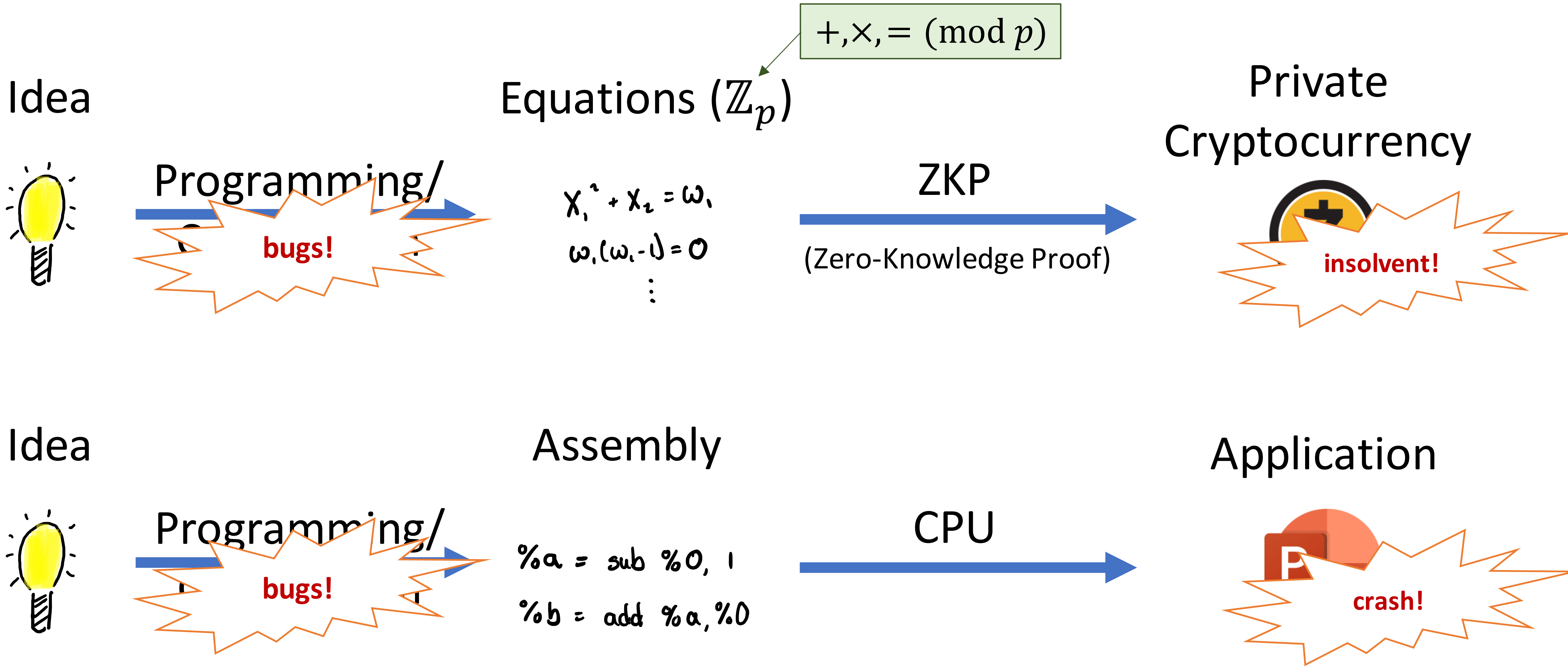
Satisfiability Modulo Finite Fields

(with applications to compilers for zero-knowledge proofs)

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett [CAV’23]

How Private Cryptocurrencies are Built



The Verification Problem

Idea

Programming/
Compilation

Equations ($\mathbb{Z}_p$)

$$x_1^2 + x_2 = \omega_1$$
$$\omega_1(\omega_1 - 1) = 0$$
$$\vdots$$

ZKP

(Zero-Knowledge Proof)

Private
Cryptocurrency

Can we verify this?

