Split Gröbner Bases

for Satisfiability Modulo Finite Fields

Alex Ozdemir[1,2], Shankara Pailoor[2,3], Alp Bassa[2],

Kostas Ferles[2], Clark Barrett[1], Işil Dillig[2,3]

$\mathbb{Z}_p$

[1]Stanford, Center for Automated Reasoning [2]Veridise Inc [3]UT Austin

$\mathbb{F}$-based cryptosytems are critical

They are everywhere:

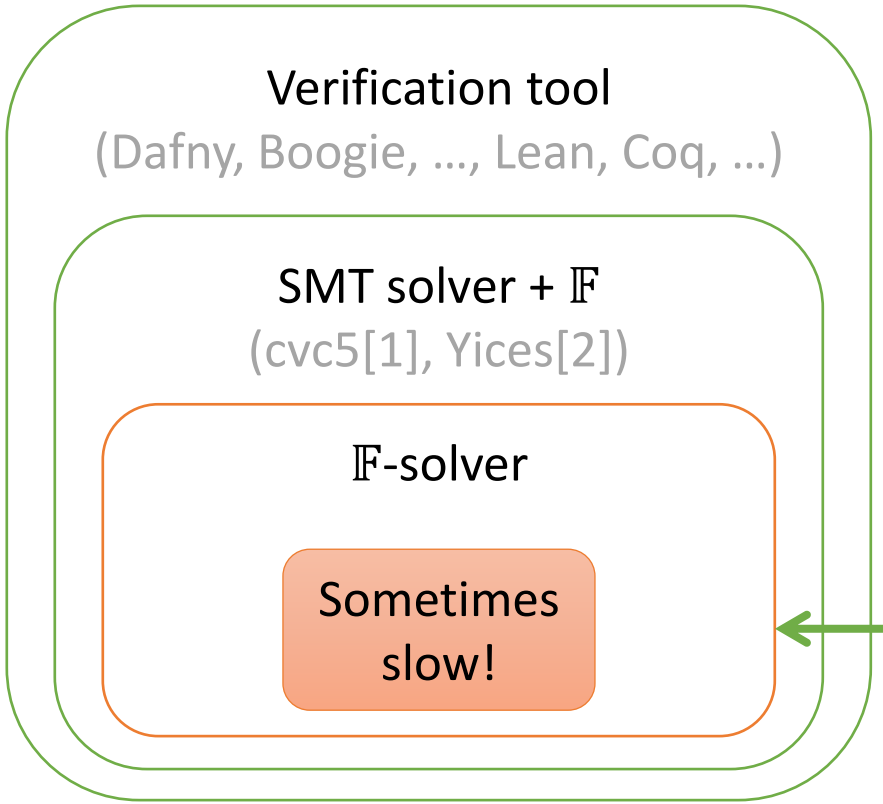
- key exchange (DHE, ECDHE, …)
- signatures (EdDSA, ECDSA, …)
- authentication (Poly1305,…)
- private cryptocurrency (Zcash, …)
- secure auctions (SPDZ, …)

Their bugs are everywhere:

Strategy: eliminate bugs through **verification**

See also: [Erbsen et al.; IEEE S&P'19] [0xPARC "ZK Bug Tracker"], [Chaliasos et al.; '24] …

