use sums, need to use ORs.

Nondeterministic Algorithm Matrix

For nondeterministic algorithms:

$M^T_{\text{start, end}}$ = #valid proofs

Matrix entries could be very large.

Random field size can help, but gives log(T) overhead of [Cook22].

Can’t use sums, need to use ORs.

M =

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

M^2 =

		1	2	1			
			2	1			
			1		1	1	
			1				
			1			1	2
			1				2
			1				1
							1

Nondeterministic Algorithm Matrix

For nondeterministic algorithms:

$M^T_{\text{start, end}}$ = #valid proofs

Matrix entries could be very large.

Random field size can help, but gives log(T) overhead of [Cook22].

Can't use sums, need to use ORs.

M =

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

M^2 =

		1	2	1			
			2	1			
			1		1	1	
			1				
			1			1	2
			1				2
			1				1
							1

M^4 =

			4		1	2	2
			3			1	2
			3				3
			1				
			2				3
			1				2
			1				1
							1

Nondeterministic Algorithm Matrix

For nondeterministic algorithms:

$M^T_{\text{start, end}}$ = #valid proofs

Matrix entries could be very large.

Random field size can help, but gives log(T) overhead of [Cook22].

Can't use sums, need to use ORs.

$M =$

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

$M^2 =$

		1	2	1			
			2	1			
			1		1	1	
			1				
			1			1	2
			1				2
			1				1
							1

$M^4 =$

			4		1	2	2
			3			1	2
			3				3
			1				
			2				3
			1				2
			1				1
							1

$M^8 =$

			7				6
			4				3
			3				3
			1				
			2				3
			1				2
			1				1
							1

Boolean Formula for $M^{(2)}$

$$M = \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
 & 1 & 1 & & & & & \\ \hline
 & & 1 & 1 & & & & \\ \hline
 & & & 1 & 1 & & & \\ \hline
 & & & 1 & & & & \\ \hline
 & & & & & 1 & 1 & \\ \hline
 & & & & & & 1 & 1 \\ \hline
 & & & 1 & & & & 1 \\ \hline
 & & & & & & & 1 \\ \hline
\end{array}$$

For notation, let $M(u,v) = M_{u,v}$

$$M^2(u,v) = \sum_{w\in\{0,1\}^S} M(u,w)\, M(w,v)$$

Replace + with $\vee$

$$M^{(2)}(u,v)=\bigvee_{w\in\{0,1\}^S} M(u,\, w)M(w,\, v)$$

Boolean Formula for $M^{(2)}$

$M =$

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

$M^{(2)} =$

		1	1	1			
			1	1			
			1		1	1	
			1				
			1			1	1
			1				1
			1				1
							1

For notation, let $M(u,v) = M_{u,v}$

$$M^2(u,v) = \sum_{w \in \{0,1\}^S} M(u,w)\, M(w,v)$$

Replace + with $\vee$

$$M^{(2)}(u,v) = \bigvee_{w \in \{0,1\}^S} M(u, w) M(w, v)$$

Boolean Formula for $M^{(2)}$

For notation, let $M(u,v) = M_{u,v}$

$$M^2(u,v) = \sum_{w \in \{0,1\}^S} M(u,w)\, M(w,v)$$

Replace + with $\vee$

$$M^{(2)}(u,v) = \vee_{w \in \{0,1\}^S} M(u, w) M(w, v)$$

$M =$

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

$M^{(2)} =$

		1	1	1			
			1	1			
			1		1	1	
			1				
			1			1	1
			1				1
			1				1
							1

$M^{(4)} =$

			1		1	1	1
			1			1	1
			1				1
			1				
			1				1
			1				1
			1				1
							1

Boolean Formula for $M^{(2)}$

For notation, let $M(u,v) = M_{u,v}$

$$M^2(u,v) = \sum_{w\in\{0,1\}^S} M(u,w)\, M(w,v)$$

Replace + with $\vee$

$$M^{(2)}(u,v)=\bigvee_{w\in\{0,1\}^S} M(u, w)M(w, v)$$

$M =$

	1	1					
		1	1				
			1	1			
			1				
					1	1	
						1	1
			1				1
							1

$M^{(2)} =$

		1	1	1			
			1	1			
			1		1	1	
			1				
			1			1	1
			1				1
			1				1
							1

$M^{(4)} =$

			1		1	1	1
			1			1	1
			1				1
			1				
			1				1
			1				1
			1				1
							1

$M^{(8)} =$

			1				1
			1				1
			1				1
			1				
			1				1
			1				1
			1				1
							1

Reduction

Let **M** be the multilinear extension of M.