SMT Solver: a general-purpose theorem prover/constraint solver

Specification

- Boolean logic ($\mathbb{Z}_2$)
- Bit-vectors ($\mathbb{Z}_{2^b}$)
- Arrays …

SMT

Implementation

equations in $\mathbb{Z}_p$

Not SMT (previously)

Our Contribution: $\mathbb{Z}_p$ reasoning in SMT

- A theory of finite fields ($\mathbb{Z}_p$)
- Verification problems
  - ZKP translation validation (Boolean functions)
- A decision procedure
  - ~~based on computer algebra~~
  - ~~constructs UNSAT cores~~
  - avoids field polynomials
- An implementation in cvc5
  - Empirically fast

Automated Verification

(SMT) solvers

+ $\mathbb{Z}_p$

Bonus: An application to partially verified finite-field blasting

Part I: Prime-Order Finite Fields

- Written $\mathbb{Z}_p$
- A set of integers: $\{0, \dots, p-1\}$
- Operations:
  - $+$ (mod p)
  - $\times$ (mod p)
  - $=$

Theory of prime-order finite fields

- For each prime p
  - A sort FF_p
  - Function symbols $+, \times, =$
  - Constants $0, 1, \dots, p-1$
- Notes:
  - fixed p

$X = X + 1$

UNSAT

$X = X + 1$

$\vee$

$Y^2 = Y$

SAT : $X = 0, Y = 0$

Part II: Verifying encodings of Boolean function evaluation

Encoding a Boolean Function as Equations

- Consider a Boolean fn. $f(z_i) \to z$
  - Ex: $z = f(z_0, z_1) = z_0 \wedge z_1$
- We want a system ϕ':
  - In variables:
    - z_i': represents z_i
    - z': represents z
    - w_i': additional variables
  - Such that
    - $[\underset{\substack{\uparrow \\ \text{equations hold}}}{\phi'} \wedge \underbrace{\bigwedge_i z_i' = \text{ite}(z_i, 1{,}0)}_{\text{inputs are valid}}] \to \underbrace{z' = \text{ite}(f(z_i), 1{,}0)}_{\text{output is correct}}$

For our example:

$$z' = z_0' z_1'$$

How We Generate Verification Problems

1. Sample a Boolean function $f(z_i) \to z$
2. Compile it to a system of equations: $\phi \leftarrow \mathrm{ZKPCompiler}(f)$
3. (Optional) remove some equations
4. Write the soundness and determinism conditions

- ZoKrates
- CirC
- CirC-ZoK

- Removals influence condition validity:
  - No removals: VALID (or the compiler has a bug!)
  - Removals: likely INVALID
- Benchmarks are in an SMT-LIB2 extension to $\mathbb{Z}_p$
  - QF_FF: mix of Boolean and $\mathbb{Z}_p$ ($p \approx 2^{256}$)
  - Alternatives: non-linear integer arithmetic, bit-vector arithmetic, pure $\mathbb{Z}_p$

Different ways to verify

- Option 1: verify these conditions as **QF_FF**:
  - Booleans
  - $\mathbb{Z}_p$ equations like $x \times y = z$ with $x, y, z \in \mathbb{Z}_p$
- Options 2, 3: explicit modulus
  - $(x \times y) \% p = z$ with
    - $x, y, z \in \{0, \ldots, p-1\} \subset \mathbb{Z}$ (**QF_NIA**)
    - $x, y, z \in \{0, \ldots, p-1\} \subset \mathbb{Z}_{2^b}$ (**QF_BV**)
      - Requires $b > \lceil 2 \log p \rceil$
- Option 4: encode the Booleans as $\mathbb{Z}_p$ (**Pure FF**)