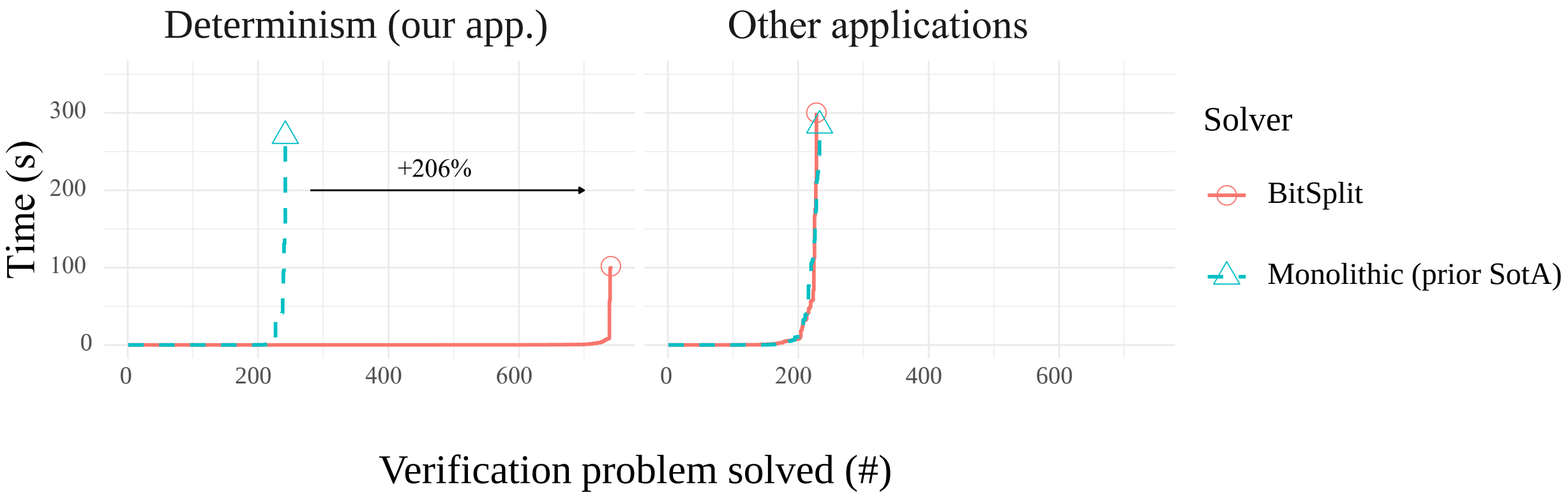
𝔽-based verification can be slow



We build **BitSplit**, which is **far faster** for **verifying determinism** (an important application).

[1]: Ozdemir et al., [2]: Hader et al.

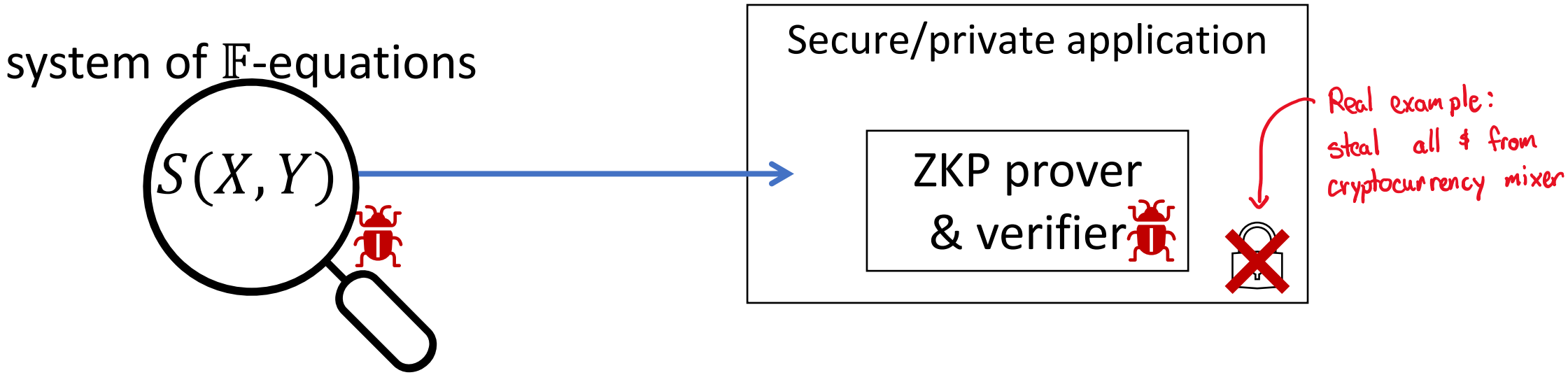
BitSplit: far better at determinism



Our work: Better $\mathbb{F}$-solving for a key application

Application	Problem	Solution
		
determinism for zero-knowledge proofs (ZKPs)	bit-splitting	split Gröbner bases with propagation

Verify what? *Determinism*



auditor

Q: "is S deterministic?"

"is S a (partial) function $X \mapsto Y$?"

$$\forall x, y, y', \big(S(x, y) \ \wedge \ S(x, y')\big) \Rightarrow (y = y')$$

Determinism rules out 95% of bugs[1]

[1] "Understanding Security Vulnerabilities in zkSNARKs," Chaliasos et al., 2024

Which systems? *Ones with bit-splitting*

bit split (with b bits)

bitsum *bit constraints*

$$\left[X = \sum_{i=1}^{b} Y_i 2^{i-1}\right] \wedge [Y_1(Y_1 - 1) = 0] \wedge \cdots \wedge [Y_b(Y_b - 1) = 0] \wedge \cdots$$

Note: these are equations in $\mathbb{F}$, not $\mathbb{Z}_{2^n}$.