Given claim that $\mathbf{M}^{(2)}(u,v) = \alpha$, want to reduce to claim that $\mathbf{M}(u',v') = \beta$

Attempt 1, use $M^{(2)}(u,v) = 1 - \prod_{w\in\{0,1\}^S}(1 - M(u, w)\, M(w, v))$.

Degree is **way too high**: 2^S

Can handle one variable of w at a time with relinearizations [She92], but takes S linearizations of S variables, requires time $\tilde{O}(S^2)$.

Degree Reduction

$$1 = \bigvee \; 0\;0\;1\;0\;1\;1\;1\;0\;1\;0\;1\;0\;0$$

Razborov Smolensky

Razborov-Smolensky reduces degree

Idea: replace $\bigvee_{i \in [k]} x_i$

with $\sum_{j \in [k]} r_j \, x_j \bmod 2$

Where r is uniform random.

- Low degree, linear in GF(2).
- If all x are 0, outputs 0.
- If any x 1, then output 1 with probability 1/2.

$$1 = \bigvee \; 0 \; 0 \; 1 \; 0 \; 1 \; 1 \; 1 \; 0 \; 1 \; 0 \; 1 \; 0 \; 0$$

$$1 = \bigoplus \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \; 0$$

Razborov Smolensky

Razborov-Smolensky reduces degree

Idea: replace $\bigvee_{i \in [k]} x_i$

with $\sum_{j \in [k]} r_j x_j \bmod 2$

Where r is uniform random.

- Low degree, linear in GF(2).
- If all x are 0, outputs 0.
- If any x 1, then output 1 with probability 1/2.

Razborov Smolensky

Razborov-Smolensky reduces degree

Idea: replace $\bigvee_{i \in [k]} x_i$

with $\sum_{j \in [k]} r_j \, x_j \bmod 2$

Where r is uniform random.

- Low degree, linear in GF(2).
- If all x are 0, outputs 0.
- If any x 1, then output 1 with probability 1/2.

```
1 = ∨ 0 0 1 0 1 1 1 0 1 0 1 0 0

1 = ⊕    0   0   1   0   0   0 0

0 = ⊕ 0   1 0 1 1     1 0     0
```

Razborov Smolensky

Razborov-Smolensky reduces degree

Idea: replace $\bigvee_{i\in[k]} x_i$

with $\sum_{j\in[k]} r_j\, x_j \bmod 2$

Where r is uniform random.

- Low degree, linear in GF(2).
- If all x are 0, outputs 0.
- If any x 1, then output 1 with probability 1/2.

1	=	$\vee$	0	0	1	0	1	1	1	0	1	0	1	0	0
1	=	$\oplus$		0		0		1		0		0		0	0
0	=	$\oplus$	0		1	0	1	1			1	0			0
0	=	$\oplus$		0		0		1	1	0	1		1	0	
1	=	$\oplus$							1	0	1	0	1	0	0
1	=	$\oplus$			1	0	1		1	0				0	0
0	=	$\oplus$		0	1		1		1	0		0	1		0

$$1 = \bigvee \; 0 \; 0 \; 1 \; 0 \; 1 \; 1 \; 1 \; 0 \; 1 \; 0 \; 1 \; 0 \; 0$$

Razborov Smolensky continued

Probability of failure decreases exponentially with repetitions.

$$\Pr_u[1-\prod_{i\in[L]}(1-\sum_{j\in[k]}r_{i,j}\, x_j)\neq\bigvee_{i\in[k]}x_i]\leq 2^{-L}$$

$m = 2^{O(S)}$ ANDs, choose $L = 2^{\ell} = \Omega(S)$. Most approximations are correct.

