Satisfiability Modulo Finite Fields

Alex Ozdemir[1(✉)], Gereon Kremer[1,2], Cesare Tinelli[3], and Clark Barrett[1]

[1] Stanford University, Stanford, USA
aozdemir@stanford.edu
[2] Certora, Tel Aviv-Yafo, Israel
[3] University of Iowa, Iowa, USA

Abstract. We study satisfiability modulo the theory of finite fields and give a decision procedure for this theory. We implement our procedure for prime fields inside the cvc5 SMT solver. Using this theory, we construct SMT queries that encode translation validation for various zero knowledge proof compilers applied to Boolean computations. We evaluate our procedure on these benchmarks. Our experiments show that our implementation is superior to previous approaches (which encode field arithmetic using integers or bit-vectors).

1 Introduction

Finite fields are critical to the design of recent cryptosystems. For instance, elliptic curve operations are defined in terms of operations in a finite field. Also, Zero-Knowledge Proofs (ZKPs) and Multi-Party Computations (MPCs), powerful tools for building secure and private systems, often require key properties of the system to be expressed as operations in a finite field.

Field-based cryptosystems already safeguard everything from our money to our privacy. Over 80% of our TLS connections, for example, use elliptic curves [4,66]. Private cryptocurrencies [32,59,89] built on ZKPs have billion-dollar market capitalizations [44,45]. And MPC protocols have been used to operate auctions [17], facilitate sensitive cross-agency collaboration in the US federal government [5], and compute cross-company pay gaps [8]. These systems safeguard our privacy, assets, and government data. Their importance justifies spending considerable effort to ensure that the systems are free of bugs that could compromise the resources they are trying to protect; thus, they are prime targets for formal verification.

However, verifying field-based cryptosystems is challenging, in part because current automated verification tools do not reason directly about finite fields. Many tools use Satisfiability Modulo Theories (SMT) solvers as a back-end [9,27,33,93,95]. SMT solvers [7,10,12,20,26,35,73,76,77] are automated reasoners that determine the satisfiability of formulas in first-order logic with respect to one or more *background theories*. They combine propositional search with specialized reasoning procedures for these theories, which model common data types such as Booleans, integers, reals, bit-vectors, arrays, algebraic datatypes, and more.

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 163–186, 2023.
https://doi.org/10.1007/978-3-031-37703-7_8

Since SMT solvers do not currently support a theory of finite fields, SMT-based tools must encode field operations using another theory.

There are two natural ways to represent finite fields using commonly supported theories in SMT, but both are ultimately inefficient. Recall that a finite field of prime order can be represented as the integers with addition and multiplication performed modulo a prime p. Thus, field operations can be represented using integers or bit-vectors: both support addition, multiplication, and modular reduction. However, both approaches fall short. Non-linear integer reasoning is notoriously challenging for SMT solvers, and bit-vector solvers perform abysmally on fields of cryptographic size (hundreds of bits).

In this paper, we develop for the first time a direct solver for finite fields within an SMT solver. We use well-known ideas from computer algebra (specifically, Gröbner bases [21] and triangular decomposition [6,99]) to form the basis of our decision procedure. However, we improve on this baseline in two important ways. First, our decision procedure does not manipulate *field polynomials* (i.e., those of form $X^p - X$). As expected, this results in a loss of completeness at the Gröbner basis stage. However, surprisingly, this often does not matter. Furthermore, completeness is recovered during the model construction algorithm (albeit in a rather rudimentary way). This modification turns out to be crucial for obtaining reasonable performance. Second, we implement a proof-tracing mechanism in the Gröbner basis engine, thereby enabling it to compute unsatisfiable cores, which is also very beneficial in the context of SMT solving. Finally, we implement all of this as a theory solver for prime-order fields inside the cvc5 SMT solver.

To guide research in this area, we also give a first set of `QF_FF` (quantifier-free, finite field) benchmarks, obtained from the domain of ZKP compiler correctness. ZKP compilers translate from high-level computations (e.g., over Booleans, bit-vectors, arrays, etc.) to systems of finite field constraints that are usable by ZKPs. We instrument existing ZKP compilers to produce translation validation [86] verification conditions, i.e. conditions that represent desirable correctness properties of a specific compilation. We give these compilers concrete Boolean computations (which we sample at random), and construct SMT formulas capturing the correctness of the ZKP compilers' translations of those computations into field constraints. We represent the formulas using both our new theory of finite fields and also the alternative theory encodings mentioned above.

We evaluate our tool on these benchmarks and compare it to the approaches based on bit-vectors, integers, and pure computer algebra (without SMT). We find that our tool significantly outperforms the other solutions. Compared to the best previous solution (we list prior alternatives in Sect. 7), it is 6× faster and it solves 2× more benchmarks.

In sum, our contributions are:

1. a definition of the theory of finite fields in the context of SMT;
2. a decision procedure for this theory that avoids field polynomials and produces unsatisfiable cores;
3. the first public theory solver for this theory (implemented in cvc5); and

4. the first set of `QF_FF` benchmarks, which encode translation validation queries for ZKP compilers on Boolean computations.

In the rest of the paper, we discuss related work (§1.1), cover background and notation (§2), define the theory of finite fields (§3), give a decision procedure (§4), describe our implementation (§5), explain the benchmarks (§6), and report on experiments (§7).

1.1 Related Work

There is a large body of work on computer algebra, with many algorithms implemented in various tools [1,18,31,37,49,52,58,72,100,101]. However, the focus in this work is on quickly constructing useful algebraic objects (e.g., a Gröbner basis), rather than on searching for a solution to a set of field constraints.

One line of recent work [54,55] by Hader and Kovács considers SMT-oriented field reasoning. One difference with our work is that it scales poorly with field size because it uses field polynomials to achieve completeness. Furthermore, their solver is not public.