98% of circom projects have bit splits

64% of SHA2 variables are in a bit split

Bit splitting is hard for a GB-based solver

$$BS(X, \boldsymbol{Y}) = \left[X = \sum_{i=1}^{b} Y_i 2^{i-1}\right] \wedge [Y_1(Y_1 - 1) = 0] \wedge \cdots \wedge [Y_b(Y_b - 1) = 0]$$

as determinism query for $X \mapsto Y_b'$

$$S = BS(X, \boldsymbol{Y}) \wedge BS(X, \boldsymbol{Y}') \wedge Y_b \neq Y_b'$$

two copies

Prior $\mathbb{F}$-solver (Monolithic)

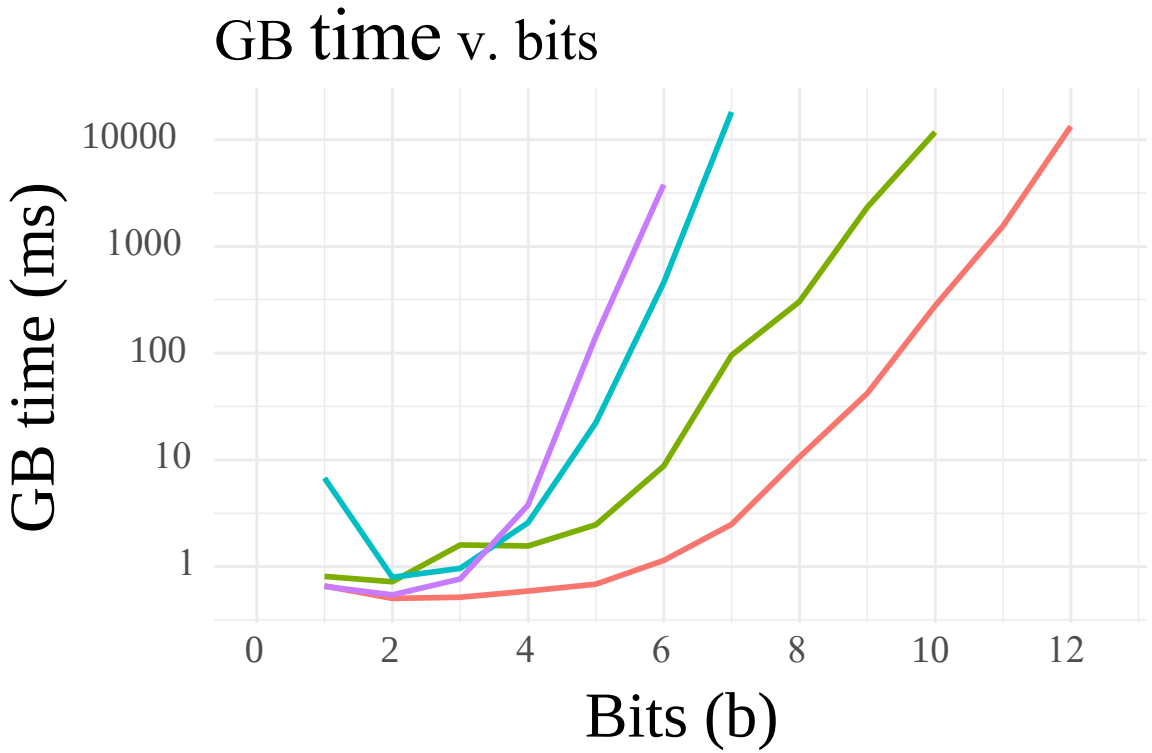
S: system of equations

$\mathbb{F}$-solver

1. $B \leftarrow GB(S)$

$GB(S)$

The root of the problem



System S

- $BS(X, \boldsymbol{Y}) \wedge BS(X, \boldsymbol{Y}') \wedge Y_b \neq Y_b'$
- $BS(X, \boldsymbol{Y}) \wedge BS(X, \boldsymbol{Y}')$
- $BS(X, \boldsymbol{Y})$
- $BS(X, \boldsymbol{Y}) \wedge X = 1$

just bitsums → easy!

just bit constraints → easy!

Goals

Goal: a decision procedure that:

1. is efficient for:
   - proving determinism
   - systems with bit-splits (**bitsums** and **bit constraints**)
2. uses a GB engine as a black-box
3. is sound and complete

Our decision procedures: “**Split**” and “**BitSplit**”

Designing (Bit)Split

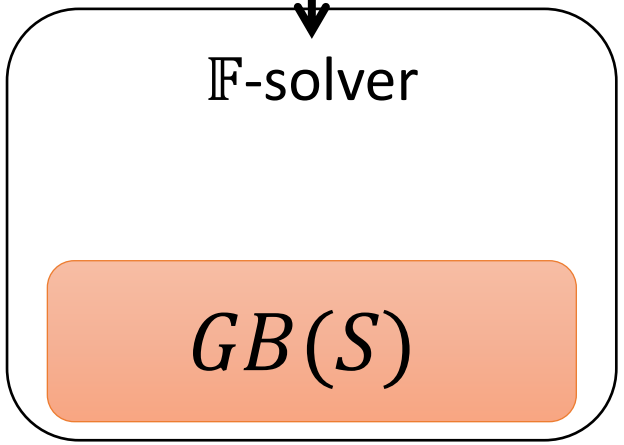
Two problems to solve:

- Can’t compute GBs for bit-splits 
- Need ***some*** bit-split reasoning 

Designing "**Split**"

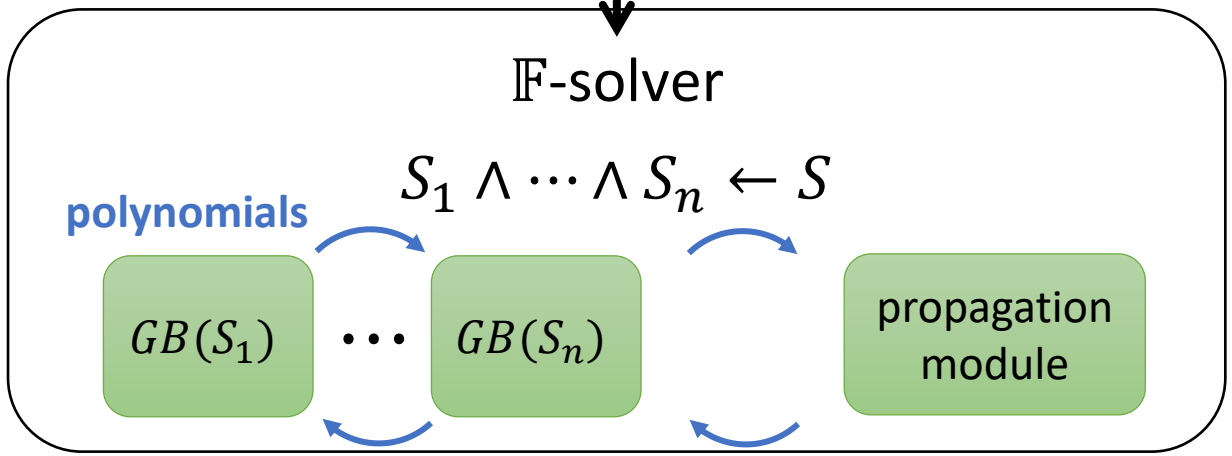
Prior $\mathbb{F}$-solver (Monolithic)

S: system of equations



Our $\mathbb{F}$-solver (Split)

S: system of equations



avoids full GB

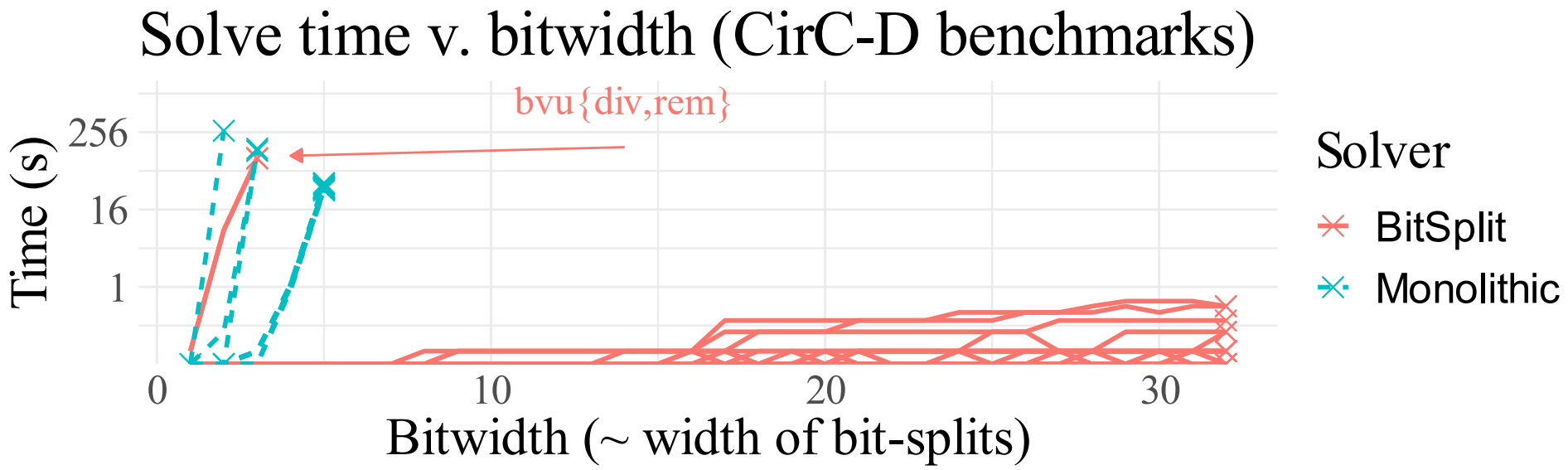
uses a *split GB*

IOIO understands
IOIO bit-splits

Instantiation: “**BitSplit**” solver

- Two bases
  - Linear (B_1):
    - initialized with linear polynomials
    - accepts linear lemmas
  - Sparse (B_2):
    - initialized with non-bitsums
    - admits equality lemmas ($X_1 = X_2$ or $X_1 = c$)
- Propagation module
  - Bit representation congruence
  - Given: $BS(X, \boldsymbol{Y}) \wedge BS(X', \boldsymbol{Y}') \wedge X = X'$
  - Deduce: $Y_1 = Y_1' \wedge \cdots \wedge Y_n = Y_n'$