SMT + $\mathbb{Z}_p$

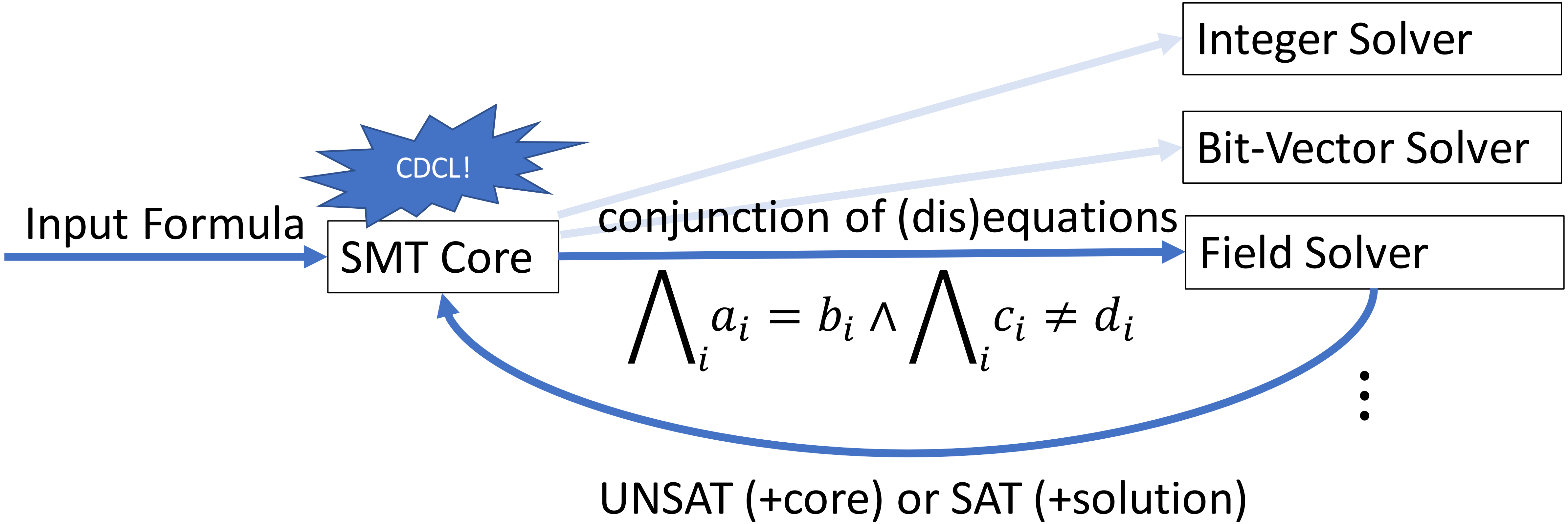
Existing SMT solvers

Computer Algebra

Part III: Decision Procedure

Building an **SMT** ***theory solver*** for $\mathbb{Z}_p$

SMT + Theory Solvers



$\mathbb{Z}_p$ Solver Pseudocode

1. Given: equalities & disequalities
2. Convert each to form: $t_i = 0$
   - t_i: a polynomial in the input variables
3. Exist polys p_i such that $1 = \sum_i p_i t_i$?
   - Return UNSAT (with core)
4. Backtracking search for a solution

Rules:
- $a_i = b_i \Rightarrow a_i - b_i = 0$
- $a_i \neq b_i \Rightarrow (a_i - b_i)w_i - 1 = 0$

Example:
- $X = 0$

Incomplete, fast UNSAT

Slow, complete.
Finds solution.

Exist polys p_i such that $1 = \sum_i p_i t_i$? (⇒UNSAT)

- Example:
  - $t_1 = XY - 1, t_2 = X$
  - $(-1)t_1 + (Y)t_2$
    - $= -XY + 1 + XY = 1$
- Gröbner Basis engine:
  - Determines whether the p_i exist
  - Worst-case time: $O\left(d^{2^n}\right)$
  - Empirical time: fast (see eval)

- $\exists p_i \Rightarrow \text{UNSAT}$ (why?)
  - $1 = \sum_i p_i \cdot t_i = \sum_i p_i \cdot 0 = 0$
- **Problem:**
  - $\neg\exists p_i \not\Rightarrow \text{SAT}$
    - Example: $X^2 + 1 = 0$ is UNSAT in $\mathbb{Z}_3$
    - Key issue: $X = \sqrt{-1}$ is a solution
- Classic solution:
  - Add polynomials $X^p - X$
    - $\text{roots}(X^p - X) = \{0, \dots, p-1\}$
  - **Problem:** high-degree ⇒ slow GB