Others consider verifying field constraints used in ZKPs. One paper surveys possible approaches [97], and another considers proof-producing ZKP compilation [24]. However, neither develops automated, general-purpose tools.

Still other works study automated reasoning for non-linear arithmetic over reals and integers [3,23,25,29,47,60–62,70,74,96,98]. A key challenge is reasoning about *comparisons*. We work over finite fields and do not consider comparisons because they are used for neither elliptic curves nor most ZKPs.

Further afield, researchers have developed techniques for verified algebraic reasoning in proof assistants [15,64,75,79], with applications to mathematics [19,28,51,65] and cryptography [39,40,85,91]. In contrast, our focus is on *fully automated* reasoning about finite fields.

2 Background

2.1 Algebra

Here, we summarize algebraic definitions and facts that we will use; see [71, Chapters 1 through 8] or [34, Part IV] for a full presentation.

Finite Fields. A *finite field* is a finite set equipped with binary operations $+$ and $\times$ that have identities (0 and 1 respectively), have inverses (save that there is no multiplicative inverse for 0), and satisfy associativity, commutativity, and distributivity. The *order* of a finite field is the size of the set. All finite fields have order $q = p^e$ for some prime p (called the *characteristic*) and positive integer e. Such an integer q is called a *prime power*.

Up to isomorphism, the field of order q is unique and is denoted $\mathbb{F}_q$, or $\mathbb{F}$ when the order is clear from context. The fields $\mathbb{F}_{q^d}$ for $d > 1$ are called *extension fields* of $\mathbb{F}_q$. In contrast, $\mathbb{F}_q$ may be called the *base field*. We write $\mathbb{F} \subset \mathbb{G}$ to indicate

that $\mathbb{F}$ is a field that is isomorphic to the result of restricting field $\mathbb{G}$ to some subset of its elements (but with the same operations). We note in particular that $\mathbb{F}_q \subset \mathbb{F}_{q^d}$. A field of prime order p is called a *prime field*.

Polynomials. For a finite field $\mathbb{F}$ and formal variables $X_1, \ldots, X_k$, $\mathbb{F}[X_1, \ldots, X_k]$ denotes the set of polynomials in $X_1, \ldots, X_k$ with coefficients in $\mathbb{F}$. By taking the variables to be in $\mathbb{F}$, a polynomial $f \in \mathbb{F}[X_1, \ldots, X_k]$ can be viewed as a function from $\mathbb{F}^k \to \mathbb{F}$. However, by taking the variables to be in an extension $\mathbb{G}$ of $\mathbb{F}$, f can also be viewed as function from $\mathbb{G}^k \to \mathbb{G}$.

For a set of polynomials $S = \{f_1, \ldots, f_m\} \subset \mathbb{F}_q[X_1, \ldots, X_k]$, the set $I = \{g_1 f_1 + \cdots + g_m f_m : g_i \in \mathbb{F}_q[X_1, \ldots, X_k]\}$ is called the *ideal* generated by S and is denoted $\langle f_1, \ldots, f_m \rangle$ or $\langle S \rangle$. In turn, S is called a *basis* for the ideal I.

The *variety* of an ideal I in field $\mathbb{G} \supset \mathbb{F}$ is denoted $\mathcal{V}_{\mathbb{G}}(I)$, and is the set $\{\mathbf{x} \in \mathbb{G}^k : \forall f \in I, f(\mathbf{x}) = 0\}$. That is, $\mathcal{V}_{\mathbb{G}}(I)$ contains the common zeros of polynomials in I, viewed as functions over $\mathbb{G}$. Note that for any set of polynomials S that generates I, $\mathcal{V}_{\mathbb{G}}(I)$ contains exactly the common zeros of S in $\mathbb{G}$. When the space $\mathbb{G}$ is just $\mathbb{F}$, we denote the variety as $\mathcal{V}(I)$. An ideal I that contains 1 contains all polynomials and is called *trivial*.

One can show that if I is trivial, then $\mathcal{V}(I) = \emptyset$. However, the converse does not hold. For instance, $X^2 + 1 \in \mathbb{F}_3[X]$ has no zeros in $\mathbb{F}_3$, but $1 \notin \langle X^2 + 1 \rangle$. But, one can also show that I is trivial iff for all extensions $\mathbb{G}$ of $\mathbb{F}$, $\mathcal{V}_{\mathbb{G}}(I) = \emptyset$.

The *field polynomial* for field $\mathbb{F}_q$ in variable X is $X^q - X$. Its zeros are all of $\mathbb{F}_q$ and it has no additional zeros in any extension of $\mathbb{F}_q$. Thus, for an ideal I of polynomials in $\mathbb{F}[X_1, \ldots, X_k]$ that contains field polynomials for each variable X_i, I is trivial iff $\mathcal{V}(I) = \emptyset$. For this reason, field polynomials are a common tool for ensuring the completeness of ideal-based reasoning techniques [48,54,97].

Representation. We represent $\mathbb{F}_p$ as the set of integers $\{0, 1, \ldots, p-1\}$, with the operations $+$ and $\times$ performed modulo p. The representation of $\mathbb{F}_{p^e}$ with $e > 1$ is more complex. Unfortunately, the set $\{0, 1, \ldots, p^e - 1\}$ with $+$ and $\times$ performed modulo p^e **is not** a field because multiples of p do not have multiplicative inverses. Instead, we represent $\mathbb{F}_{p^e}$ as the set of polynomials in $\mathbb{F}[X]$ of degree less than e. The operations $+$ and $\times$ are performed modulo $q(X)$, an irreducible polynomial[1] of degree e [71, Chapter 6]. There are p^e such polynomials, and so long as $q(X)$ is irreducible, all (save 0) have inverses. Note that this definition of $\mathbb{F}_{p^e}$ generalizes $\mathbb{F}_p$, and captures the fact that $\mathbb{F}_p \subset \mathbb{F}_{p^e}$.