- Degree is small, $O(S)$
- $\ell = O(\log(S))$ variables

Razborov Smolensky continued

Probability of failure decreases exponentially with repetitions.

$$\Pr_u[1-\prod_{i\in[L]}(1-\sum_{j\in[k]} r_{i,j}\, x_j) \neq \bigvee_{i\in[k]} x_i] \leq 2^{-L}$$

$m = 2^{O(S)}$ ANDs, choose $L = 2^\ell = \Omega(S)$. Most approximations are correct.

- Degree is small, $O(S)$
- $\ell = O(\log(S))$ variables

$1 = \bigvee$ 0 0 1 0 1 1 1 0 1 0 1 0 0

$1 = \bigvee$ $\oplus$		0		0		1		0		0		0	0
$\oplus$	0		1	0	1	1			1	0			0
$\oplus$		0		0		1	1	0	1		1	0	
$\oplus$							1	0	1	0	1	0	0
$\oplus$		0	1		1		1	0		0	1		0
$\oplus$	0	0						0	1		1		

Applying Razborov Smolensky

Approximate $\bigvee_{w \in \{0,1\}^S} M(u, w)\, M(w, v)$

With $1 - \prod_{i \in [O(S)]} \left(1 - \sum_{w \in \{0,1\}^S} r_{i,w}\, M(u, w)\, M(w, v)\right)$

Applying Razborov Smolensky

Approximate $\bigvee_{w \in \{0,1\}^S} M(u, w)\, M(w, v)$

With $1 - \prod_{i \in [O(S)]} (1 - \sum_{w \in \{0,1\}^S} r_{i,w}\, M(u, w)\, M(w, v))$

Problem, too much randomness!!

Want seed length $O(S)$!

Completely random r has seed length $S\, 2^S$!!

Derandomization

Derandomization Step 1, Epsilon Biased Sets

Like [GR20], epsilon biased sets to sample r.

- Epsilon biased sets fool parity functions, Razborov-Smolensky IS a conjunction of parity functions!
- Epsilon biased sets equivalent to codes, easy to compute.

Derandomization Step 1, Epsilon Biased Sets

Like [GR20], epsilon biased sets to sample r.

- Epsilon biased sets fool parity functions, Razborov-Smolensky IS a conjunction of parity functions!
- Epsilon biased sets equivalent to codes, easy to compute.

$$1 - \prod_{i \in [O(S)]} (1 - \sum_{w \in \{0,1\}^S} D(seed_i, w)\, M(u, w)\, M(w, v))$$

Derandomization Step 1, Epsilon Biased Sets

Like [GR20], epsilon biased sets to sample r.

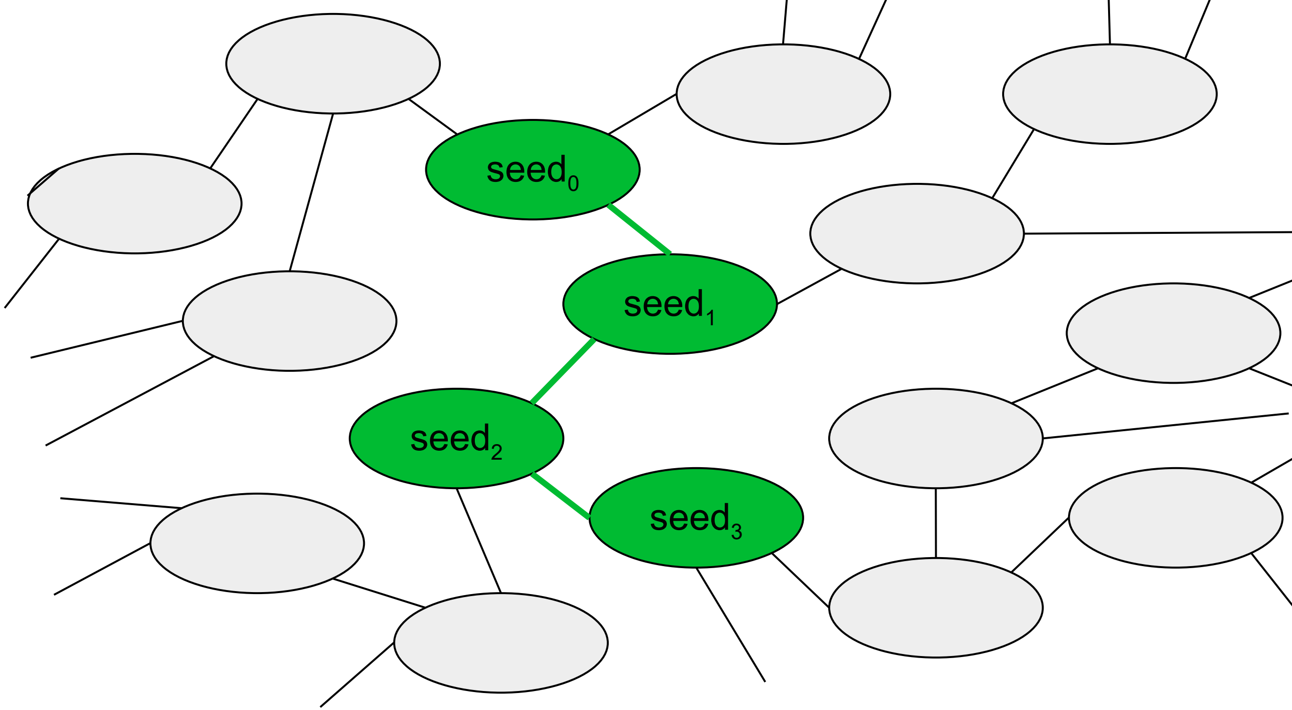
- Epsilon biased sets fool parity functions, Razborov-Smolensky IS a conjunction of parity functions!
- Epsilon biased sets equivalent to codes, easy to compute.