$\boxed{\text{Live with the incompleteness…}}$

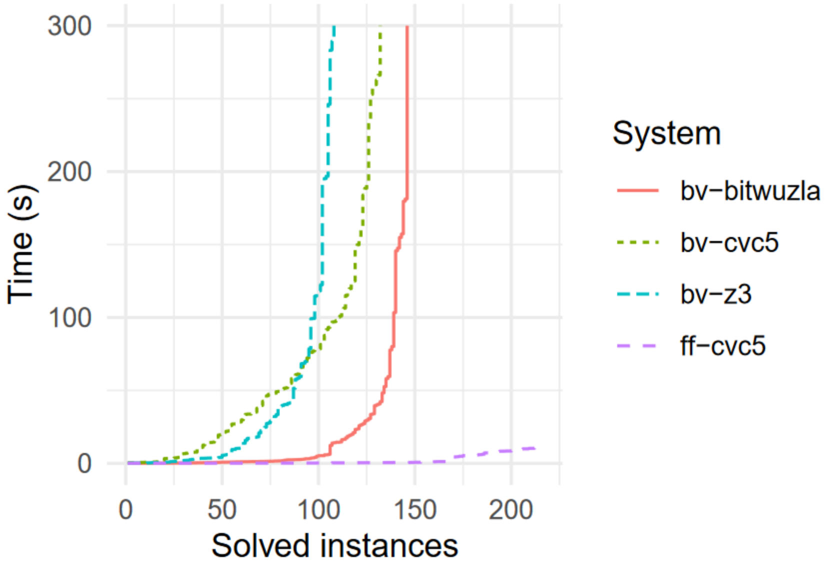
$\mathbb{Z}_p$ Solver Pseudocode

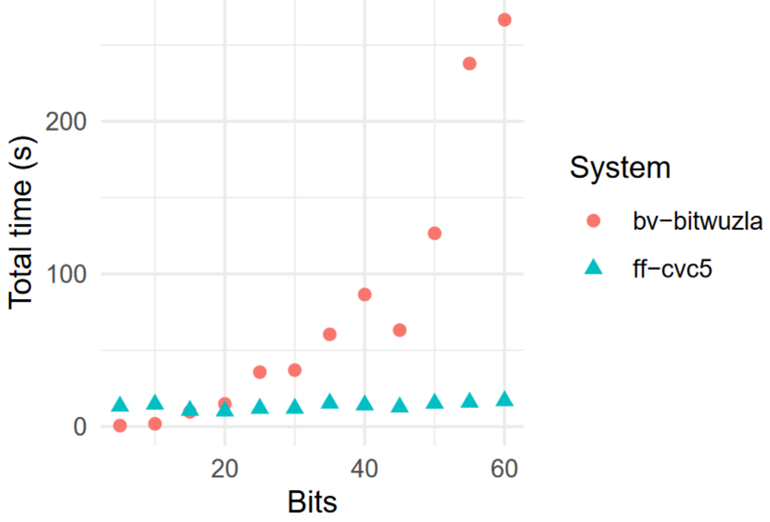
1. Given: equalities & disequalities
2. Convert each to form: $t_i = 0$
   - t_i: a polynomial in the input variables
3. Exist polys p_i such that $1 = \sum_i p_i t_i$?
   - Return UNSAT (with core)
4. Backtracking search for a solution
   - If ∃ univariate $t_i(X_j)$: factor & branch
   - If $\dim(t_i) = 0$: compute a univariate, factor, & branch
   - Else: exhaustive search

- Could perform *terribly.*
- Experiments show otherwise.
- Insight: + dimension ⇝ many solutions