2.2 Ideal Membership

The ideal membership problem is to determine whether a given polynomial p is in the ideal generated by a given set of polynomials D. We summarize definitions and facts relevant to algorithms for this problem; see [30] for a full presentation.

Monomial Ordering. In $\mathbb{F}[X_1, \ldots, X_k]$, a *monomial* is a polynomial of form $X_1^{e_1} \cdots X_k^{e_k}$ with non-negative integers e_i. A *monomial ordering* is a total ordering on monomials such that for all monomials p, q, r, if $p < q$, then $pr < qr$.

[1] Recall that an irreducible polynomial cannot be factored into two or more non-constant polynomials.

The *lexicographical* ordering for monomials $X_1^{e_1} \cdots X_k^{e_k}$ orders them lexicographically by the tuple $(e_1, \ldots, e_k)$. The *graded-reverse lexicographical* (grevlex) ordering is lexicographical by the tuple $(e_1 + \cdots + e_k, e_1, \ldots, e_k)$. With respect to an ordering, $\mathsf{lm}(f)$ denotes the greatest monomial of a polynomial f.

Reduction. For polynomials p and d, if $\mathsf{lm}(d)$ divides a term t of p, then we say that p *reduces* to r *modulo* d (written $p \to_d r$) for $r = p - \frac{t}{\mathsf{lm}(d)} d$. For a set of polynomials D, we write $p \to_D r$ if $p \to_d r$ for some $d \in D$. Let $\to_D^*$ be the transitive closure of $\to_D$. We define $p \Rightarrow_D r$ to hold when $p \to_D^* r$ and there is no r' such that $r \to_D r'$.

Reduction is a sound—but incomplete—algorithm for ideal membership. That is, one can show that $p \Rightarrow_D 0$ implies $p \in \langle D \rangle$, but the converse does not hold in general.

Gröbner Bases. Define the *s-polynomial* for polynomials p and q, by $\mathsf{spoly}(p, q) = p \cdot \mathsf{lm}(q) - q \cdot \mathsf{lm}(p)$. A Gröbner basis (GB) [21] is a set of polynomials P characterized by the following equivalent conditions:

1. $\forall p, p' \in P$, $\mathsf{spoly}(p, p') \Rightarrow_P 0$ (*closure under the reduction of s-polynomials*)
2. $\forall p \in \langle P \rangle$, $p \Rightarrow_P 0$ (*reduction is a complete test for ideal membership*)

Gröbner bases are useful for deciding ideal membership. From the first characterization, one can build algorithms for constructing a Gröbner basis for any ideal [21]. Then, the second characterization gives an ideal membership test. When P is a GB, the relation $\Rightarrow_P$ is a function (i.e., $\to_P$ is confluent), and it can be efficiently computed [1,21]; thus, this test is efficient.

A *Gröbner basis engine* takes a set of generators G for some ideal I and computes a Gröbner basis for I. We describe the high-level design of such engines here. An engine constructs a sequence of bases $G_0, G_1, G_2, \ldots$ (with $G_0 = G$) until some G_i is a Gröbner basis. Each G_i is constructed from G_{i-1} according to one of three types of steps. First, for some $p, q \in G_{i-1}$ such that $\mathsf{spoly}(p, q) \Rightarrow_{G_{i-1}} r \neq 0$, the engine can set $G_i = G_{i-1} \cup \{r\}$. Second, for some $p \in G_{i-1}$ such that $p \Rightarrow_{G_{i-1} \setminus \{p\}} r \neq p$, the engine can set $G_i = (G_{i-1} \setminus \{p\}) \cup \{r\}$. Third, for some $p \in G_{i-1}$ such that $p \Rightarrow_{G_{i-1} \setminus \{p\}} 0$, the engine can set $G_i = G_{i-1} \setminus \{p\}$. Notice that all rules depend on the current basis; some add polynomials, and some remove them. In general, it is unclear which sequence of steps will construct a Gröbner basis most quickly: this is an active area of research [1,18,41,43].

2.3 Zero Knowledge Proofs

Zero-knowledge proofs allow one to prove that some secret data satisfies a public property, without revealing the data itself. See [94] for a full presentation; we give a brief overview here. There are two parties: a *verifier* $\mathcal{V}$ and a *prover* $\mathcal{P}$. $\mathcal{V}$ knows a public *instance* x and asks $\mathcal{P}$ to show that it has knowledge of a secret *witness* w satisfying a public *predicate* $\phi(x, w)$. To do so, $\mathcal{P}$ runs an efficient (i.e., polytime in a security parameter λ) proving algorithm $\mathsf{Prove}(\phi, x, w) \to \pi$ and sends the resulting *proof* π to $\mathcal{V}$. Then, $\mathcal{V}$ runs an efficient verification

algorithm $\mathsf{Verify}(\phi, x, \pi) \to \{0, 1\}$ that accepts or rejects the proof. A system for Zero-Knowledge Proofs of knowledge (ZKPs) is a $(\mathsf{Prove}, \mathsf{Verify})$ pair with:

- *completeness*: If $\phi(x, w)$, then $\Pr[\mathsf{Verify}(\phi, x, \mathsf{Prove}(\phi, x, w)) = 0] \leq \mathsf{negl}(\lambda)$,[2]
- *computational knowledge soundness* [16]: (informal) a polytime adversary that does not know w satisfying ϕ can produce an acceptable π with probability at most $\mathsf{negl}(\lambda)$.
- *zero-knowledge* [50]: (informal) π reveals nothing about w, other than its existence.