$$1 - \prod_{i \in [O(S)]} \left(1 - \sum_{w \in \{0,1\}^S} D(seed_i, w)\, M(u, w)\, M(w, v)\right)$$

- Takes O(S) bits for one seed for one choice of i: $seed_i$.
- But O(S) parities, $O(S^2)$ bits to sample them independently. Too much!

Derandomization Step 2, Random Walk Set Sampling

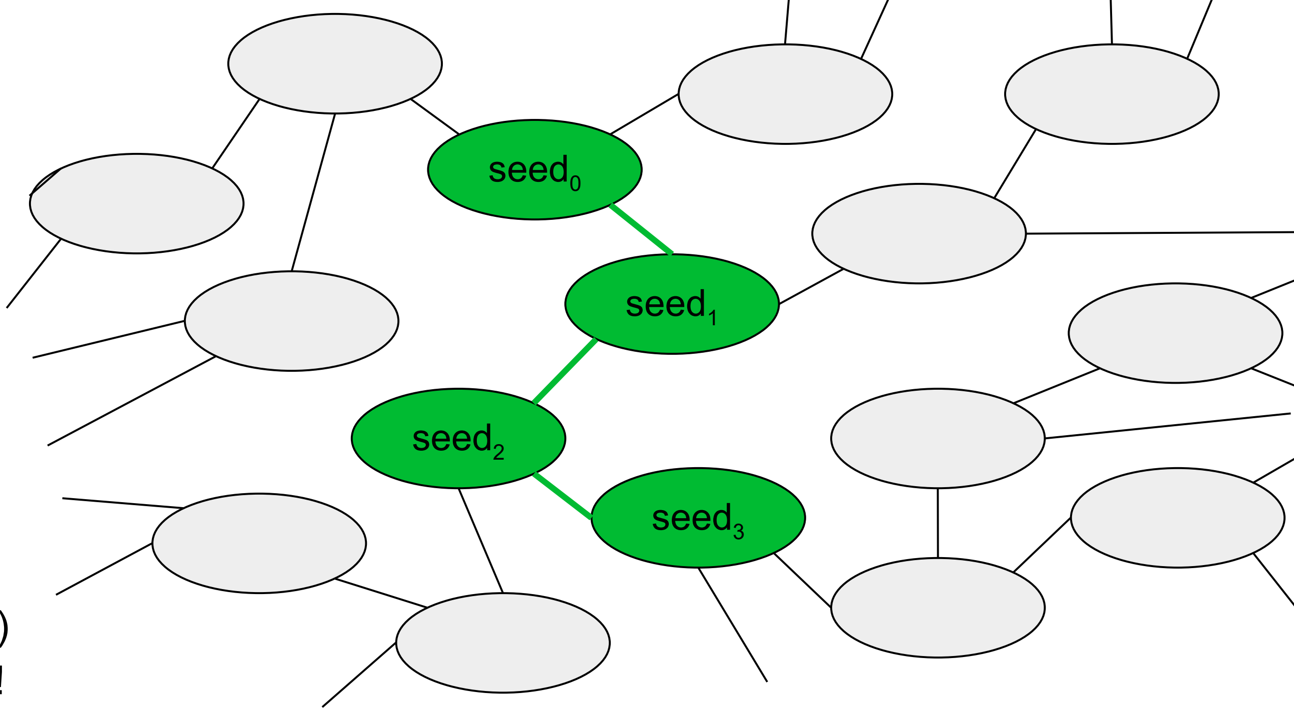
Unlike [GR20], select O(S) seeds using a length O(S) walk on expander graph.



