SMT + Finite Fields
A foundation for field-related verification

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett [CAV’23]
Verifying Equations for a ZKP:

**Idea**

Can we verify this?

**Specification**

Boolean logic ($\mathbb{Z}_2$)

Bit-vectors ($\mathbb{Z}_{2^b}$)

Arrays ...

**Implementation**

equations in $\mathbb{Z}_p$

**Equations ($\mathbb{Z}_p$)**

$\chi_i^* + x_i = \omega_i$

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

**Compilation**

**ZKP** (Zero-Knowledge Proof)

(Zero-Knowledge Proof)

**Private Cryptocurrency**

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

- equations in $\mathbb{Z}_p$ (mod $p$)
- equations in $\mathbb{Z}_{2^b}$ (mod $p$)

bugs!

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

- A theory of finite fields ($\mathbb{Z}_p$)
- Verification problems
  - translation validation for ZKP compilers (Boolean functions)
- A decision procedure
  - uses computer algebra
  - avoids field polynomials
- An implementation in cvc5

Bonus: An application to partially verified finite-field blasting (in CirC)
Prime-Order Finite Fields

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

\[
\begin{align*}
X &= X + 1 \\
\forall Y^2 &= Y \\
\text{SAT} : \ X &= 0, \ Y = 0 \\
X &= X + 1 \\
\text{UNSAT}
\end{align*}
\]
Encoding a Boolean Function as Equations

• Consider a Boolean formula \( f(z_i) \rightarrow z \)
• Consider a system \( \phi' \):
  • In variables:
    • \( z'_i \): represents \( z_i \)
    • \( z' \): represents \( z \)
    • additional variables

• \( \phi' \) is **sound** w.r.t \( f \) when this is valid:
  • \( [\phi' \land \land_i z'_i = \text{ite}(z_i, 1,0)] \rightarrow z' = \text{ite}(f(z_i), 1,0) \)

Example

\[
Z = Z_0 \land Z_1 \land Z_2
\]

\[
Z' = Z'_0 Z'_1 Z'_2
\]
Generating Verification Problems

Sampler → Boolean formula → ZKP Compiler → Field equations → Maybe Drop One → Field equations → Build Formulas

Drop ↔ SAT

SMT queries: Booleans + $\mathbb{Z}_p$

$(p \approx 2^{256})$
Decision Procedure:

Problem: SAT in extension field \((x^p - x)\) → Unsound!

'Solution': field polynomials \((x^p - x)\)

pro: roots are (just) \(\mathbb{Z}_p\) con: degree is \(p\) (~2^{255})

Gröbner Basis:

\[ 1 \in \langle t_i \rangle? \]

\(\mathbb{Z}_p\)-equations

\[
\bigwedge_i t_i = 0
\]

Backtracking Search

\(\bigO\left( d^{2^n} \right)\)

SAT

Worst-case \(O_{n,d}(p)\)

UNSAT
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
• Full decision procedure
  • backtracking search
  • for all finite fields
  • UNSAT cores
    • Calculus for proving ideal membership
• Detailed benchmarks

Future Work:
• Asymptotic improvements?
  • $O_{d,n}(p)$ to $O_{d,n}(\log p)$?
• Incremental computation within field solver.
Bounded Verification in CirC

“Bounded Verification for Finite Field Blasting”

Alex Ozdemir, Riad S. Wahby, Fraser Brown, Clark Barrett [CAV’23]
The Verification Problem

Idea

Programming/Compilation

Equations ($\mathbb{Z}_p$)

\[
\chi_i \cdot \chi_\ell = \omega_i \\
\omega_i (\omega_i - 1) = 0
\]

ZKP
(Zero-Knowledge Proof)

Private Cryptocurrency

Can we verify this?