ZKP applications are manifold. ZKPs are the basis of private cryptocurrencies such as Zcash and Monero, which have a combined market capitalization of $2.80B as of 30 June 2022 [44,45]. They've also been proposed for auditing sealed court orders [46], operating private gun registries [63], designing privacy-preserving middleboxes [53] and more [22,56].

This breadth of applications is possible because implemented ZKPs are very general: they support any ϕ checkable in polytime. However, ϕ must be first compiled to a cryptosystem-compatible computation language. The most common language is a *rank-1 constraint system* (R1CS). In an R1CS $\mathcal{C}$, x and w are together encoded as a vector $\mathbf{z} \in \mathbb{F}^m$. The system $\mathcal{C}$ is defined by three matrices $A, B, C \in \mathbb{F}^{n \times m}$; it is satisfied when $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}$, where $\circ$ is the element-wise product. Thus, the predicate can be viewed as n distinct *constraints*, where constraint i has form $(\sum_j A_{ij} z_j)(\sum_j B_{ij} z_j) - (\sum_j C_{ij} z_j) = 0$. Note that each constraint is a degree ≤ 2 polynomial in m variables that $\mathbf{z}$ must be a zero of. For security reasons, $\mathbb{F}$ must be large: its prime must have $\approx$255 bits.

Encoding. The efficiency of the ZKP scales quasi-linearly with n. Thus, it's useful to encode ϕ as an R1CS with a minimal number of constraints. Since equisatifiability—not logical equivalence—is needed, encodings may introduce new variables.

As an example, consider the Boolean computation $a \leftarrow c_1 \vee \cdots \vee c_k$. Assume that $c'_1, \ldots, c'_k \in \mathbb{F}$ are elements in $\mathbf{z}$ that are 0 or 1 such that $c_i \leftrightarrow (c'_i = 1)$. How can one ensure that $a' \in \mathbb{F}$ (also in $\mathbf{z}$) is 0 or 1 and $a \leftrightarrow (a' = 1)$? Given that there are $k - 1$ ORs, natural approaches use $\Theta(k)$ constraints. One clever approach is to introduce variable x' and enforce constraints $x'(\sum_i c'_i) = a'$ and $(1 - a')(\sum_i c'_i) = 0$. If any c_i is true, a' must be 1 to satisfy the second constraint; setting x' to the sum's inverse satisfies the first. If all c_i are false, the first constraint ensures a' is 0. This encoding is correct when the sum does not overflow; thus, k must be smaller than $\mathbb{F}$'s characteristic.

Optimizations like this can be quite complex. Thus, ZKP programmers use constraint synthesis libraries [14,69] or compilers [13,24,80,81,84,92,102] to generate an R1CS from a high-level description. Such tools support objects like Booleans, fixed-width integers, arrays, and user-defined data-types. The correctness of these tools is critical to the correctness of any system built with them.

[2] $f(\lambda) \leq \mathsf{negl}(\lambda)$ if for all $c \in \mathbb{N}$, $f(\lambda) = o(\lambda^{-c})$.

2.4 SMT

We assume usual terminology for many-sorted first order logic with equality ([38] gives a complete presentation). Let Σ be a many-sorted signature including a sort Bool and symbol family $\approx_\sigma$ (abbreviated $\approx$) with sort $\sigma \times \sigma \to \mathsf{Bool}$ for all σ in Σ. A *theory* is a pair $T = (\Sigma, \mathbf{I})$, where Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations. A Σ-formula ϕ is *satisfiable* (resp., *unsatisfiable*) in T if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. Given a (set of) formula(s) S, we write $S \models_T \phi$ if every interpretation $\mathcal{M} \in \mathbf{I}$ that satisfies S also satisfies ϕ.

When using the CDCL(T) framework for SMT, the reasoning engine for each theory is encapsulated inside a *theory solver*. Here, we mention the fragment of CDCL(T) that is relevant for our purposes ([78] gives a complete presentation)).

The goal of CDCL(T) is to check a formula ϕ for satisfiability. A *core* module manages a propositional search over the propositional abstraction of ϕ and communicates with the theory solver. As the core constructs partial propositional assignments for the abstract formula, the theory solver is given the literals that correspond to the current propositional assignment. When the propositional assignment is completed (or, optionally, before), the theory solver must determine whether its literals are jointly satisfiable. If so, it must be able to provide an interpretation in $\mathbf{I}$ (which includes an assignment to theory variables) that satisfies them. If not, it may indicate a strict subset of the literals which are unsatisfiable: an unsatisfiable core. Smaller unsatisfiable cores usually accelerate the propositional search.

3 The Theory of Finite Fields

We define the theory $T_{\mathbb{F}_q}$ of the finite field $\mathbb{F}_q$, for any order q. Its sort and symbols are indexed by the parameter q; we omit q when clear from context.

The signature of the theory is given in Fig. 1. It includes sort F, which intuitively denotes the sort of elements of $\mathbb{F}_q$ and is represented in our proposed SMT-LIB format as `(_ FiniteField q)`. There is a constant symbol for each element of $\mathbb{F}_q$, and function symbols for addition and multiplication. Other finite field operations (e.g., negation, subtraction, and inverses) naturally reduce to this signature.

An interpretation $\mathcal{M}$ of $T_{\mathbb{F}_q}$ must interpret: F as $\mathbb{F}_q$, $n \in \{0, \ldots, q-1\}$ as the n^{th} element of $\mathbb{F}_q$ in lexicographical order,[3] $+$ as addition in $\mathbb{F}_q$, $\times$ as multiplication in $\mathbb{F}_q$, and $\approx$ as equality in $\mathbb{F}_q$.

Note that in order to avoid ambiguity, we require that the sort of any constant `ff`n must be ascribed. For instance, the n^{th} element of $\mathbb{F}_q$ would be `(as ff`n `(_ FiniteField q))`. The sorts of non-nullary function symbols need not be ascribed: they can be inferred from their arguments.