BitSplit scales better with bitwidth

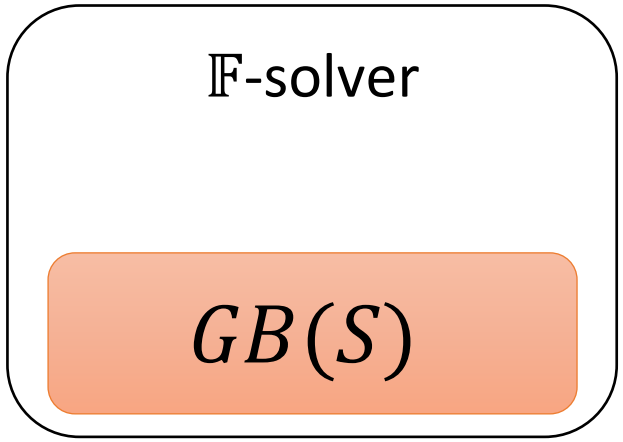


In the paper…

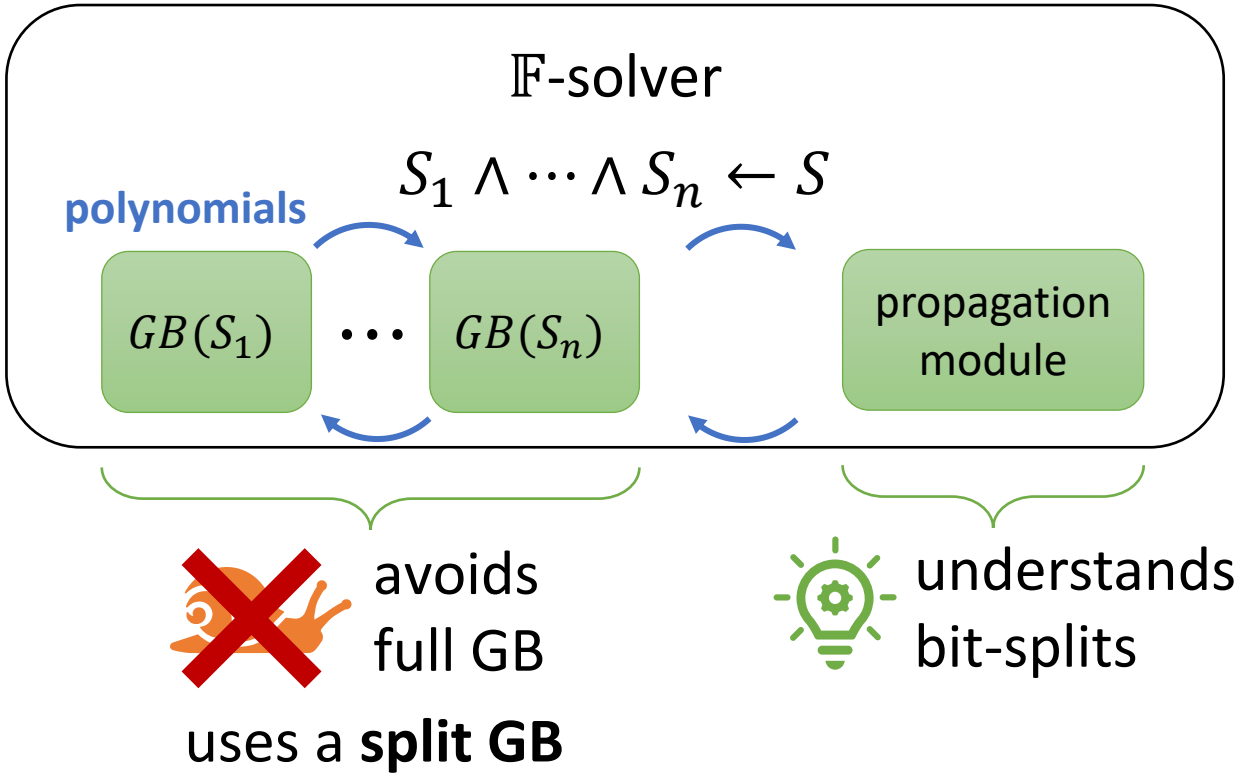
- decision procedures:
  - **Split**: for a generic split GB & propagator
    - Key technical contribution: generalizing GB-based model construction to split GBs.
  - **BitSplit**: specific # of GBs, propagator
- a study of the completeness of lemma propagation
- ablation studies
- case study: $\mathbb{F}$-blaster verification
  - determinism ⇒ full correctness