Part IV: Experiments

Comparison with Bit-Vectors





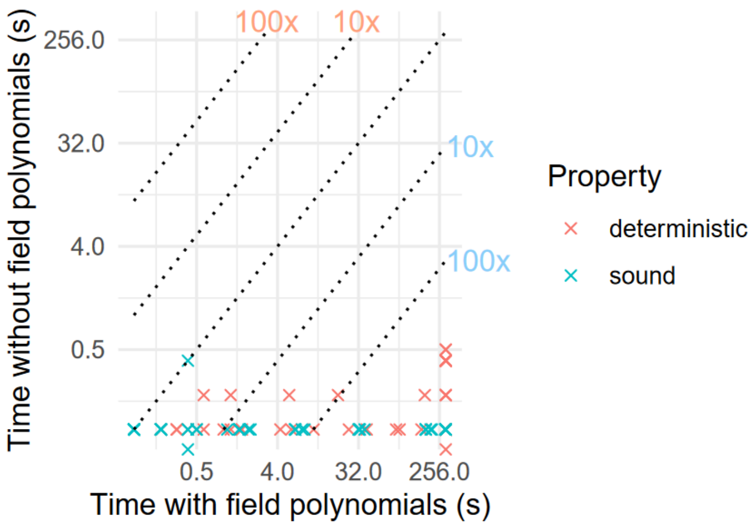
(p: 5-60 bits, 1CPU, 8GB, 5min)

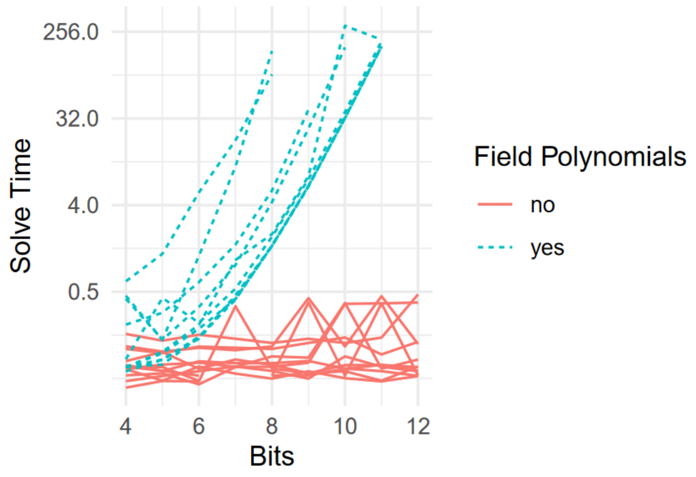
(commonly solved)

BV is worse, even for small fields

BV scales poorly with field size

Effect of Field Polynomials





(p: 4-12 bits, 8GB, 5min)

(commonly solved SAT)

Field polys: terrible for even tiny fields

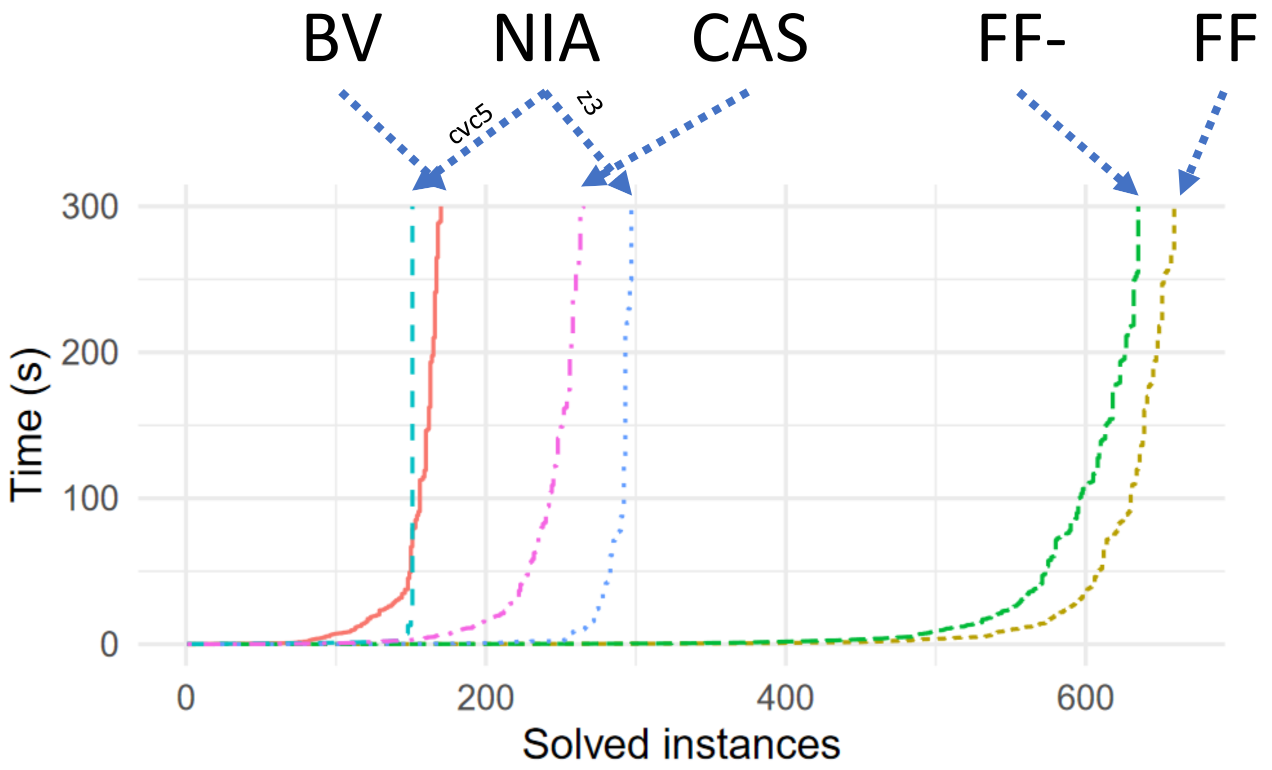
Field polys scale poorly with field size