[3] For non-prime $\mathbb{F}_{p^e}$, we use the lexicographical ordering of elements represented as polynomials in $\mathbb{F}_p[X]$ modulo the Conway polynomial [83,90] $C_{p,e}(X)$. This representation is standard [57].

Symbol	Arity	SMT-LIB	Description
$n \in \{0, \ldots, q-1\}$	F	`ff`n	The n^{th} element of $\mathbb{F}_q$
$+$	$\mathsf{F} \times \mathsf{F} \to \mathsf{F}$	`ff.add`	Addition in $\mathbb{F}_q$
$\times$	$\mathsf{F} \times \mathsf{F} \to \mathsf{F}$	`ff.mul`	Multiplication in $\mathbb{F}_q$

Fig. 1. Signature of the theory of $\mathbb{F}_q$

```
1  Function DecisionProcedure:
       Input: A set of 𝔽-literals L in variables X
       Output: UNSAT and a core C ⊆ L, or
       Output: SAT and a model M : X → 𝔽
2      P ← empty set; W_i ← fresh, ∀i;
3      for s_i ⋈_i t_i ∈ L do
4          if ⋈_i = ≈ then P ← P ∪ {[[s_i]] − [[t_i]]} ;
5          else if ⋈_i = ≉ then P ← P ∪ {W_i([[s_i]] − [[t_i]]) − 1} ;
6      B ← GB(P);
7      if 1 ⇒_B 0 then return UNSAT, CoreFromTree() ;
8      m ← FindZero(P);
9      if m = ⊥ then return UNSAT, L ;
10     else return SAT, {X ↦ z : (X ↦ z) ∈ m, X ∈ X} ;
```

Fig. 2. The decision procedure for $\mathbb{F}_q$.

4 Decision Procedure

Recall (§2.4) that a CDCL(T) theory solver for $\mathbb{F}$ must decide the satisfiability of a set of $\mathbb{F}$-literals. At a high level, our decision procedure comprises three steps. First, we reduce to a problem concerning a single algebraic variety. Second, we use a GB-based test for unsatisfiability that is fast and sound, but incomplete. Third, we attempt model construction. Figure 2 shows pseudocode for the decision procedure; we will explain it incrementally.

4.1 Algebraic Reduction

Let $L = \{\ell_1, \ldots, \ell_{|L|}\}$ be a set of literals. Each $\mathbb{F}$-literal has the form $\ell_i = s_i \bowtie t_i$ where s and t are $\mathbb{F}$-terms and $\bowtie \in \{\approx, \not\approx\}$. Let $\mathbf{X} = \{X_1, \ldots, X_k\}$ denote the free variables in L. Let $E, D \subseteq \{1, \ldots, |L|\}$ be the sets of indices corresponding to equalities and disequalities in L, respectively. Let $[\![t]\!] \in \mathbb{F}[\mathbf{X}]$ denote the natural interpretation of $\mathbb{F}$-terms as polynomials in $\mathbb{F}[\mathbf{X}]$ (Fig. 3). Let $P_E \subset \mathbb{F}[\mathbf{X}]$ be the set of interpretations of the equalities; i.e., $P_E = \{[\![s_i]\!] - [\![t_i]\!]\}_{i \in E}$. Let $P_D \subset \mathbb{F}[\mathbf{X}]$ be the interpretations of the *dis*equalities; i.e., $P_D = \{[\![s_i]\!] - [\![t_i]\!]\}_{i \in D}$. The satisfiability of L reduces to whether $\mathcal{V}(\langle P_E \rangle) \setminus \left[\bigcup_{p \in P_D} \mathcal{V}(\langle p \rangle)\right]$ is non-empty.

To simplify, we reduce disequalities to equalities using a classic technique [88]: we introduce a fresh variable W_i for each $i \in D$ and define P'_D as

$$P'_D = \{W_i([\![s_i]\!] - [\![t_i]\!]) - 1\}_{i \in D}$$

$$\text{Const}\frac{t \in \mathbb{F}}{[\![t]\!] = t} \quad \text{Var}\frac{}{[\![X_i]\!] = X_i} \quad \text{Add}\frac{[\![s]\!] = s' \quad [\![t]\!] = t'}{[\![s+t]\!] = s' + t'} \quad \text{Mul}\frac{[\![s]\!] = s' \quad [\![t]\!] = t'}{[\![s \times t]\!] = s' \times t'}$$

Fig. 3. Interpreting $\mathbb{F}$-terms as polynomials

Note that each $p \in P'_D$ has zeros for exactly the values of $\mathbf{X}$ where its analog in P_D is *not* zero. Also note that $P'_D \subset \mathbb{F}_q[\mathbf{X}']$, with $\mathbf{X}' = \mathbf{X} \cup \{W_i\}_{i \in D}$.

We define P to be $P_E \cup P'_D$ (constructed in lines 2 to 6, Fig. 2) and note three useful properties of P. First, L is satisfiable if and only if $\mathcal{V}(\langle P \rangle)$ is non-empty. Second, for any $P' \subset P$, if $\mathcal{V}(\langle P' \rangle) = \emptyset$, then $\{\pi(p) : p \in P'\}$ is an unsatisfiable core, where π maps a polynomial to the literal it is derived from. Third, from any $\mathbf{x} \in \mathcal{V}(\langle P \rangle)$ one can immediately construct a model. Thus, our theory solver reduces to understanding properties of the variety $\mathcal{V}(\langle P \rangle)$.