Split Gröbner Bases for Satisfiability Modulo Finite Fields

Prior $\mathbb{F}$-solver (Monolithic)



Our $\mathbb{F}$-solver (BitSplit)



Appendices

Background & terminology

- $\mathbb{F}[\boldsymbol{X}]$: all polynomials, coefficients $\mathbb{F}$, variables $\boldsymbol{X}$
- S, G, B: sets of polynomials
- $\mathcal{V}(S)$: the common zeros ($\{M \in \mathbb{F}^m : \forall s_i \in S,\ s_i(M) = 0\}$)
- $\langle S \rangle = \sum_i c_i s_i$ with $c_i \in \mathbb{F}[\boldsymbol{X}]$
  - Intuition: all polynomial (equations) implied by S
- Hilbert's Nullstellensatz
  - $1 \in \langle S \rangle \ \Leftrightarrow \ \mathcal{V}(S) = \emptyset$
  - **False** for finite fields, but assume it's true for the talk; see paper
- Gröbner Bases
  - Computed by $B \leftarrow GB(S)$; with $\langle B \rangle = \langle S \rangle$; not polytime
  - Efficient test for $1 \in \langle B \rangle$

Definition: split Gröbner basis

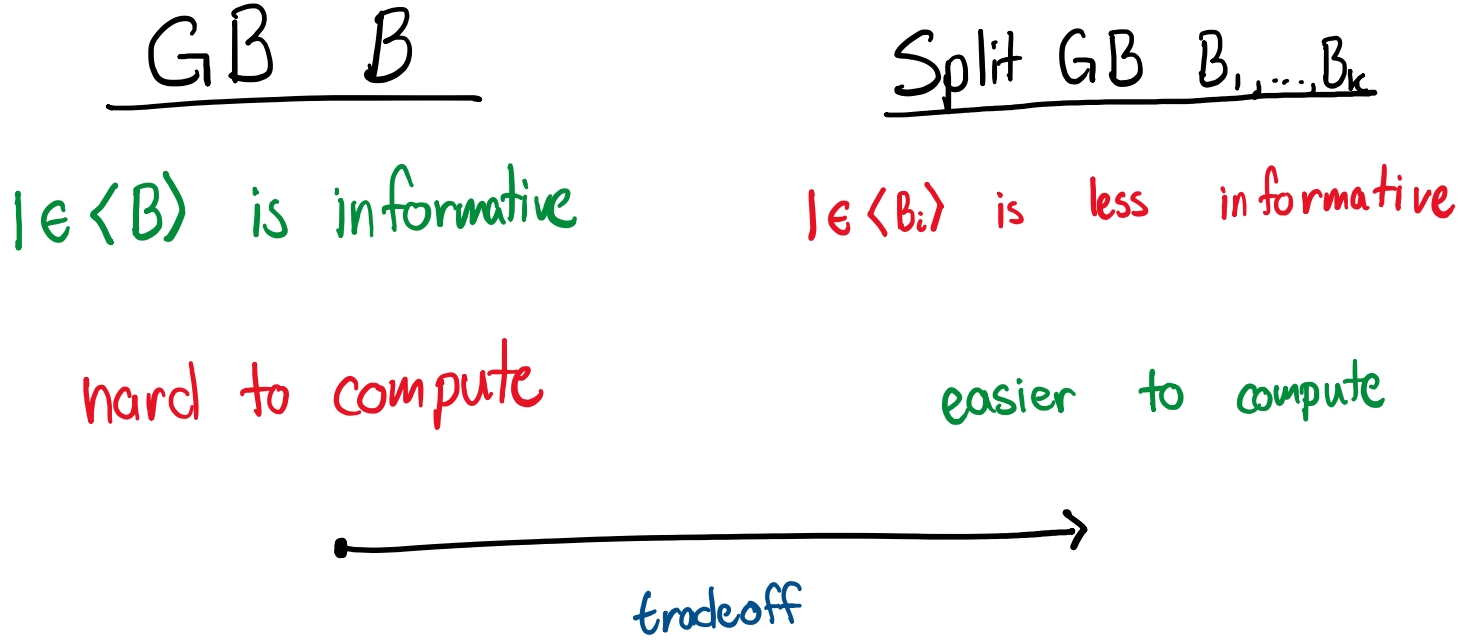
Definition

A Split GB for $\langle S \rangle$ is a sequence of poly sets

$$\boldsymbol{B} = (B_1, \ldots B_k)$$

such that

- each B_i is a GB and
- $\langle \boldsymbol{B} \rangle \triangleq \langle \cup_i B_i \rangle = \langle S \rangle$.