Main Experiment

Full-size field: $p \approx 2^{255}$

What we benchmarked:

- Our solver (FF)
- Our solver, no cores (FF-)
- Embed $\mathbb{Z}_p$ in bit-vectors (BV)
- Embed $\mathbb{Z}_p$ in integers (NIA)
- Embed Booleans in $\mathbb{Z}_p$ (CAS)



solvers: bitwuzla-1.0pre, z3-4.11.2, cvc5 nightly 17 Dec ’22, CoCoALib 0.998

More in the paper

- Translation validation for *determinism*
- Theory & decision procedure for non-prime finite fields.
- UNSAT cores
  - Calculus for proving ideal membership

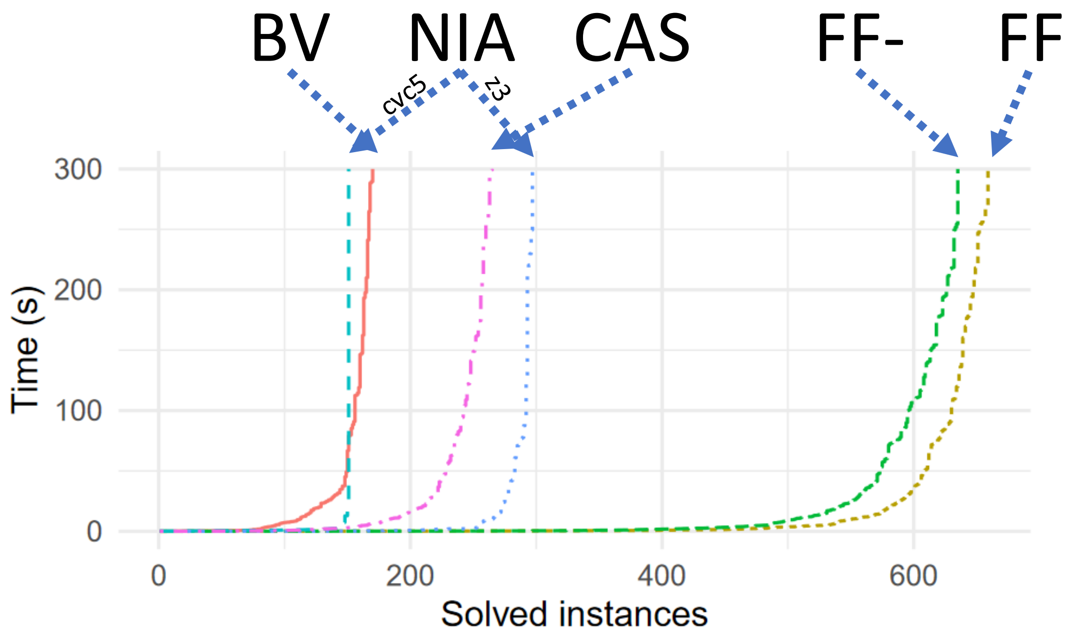
Future Work:

- Worst-case runtime $O_{d,n}(\log p)$, not $O_{d,n}(p)$?
- Avoid full GB?
- Incremental computation *within* field solver.

Satisfiability Modulo Finite Fields

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett

ePrint: https://ia.cr/2023/091



- First SMT $\mathbb{Z}_p$ solver
  - UNSAT cores
  - No field polynomials
- First QF_FF benchmarks
  - Application area: ZKPs
- A foundation for $\mathbb{Z}_p$ verification
- Try it! (cvc5 main, all APIs)

Bonus: Bounded Verification for Finite-Field Blasting

The Verification Problem

Idea

Programming/

Compilation

Equations ($\mathbb{Z}_p$)

$x_1^2 + x_2 = \omega_1$

$\omega_1(\omega_1 - 1) = 0$

$\vdots$

Can we verify this?

ZKP

(Zero-Knowledge Proof)

Private Cryptocurrency

Previous benchmarks: translation validation

What about (automatic) verification?

The CirC Compiler Infrastructure

Program → IR[…] → IR[Bool,BV,$\mathbb{Z}_p$] → $\mathbb{Z}_p$-Equations

Front-End

Optimization,
Simplification

“Finite-Field Basting”

Correctness for a ZKP compiler

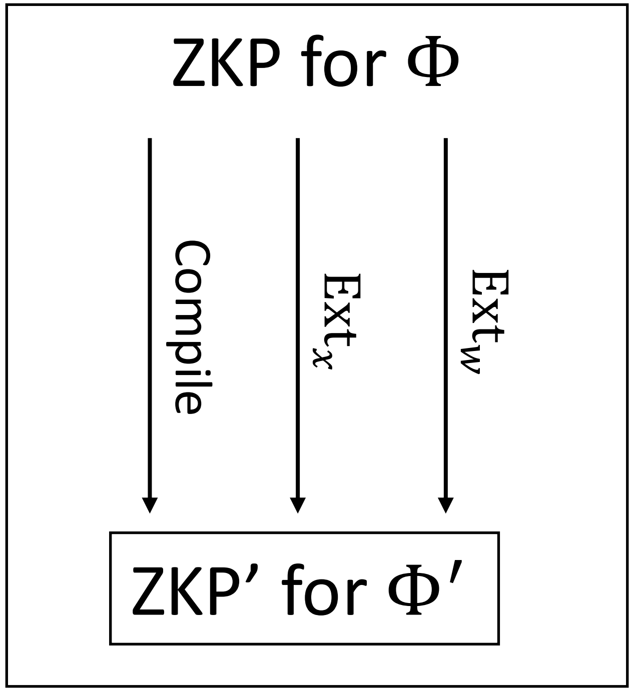
<u>Syntax:</u>

equisat

$$\mathrm{Compile}(\phi(x, w) \in \Phi) \rightarrow \phi'(x', w') \in \Phi'$$

$$\mathrm{Ext}_x(x) \rightarrow x'$$

$$\mathrm{Ext}_w(x, w) \rightarrow w'$$



<u>Correctness:</u>

- *Demonstrable* Soundness
- *Demonstrable* Completeness

<u>Key Properties:</u>

- Closed under sequential composition
- ZKP is secure

The CirC Compiler Infrastructure

Program → IR[…] → IR[Bool,BV,$\mathbb{Z}_p$] → $\mathbb{Z}_p$-Equations

Front-End

Optimization,
Simplification

“Finite-Field Blasting”