Previous benchmarks: translation validation

What about (automatic) verification?
The CirC Compiler Infrastructure

Program → IR[...] → IR[Boolean,BV,ℤ_p] → ℤ_p-Equations

Front-End
Optimization, Simplification

“Finite-Field Basting”

Encoding Rules:
- bv.add
- bv.sub
- bv.lshl
- bv.shr
- bv.ashl
- bv.uge
- bv.ule
- bv.ult
- bv.ugt
- bv.sge
- bv.sle
- bv.slt
- bv.sgt
- bv.or
- bv.xor
- bv.and
- bv.mul
- bv.udiv
- bv.not
- bv.neg
- bv.urem
- bool.and
- bool.or
- bool.xor
- bool.not
- ff.div
- ff.add
- ff.neg
- ff.mul

IR traversal → induction proof
valid encoding → inductive invariant

Each rule → SMT query

4 bugs!
Satisfiability Modulo Finite Fields

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett [CAV’23]  
ePrint: https://ia.cr/2023/091

- Solver for SMT with $\mathbb{Z}_p$
- First SMT $\mathbb{Z}_p$ benchmarks
  - ZKP security
- Application: CirC’s $\mathbb{Z}_p$-blaster
- Try it! (cvc5 main, all APIs)

Bounded Verification for Finite-Field Blasting

Alex Ozdemir, Riad S. Wahby, Fraser Brown, Clark Barrett [CAV’23]
Appendices
Two Properties

**Soundness**

Any solution with valid inputs has the correct output.

\[
\phi' \land \bigwedge_i z'_i = \text{ite}(z_i, 1, 0) \\
\rightarrow \\
z' = \text{ite}(f(z_i), 1, 0)
\]

**Determinism**

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

\[
\phi' \land \phi'' \land \bigwedge_i z'_i = z'' \\
\text{copy of } \phi' \rightarrow \\
z' = z''
\]
What does a theory solver do?

SMT Formula → SMT Core → conjunction of (dis)equations

\[ \bigwedge_i a_i = b_i \land \bigwedge_i c_i \neq d_i \]

→ Integer Solver
→ Bit-Vector Solver
→ Field Solver

UNSAT (+core) or SAT (+solution)
Decision Procedure:

Problem: SAT in extension field \((x^2=1)\)

'Solution': field polynomials \((x^p-x)\)

pro: roots are (just) \(\mathbb{Z}_p\)
con: degree is \(p\) \((\approx 2^{256})\)

\[\exists \text{ polys } p_i, \quad 1 = \sum_i p_i \cdot t_i\]

\(\mathbb{Z}_p\)-equations

Backtracking Search

Sound!

\[1 = \sum p_i t_i = \sum p_i \cdot 0 = 0\]

Worst-case \(O_{n,d}(p)\)

Unsound!

\(x^2+1=0\) \((\mathbb{Z}_p)\)

SAT

UNSAT

How?
Gröbner Bases!
\(O(d^g)\)
\[ \mathbb{Z}_p \text{ Solver Pseudocode} \]

1. Given: equalities & disequalities
2. Rewrite as: \( t_i = 0 \)
   - \( t_i \): 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

- Incomplete, fast UNSAT
- Slow, complete. Finds solution.
Exist polys $p_i$ such that $1 = \sum_i p_i t_i$? ($\Rightarrow$ 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: $O(d^{2^n})$
  • Empirical time: fast

• $\exists p_i \Rightarrow$ UNSAT (why?)
  • $1 = \sum_i p_i \cdot t_i = \sum_i p_i \cdot 0 = 0$

• Problem:
  • $\neg \exists p_i \Rightarrow$ SAT
    • Example: $X^2 + 1 = 0$ is UNSAT in $\mathbb{Z}_3$
    • Key issue: $X = \sqrt{-1}$ is a solution

• Classic solution:
  • Add polynomials $X^p - X$
    • roots$(X^p - X) = \{0, ..., p - 1\}$
  • Problem: high-degree $\Rightarrow$ slow GB

Live with the incompleteness...
Part IV: Experiments
Effect of Field Polynomials

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

Field polys: terrible for even tiny fields

Field polys scale poorly with field size

(commonly solved SAT)
Comparison with Bit-Vectors

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

BV is worse, even for small fields

BV scales poorly with field size

(commonly solved)
Correctness for a ZKP compiler

Syntax:
Compile($\phi(x, w) \in \Phi$) $\rightarrow$ $\phi'(x', w') \in \Phi'$
Ext$_x(x) \rightarrow x'$
Ext$_w(x, w) \rightarrow w'$

Correctness:
• *Demonstrable* Soundness
• *Demonstrable* Completeness

Key Properties:
• Closed under sequential composition
• ZKP is secure
Verification Recipe

- Post-order traversal $\rightarrow$ inductive proof
- Encoding 'validity' $\rightarrow$ inductive invariants
- Each encoding rule $\rightarrow$ 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