Derandomization Step 2, Random Walk Set Sampling

Unlike [GR20], select O(S) seeds using a length O(S) walk on expander graph.

- Only takes seed length O(S) to sample length O(S) walk on length O(S) seeds!
- Walk is efficient, epsilon biased set is efficient, D is efficient.



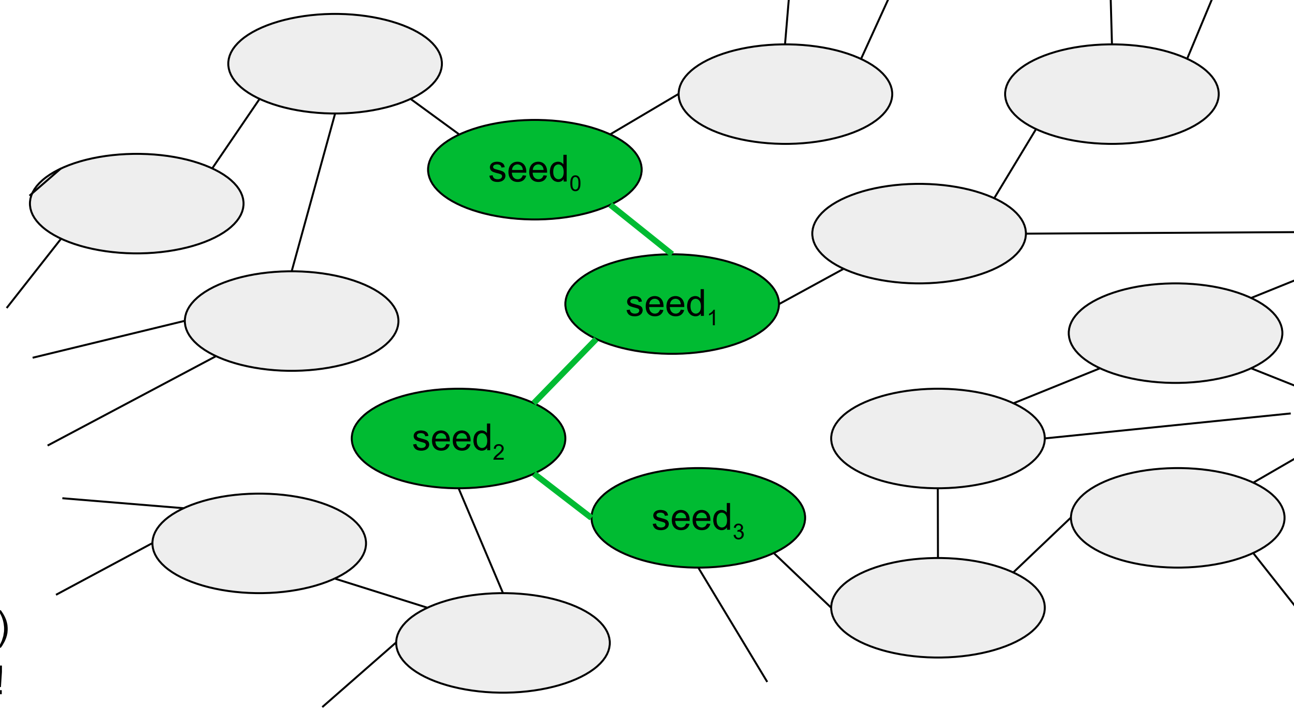
$$walk = (seed_0, edge_1, edge_2, edge_3)$$

$$|walk| = O(S) + O(1) + O(1) + O(1)$$

Derandomization Step 2, Random Walk Set Sampling

Unlike [GR20], select O(S) seeds using a length O(S) walk on expander graph.