An overview of the **Split** solver

```
1 Function Monolithic:
      In: G ⊂ 𝔽[X]
      Out: A zero M ∈ 𝒱(G) or ⊥
2
3     B ← GB(G);
4     if 1 ∈ ⟨B⟩ then return ⊥;
5     return FindZero(B)
```

```
1 Function Split:
      In: G ⊂ 𝔽[X]
      Out: A zero M ∈ 𝒱(G) or ⊥
2     G ← ({p ∈ G : init(i, p)})_{i=1}^{s};
3     B ← SplitGB(G);
4     if ∃ i. 1 ∈ ⟨B_i⟩ then return ⊥;
5     return SplitFindZero(B)
```

first two parameters

Computing a split GB

$\mathrm{SplitGB}(\boldsymbol{G}) \to \boldsymbol{B}$:

- For each G_i simultaneously:
  1. Compute a GB B_i
  2. Offer $p \in B_i$ to other B_j's
  3. Accept p into B_j if $\mathrm{admit}(j, p)$ and $p \notin \langle B_j \rangle$
  4. Accept $p \in \mathrm{extraProp}(\boldsymbol{B})$ if $\mathrm{admit}(j, p)$ and $p \notin \langle B_j \rangle$
  5. Repeat to fixpoint
- Return $\boldsymbol{B}$

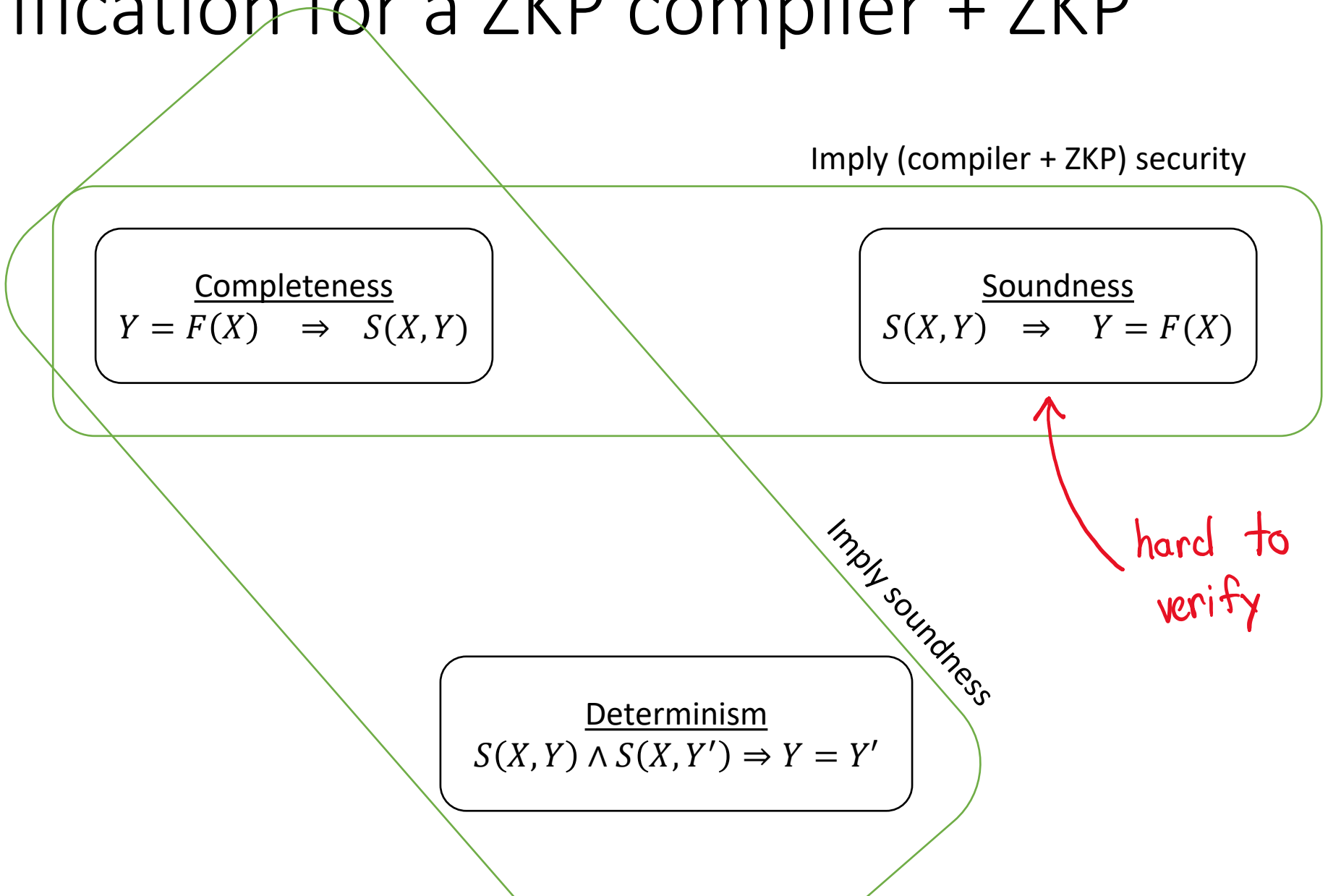
third param: reject p that would slow GBs