4.2 Incomplete Unsatisfiability and Cores

Recall (§2.2) that if $1 \in \langle P \rangle$, then $\mathcal{V}(\langle P \rangle)$ is empty. We can answer this ideal membership query using a Gröbner basis engine (line 7, Fig. 2). Let GB be a subroutine that takes a list of polynomials and computes a Gröbner basis for the ideal that they generate, according to some monomial ordering. We use grevlex: the ordering for which GB engines are typically most efficient [42]. We compute $GB(P)$ and check whether $1 \Rightarrow_{GB(P)} 0$. If so, we report that $\mathcal{V}(\langle P \rangle)$ is empty. If not, recall (§2.2) that $\mathcal{V}(\langle P \rangle)$ may still be empty; we proceed to attempt model construction (lines 9 to 11, Fig. 2, described in the next subsection).

If 1 *does* reduce by the Gröbner basis, then identifying a subset of P which is sufficient to reduce 1 yields an unsatisfiable core. To construct such a subset, we formalize the inferences performed by the Gröbner basis engine as a calculus for proving ideal membership.

Figure 4 presents IdealCalc: our ideal membership calculus. IdealCalc proves facts of the form $p \in \langle P \rangle$, where p is a polynomial and P is the set of generators for an ideal. The G rule states that the generators are in the ideal. The Z rule states that 0 is in the ideal. The S rule states that for any two polynomials in the ideal, their s-polynomial is in the ideal too. The $\mathsf{R}_\uparrow$ and $\mathsf{R}_\downarrow$ rules state that if $p \rightarrow_q r$ with q in the ideal, then p is in the ideal if and only if r is.

The soundness of IdealCalc follows immediately from the definition of an ideal. Completeness relies on the existence of algorithms for computing Gröbner bases using only s-polynomials and reduction [21,41,43]. We prove both properties in Appendix A.

Theorem 1 (IdealCalc**Soundness**). *If there exists an* IdealCalc *proof tree with conclusion* $p \in \langle P \rangle$*, then* $p \in \langle P \rangle$*.*

Theorem 2 (IdealCalc**Completeness**). *If* $p \in \langle P \rangle$*, then there exists an* IdealCalc *proof tree with conclusion* $p \in \langle P \rangle$*.*

$$\mathsf{Z}\ \frac{}{0 \in \langle P \rangle} \qquad \mathsf{G}\ \frac{p \in P}{p \in \langle P \rangle} \qquad \mathsf{R}_{\uparrow}\ \frac{r \in \langle P \rangle \quad q \in \langle P \rangle \quad p \to_q r}{p \in \langle P \rangle}$$

$$\mathsf{S}\ \frac{p \in \langle P \rangle \quad q \in \langle P \rangle}{\mathsf{spoly}(p,q) \in \langle P \rangle} \qquad \mathsf{R}_{\downarrow}\ \frac{p \in \langle P \rangle \quad q \in \langle P \rangle \quad p \to_q r}{r \in \langle P \rangle}$$

Fig. 4. IdealCalc: a calculus for ideal membership

```
1  Function FindZero:
       Input: A Gröbner basis B ⊂ 𝔽[X′]
       Input: A partial map M : X′ → 𝔽 (empty by default)
       Output: A total map M : X′ → 𝔽 or ⊥
2      if 1 ∈ ⟨B⟩ then return ⊥ ;
3      if |M| = |X′| then return M ;
4      for (X′_i ↦ z) ∈ ApplyRule(B, M) do
5          r ← FindZero(GB(B ∪ {X′_i − z}), M ∪ {X′_i ↦ z});
6          if r ≠ ⊥ then return r;
7      return ⊥
```

Fig. 5. Finding common zeros for a Gröbner basis. After handling trivial cases, *FindZero* uses *ApplyRule* to apply the first applicable rule from Fig. 6.

By instrumenting a Gröbner basis engine and reduction engine, one can construct IdealCalc proof trees. Then, for a conclusion $1 \in \langle P \rangle$, traversing the proof tree to its leaves gives a subset $P' \subseteq P$ such that $1 \in \langle P' \rangle$. The procedure *CoreFromTree* (called in line 8, Fig. 2) performs this traversal, by accessing a proof tree recorded by the *GB* procedure and the reductions. The proof of Theorem 2 explains our instrumentation in more detail (Appendix A).

4.3 Completeness Through Model Construction

As discussed, we still need a *complete* decision procedure for determining if $\mathcal{V}(\langle P \rangle)$ is empty. We call this procedure *FindZero*; it is a backtracking search for an element of $\mathcal{V}(\langle P \rangle)$. It also serves as our model construction procedure.

Figure 5 presents *FindZero* as a recursive search. It maintains two data structures: a Gröbner basis B and partial map $M : \mathbf{X}' \to \mathbb{F}$ from variables to field elements. By applying a branching rule (which we will discuss in the next paragraph), *FindZero* obtains a disjunction of single-variable assignments $X'_i \mapsto z$, which it branches on. *FindZero* branches on an assignment $X'_i \mapsto z$ by adding it to M and updating B to $GB(B \cup \{X'_i - z\})$.

Figure 6 shows the branching rules of *FindZero*. Each rule comprises *antecedents* (conditions that must be met for the rule to apply) and a *conclusion* (a disjunction of single-variable assignments to branch on). The Univariate rule applies when B contains a polynomial p that is univariate in some variable X'_i that M does not have a value for. The rule branches on the univariate roots of p. The Triangular rule comes from work on triangular decomposition [68]. It

$$\textsf{Univariate}\ \frac{p \in B \quad p \in \mathbb{F}[X_i'] \quad X_i' \notin M \quad Z \leftarrow \mathit{UnivariateZeros}(p)}{\bigvee_{z \in Z}(X_i' \mapsto z)}$$

$$\textsf{Triangular}\ \frac{\mathit{Dim}(\langle B \rangle) = 0 \quad X_i' \notin M \quad p \leftarrow \mathit{MinPoly}(B, X_i') \quad Z \leftarrow \mathit{UnivariateZeros}(p)}{\bigvee_{z \in Z}(X_i' \mapsto z)}$$

$$\textsf{Exhaust}\ \frac{}{\bigvee_{z \in \mathbb{F}} \bigvee_{X_i' \notin M}(X_i' \mapsto z)}$$

Fig. 6. Branching rules for *FindZero*.

applies when B is zero-dimensional.[4] It computes a univariate *minimal polynomial* $p(X_i')$ in some unassigned variables X_i', and branches on the univariate roots of p. The final rule Exhaust has no conditions and simply branches on all possible values for all unassigned variables.