- Only takes seed length O(S) to sample length O(S) walk on length O(S) seeds!
- Walk is efficient, epsilon biased set is efficient, D is efficient.



$$\text{walk} = (\text{seed}_0, \text{edge}_1, \text{edge}_2, \text{edge}_3)$$

$$|\text{walk}| = O(S) + O(1) + O(1) + O(1)$$

$$1 - \prod_{i \in [O(S)]} \left(1 - \sum_{w \in \{0,1\}^S} D(\text{seed}_i, w)\, M(u, w)\, M(w, v)\right)$$

Main Reduction

$$1 - \prod_{i\in\{0,1\}^\ell} (1 - \sum_{w\in\{0,1\}^S} D(walk_i, w)\ M(u, w)\ M(w, v))$$

- Represent i as binary, so it has $\ell = O(\log(S))$ bits.
- Remove one variable in i at a time.
  - For each variable need to do relinearization: $S\ \tilde{O}(\log(S))$
  - Need to do a product reduction: $\tilde{O}(\log(S))$
- Total time: $S \log(S)\ \tilde{O}(\log(S))$

Left with a claim about the multilinear extension of $D \circ walk$ and **M**.

Final points

- Need to run reduction log(T) times and compute multilinear extension of computation graph. Total time:

$$O(n + S \log(T) \log(S))\tilde{O}(\log(S))$$

- Use IP for deterministic algorithms to verify claim about $D \circ$ walk.
- If seeds fail, can prove they fail.
- Same idea works for unbounded fan-in circuits.
  - Faster than GKR for very large fan in.
  - Less optimized prover.

Contrast with GR20

Similar:

- Optimized for unbounded fan-in circuits.
- Uses Razborov Smolensky.
- Uses same epsilon biased sets.
- Achieve perfect completeness when seed is bad in same way.

Different:

- Optimized for time, instead of rounds.
    - More rounds to do degree reduction.
    - Lower degree polynomials.
    - Only constant number of claims at once.
- Requires further derandomization using random walks on expanders.
- Uses fast Interactive Provers for deterministic algorithms instead of direct arithmetization.

Open Problems

- Better Doubly Efficient Interactive Proofs (fast provers and verifiers).
- Extend Results to Threshold Circuits (Our results give fast verifiers for $AC[\oplus]$ circuits).
- Give protocols for randomized algorithms with simultaneous:
  - Same verifier time
  - Perfect completeness
  - $2^{O(S)}$ prover time (as opposed to $2^{O(S\log(T))}$).

Citations

[Cook22] Joshua Cook. "More Verifier Efficient Interactive Protocols for Bounded Space". In: 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022).

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: J. ACM 62.4 (Sept. 2015).

[GR20] Oded Goldreich and Guy N. Rothblum. "Constant-Round Interactive Proof Systems for AC0[2] and NC1". In: Computational Complexity and Property Testing: On the Interplay Between Randomness and Computation.

[HMS13] Edward Hirsch, Dieter van Melkebeek, and Alexander Smal. "Succinct Interactive Proofs for Quantified Boolean Formulas, Comment 2". In: Electronic Colloquium on Computational Complexity (ECCC), 2013.

[Sha92] Adi Shamir. "IP = PSPACE". In: J. ACM 39.4 (Oct. 1992)

[She92] A. Shen. "IP = SPACE: Simplified Proof". In: J. ACM 39.4 (1992)

[Tha20] Justin Thaler. The Unreasonable Power of the Sum-Check Protocol. 2020

[Nis90] Noam Nisan. "Pseudorandom Generators for Space-Bounded Computations". In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing. STOC '90.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing. STOC '16.

[Ruz81] Walter L. Ruzzo. "On uniform circuit complexity". In: Journal of Computer and System Sciences 22.3 (1981)