fourth param: find $p \in \langle \mathbf{B} \rangle$ with non-GB reasoning

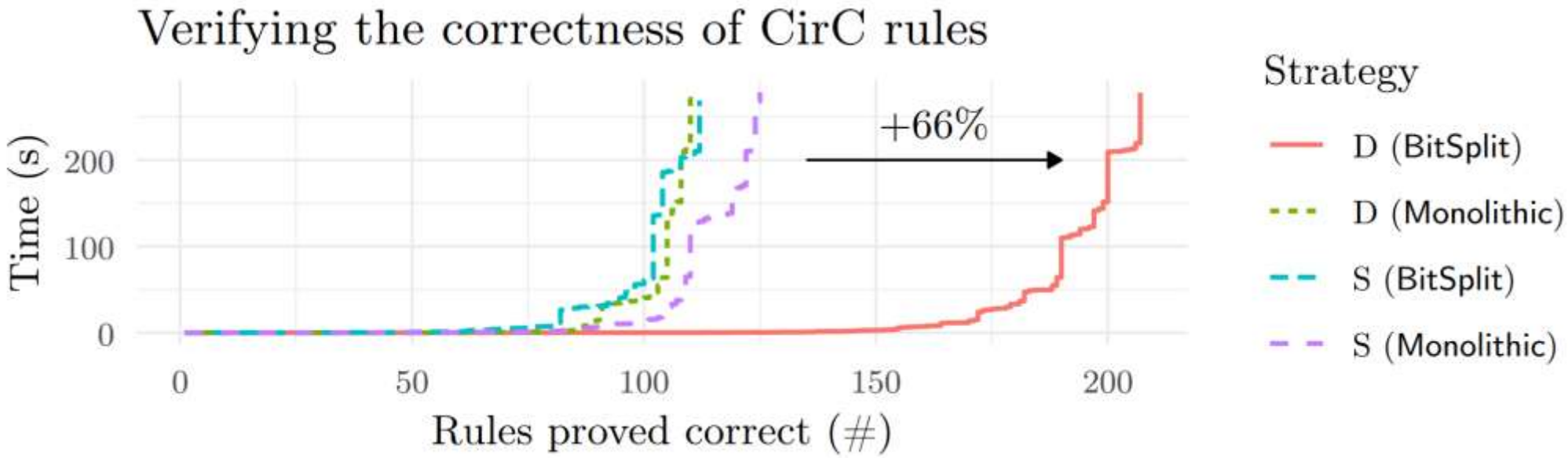
The benchmarks

Family	#	Description
CirC-D	640	Determinism for CirC $\mathbb{F}$-blaster rules of bitwidth ≤ 32 (Sec. 6)
Seq	100	Determinism for sequenced bit-splits (App. F)
QED^2	100	Determinism for circomlib, generated by QED^2 [48]
CirC-S	100	Soundness for CirC $\mathbb{F}$-blaster rules of bitwidth ≤ 4 [47]
TV	100	Translation validation for ZKP compilers on boolean programs [46]
Small	100	Randomly generated with a small field: $\lvert\mathbb{F}\rvert \leq 211$ [34]

Verification for a ZKP compiler + ZKP



Best option: determinism w/ BitSplit



Variations of **BitSplit** aren't better

Solver	Solved	By Family						By Result	
		CirC-D	Seq	QED^2	CirC-S	TV	Small	SAT	UNSAT
BitSplit	**969**	**582**	**100**	**59**	92	70	66	88	**881**
BS-LinFirst	959	576	**100**	58	92	69	64	84	875
BS-NoIntProp	877	576	24	58	86	70	63	84	793
BS-NoExtProp	344	131	0	34	45	69	65	85	259
BS-FullIntProp	953	576	97	56	92	69	63	83	870
BS-DenseProp	898	580	33	58	92	70	65	85	813
BS-QuadProp	898	580	32	**59**	92	71	64	86	812
Monolithic	475	191	13	38	**94**	**72**	**67**	**90**	385