FindZero's *ApplyRule* sub-routine applies the first rule in Fig. 6 whose conditions are met. The other subroutines (*GB* [21,41,43], *Dim* [11], *MinPoly* [2], and *UnivariateZeros* [87]) are commonly implemented in computer algebra libraries. *Dim*, *MinPoly*, and *UnivariateZeros* run in (randomized) polytime.

Theorem 3 (*FindZero***Correctness).** *If $\mathcal{V}(\langle B \rangle) = \emptyset$ then FindZero returns $\perp$; otherwise, it returns a member of $\mathcal{V}(\langle B \rangle)$. (Proof: Appendix B)*

Correctness and Efficiency. The branching rules achieve a careful balance between correctness and efficiency. The Exhaust rule is always applicable, but a full exhaustive search over a large field is unreasonable (recall: ZKPs operate of ≈255-bit fields). The Triangular and Univariate rules are important alternatives to exhaustion. They create a far smaller set of branches, but apply only when the variety has dimension zero or the basis has a univariate polynomial.

As an example of the importance of Univariate, consider the univariate system $X^2 = 2$, in a field where 2 is not a perfect square (e.g., $\mathbb{F}_7$). $X^2 - 2$ is already a (reduced) Gröbner basis, and it does not contain 1, so *FindZero* applies. With the Univariate rule, *FindZero* computes the univariate zeros of $X^2 - 2$ (there are none) and exits. Without it, the Exhaust rule creates $|\mathbb{F}|$ branches.

As an example of when Triangular is critical, consider

$$X_1 + X_2 + X_3 + X_4 + X_5 = 0$$
$$X_1X_2 + X_2X_3 + X_3X_4 + X_4X_5 + X_5X_1 = 0$$
$$X_1X_2X_3 + X_2X_3X_4 + X_3X_4X_5 + X_4X_5X_1 + X_5X_1X_2 = 0$$
$$X_1X_2X_3X_4 + X_2X_3X_4X_5 + X_3X_4X_5X_1 + X_4X_5X_1X_2 + X_5X_1X_2X_3 = 0$$
$$X_1X_2X_3X_4X_5 = 1$$

[4] The *dimension* of an ideal is a natural number that can be efficiently computed from a Gröbner basis. If the dimension is zero, then one can efficiently compute a minimal polynomial in any variable X, given a Gröbner basis [2,68].

in $\mathbb{F}_{394357}$ [68]. The system is unsatisfiable, it has dimension 0, and its ideal does not contain 1. Moreover, our solver computes a (reduced) Gröbner basis for it that does not contain any univariate polynomials. Thus, Univariate does not apply. However, Triangular does, and with it, *FindZero* quickly terminates. Without Triangular, Exhaust would create at least $|\mathbb{F}|$ branches.

In the above examples, Exhaust performs very poorly. However, that is not always the case. For example, in the system $X_1 + X_2 = 0$, using Exhaust to guess X_1, and then using the univariate rule to determine X_2 is quite reasonable. In general, Exhaust is a powerful tool for solving *underconstrained* systems. Our experiments will show that despite including Exhaust, our procedure performs quite well on our benchmarks. We reflect on its performance in Sect. 8.

Field Polynomials: A Road not Taken. By guaranteeing completeness through (potential) exhaustion, we depart from prior work. Typically, one ensures completeness by including *field polynomials* in the ideal (§2.2). Indeed, this is the approach suggested [97] and taken [55] by prior work. However, field polynomials induce enormous overhead in the Gröbner basis engine because their degree is so large. The result is a procedure that is only efficient for tiny fields [55]. In our experiments, we compare our system's performance to what it would be if it used field polynomials.[5] The results confirm that deferring completeness to *FindZero* is far superior for our benchmarks.

5 Implementation

We have implemented our decision procedure for prime fields in the cvc5 SMT solver [7] as a theory solver. It is exposed through cvc5's SMT-LIB, C++, Java, and Python interfaces. Our implementation comprises ≈2k lines of C++. For the algebraic sub-routines of our decision procedure (§4), it uses CoCoALib [1]. To compute unsatisfiable cores (§4.2), we inserted hooks into CoCoALib's Gröbner basis engine (17 lines of C++).

Our theory solver makes sparse use of the interface between it and the rest of the SMT solver. It acts only once a full propositional assignment has been constructed. It then runs the decision procedure, reporting either satisfiability (with a model) or unsatisfiability (with an unsatisfiable core).

6 Benchmark Generation

Recall that one motivation for this work is to enable translation validation for compilers to field constraint systems (R1CSs) used in zero-knowledge proofs (ZKPs). Our benchmarks are SMT formulas that encode translation validation queries for compilers from *Boolean* computations to R1CS. At a high level, each benchmark is generated as follows.

[5] We add field polynomials to our procedure on line 2, Fig. 2. This renders our ideal triviality test (lines 7 and 8) complete, so we can eliminate the fallback to *FindZero*.

1. Sample a Boolean formula Ψ in v variables with t non-variable terms.
2. Compile Ψ to R1CS using ZoKrates [36], CirC [81], or ZoK-CirC [81].
3. Optionally remove some constraints from the R1CS.
4. Construct a formula ϕ in `QF_FF` that tests the soundness (all assignments satisfying the R1CS agree with Ψ) or determinism (the inputs uniquely determine the output) of the R1CS.
5. Optionally encode ϕ in `QF_BV`, in `QF_NIA`, or as (Boolean-free) $\mathbb{F}$-equations.