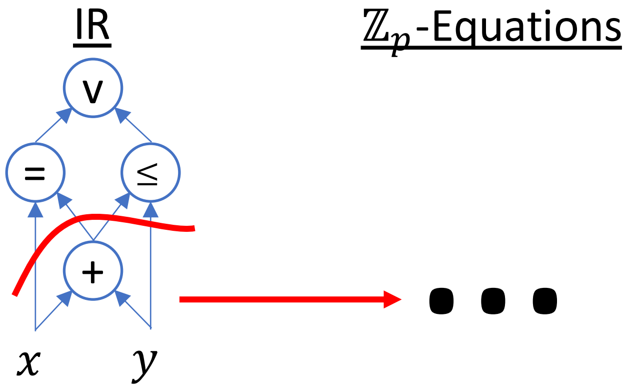
correct

correct

correct

The anatomy of a field blaster

Post-order traversal



Encodings

Bool $x \to x' \in \{0,1\} \in \mathbb{Z}_p$

BV $x \to x' \in \{0, \dots, 2^b - 1\} \in \mathbb{Z}_p$

BV $x \to x'_1, \dots, x'_b \in \{0,1\} \in \mathbb{Z}_p$

Field Element $x \to x' \in \mathbb{Z}_p$

Encoding Rules

bv.add	bv.mul		
bv.sub	bv.udiv	bo	
bv.lshl	bv.not		
bv.shr	bv.shr'		ion
bv.ashl	bv.urem	bool xor	*variable introductions*
bv.uge	bv.xor	ff.mul	*private variable introductions*
bv.or	bv.and	ff.add	

……………………………………………

Automation Needed

Verification Recipe

- Post-order traversal → inductive proof
- Encoding ‘validity’ → inductive invariants
- Each encoding rule → verification conditions
  - Soundness + completeness
  - Automatically generated
  - Automatically verified (SMT)
    - Up to tiny bounds on:
      - bit-vector length (4)
      - operator arity (4)

4 Bugs

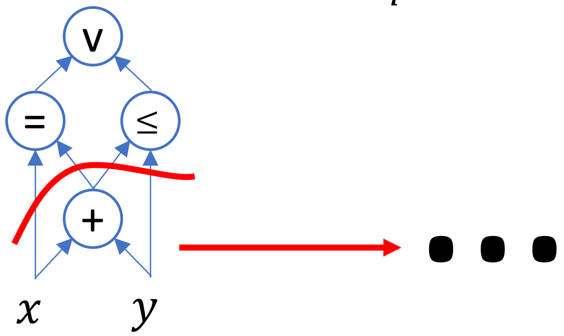
- Incompleteness in `bv.*sh*`
- Incompleteness in `ff.div`
- Unsoundness in `bv.udiv`
- Unsoundness in bv comparisons
  - **Severe**

Bounded Verification for Finite-Field Blasting

Alex Ozdemir, Riad S. Wahby, Fraser Brown, Clark Barrett

Post-order traversal

IR $\mathbb{Z}_p$-Equations



Encodings

Bool $x \to x' \in \{0,1\} \in \mathbb{Z}_p$

BV $x \to x' \in \{0, \dots, 2^b - 1\} \in \mathbb{Z}_p$

BV $x \to x'_1, \dots, x'_b \in \{0,1\} \in \mathbb{Z}_p$

Field Element $x \to x' \in \mathbb{Z}_p$

Encoding Rules

bv.add bv.mul

bv.sub bv.udiv

bv.lshl bv.not

bv.shr bv.shr'

bv.ashl bv.urem bool xor *variable introductions*

bv.uge bv.xor ff.mul *private variable introductions*

bv.or bv.and ff.add

....

Automation Needed

Two Properties

Soundness

Any solution with valid inputs has the correct output.

$$\phi' \wedge \bigwedge_i z_i' = \mathrm{ite}(z_i, 1,0)$$
$$\rightarrow$$
$$z' = \mathrm{ite}(f(z_i), 1,0)$$

Determinism

Any solution, for fixed inputs, has a unique output.

$$\phi' \wedge \phi'' \wedge \bigwedge_i z_i' = z_i''$$

copy of ϕ'

$$\rightarrow$$
$$z' = z''$$