Through step 3, we construct SMT queries that are satisfiable, unsatisfiable, and of unknown status. Through step 5, we construct queries solvable using bit-vector reasoning, integer reasoning, or a stand-alone computer algebra system.

6.1 Examples

We describe our benchmark generator in full and give the definitions of soundness and determinism in Appendix C. Here, we give three example benchmarks. Our examples are based on the Boolean formula $\Psi(x_1, x_2, x_3, x_4) = x_1 \vee x_2 \vee x_3 \vee x_4$. Our convention is to mark field variables with a prime, but not Boolean variables. Using the technique from Sect. 2.3, CirC compiles this formula to the two-constraint system: $i's' = r' \wedge (1-r')s' = 0$ where $s' \triangleq \sum_{i=0}^{3} x'_i$. Each Boolean input x_i corresponds to field element x'_i and r' corresponds to the result of Ψ.

Soundness. An R1CS is sound if it ensures the output r' corresponds to the value of Ψ (when given valid inputs). Concretely, our system is sound if the following formula is valid:

$$\underbrace{\forall i.(x'_i = 0 \vee x'_i = 1) \wedge (x'_i = 1 \iff x_i)}_{\text{inputs are correct}} \wedge \underbrace{i's' = r' \wedge (1-r')s' = 0}_{\text{constraints hold}}$$
$$\implies$$
$$\underbrace{(r'_i = 0 \vee r'_i = 1) \wedge (r'_i = 1 \iff \Psi)}_{\text{output is correct}}$$

where Ψ and s' are defined as above. This is an UNSAT benchmark, because the formula is valid.

Determinism. An R1CS is deterministic if the values of the inputs uniquely determine the value of the output. To represent this in a formula, we use two copies of the constraint system: one with primed variables, and one with double-primed variables. Our example is deterministic if the following formula is valid:

$$\underbrace{\forall i.(x'_i = x''_i)}_{\text{inputs agree}} \wedge \underbrace{i's' = r' \wedge (1-r')s' = 0 \wedge i''s'' = r'' \wedge (1-r'')s'' = 0}_{\text{constraints hold for both systems}}$$
$$\implies$$
$$\underbrace{r' = r''}_{\text{outputs agree}}$$

Unsoundness. Removing constraints from the system can give a formula that is not valid (a SAT benchmark). For example, if we remove $(1-r')s' = 0$, then the soundness formula is falsified by $\{x_i \mapsto \top, x_i' \mapsto 1, r' \mapsto 0, i' \mapsto 0\}$.

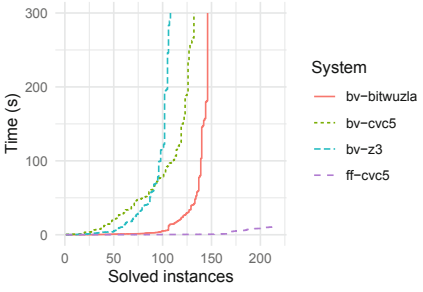
7 Experiments

Our experiments show that our approach:

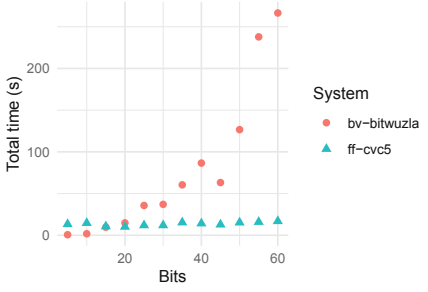
1. scales well with the size of $\mathbb{F}$ (unlike a BV-based approach),
2. would scale poorly with the size of $\mathbb{F}$ if field polynomials were used,
3. benefits from unsatisfiable cores, and
4. substantially outperforms all reasonable alternatives.

Our test bed is a cluster with Intel Xeon E5-2637 v4 CPUs. Each run is limited to one physical core, 8GB memory, and 300s.

Throughout, we generate benchmarks for two correctness properties (soundness and determinism), three different ZKP compilers, and three different statuses (sat, unsat, and unknown). We vary the field size, encoding, number of inputs, and number of terms, depending on the experiment. We evaluate our cvc5 extension, Bitwuzla (commit `27f6291`), and z3 (version 4.11.2).



(a) Instances solved



(b) Total solve time for (field-based) cvc5 and (BV-based) Bitwuzla on commonly solved instances at all bit-widths.

Fig. 7. The performance of field-based and BV-based approaches (with various BV solvers) when the field size ranges from 5 to 60 bits.

7.1 Comparison with Bit-Vectors

Since bit-vector solvers scale poorly with bit-width, one would expect the effectiveness of a BV encoding of our properties to degrade as the field size grows. To validate this, we generate BV-encoded benchmarks for varying bit-widths and evaluate state-of-the-art bit-vector solvers on them. Though our applications of interest use $b = 255$, we will see that the BV-based approach does not scale to

Table 1. Solved small-field benchmarks by tool, property, and status.

	determinism			soundness			total		
system	unsat	unk.	sat	unsat	unk.	sat	timeout	memout	solved
bv-bitwuzla	4	16	29	28	32	36	71	0	145
bv-cvc5	5	11	36	25	25	29	78	7	131
bv-z3	5	9	14	25	25	29	100	9	107
ff-cvc5	36	36	36	36	36	36	0	0	216
all benchmarks	36	36	36	36	36	36			216

fields this large. Thus, for this set of experiments we use $b \in \{5, 10, \ldots, 60\}$, and we sample formulas with 4 inputs and 8 intermediate terms.

Figure 7a shows performance of three bit-vector solvers (cvc5 [7], Bitwuzla [76], and z3 [73]) and our $\mathbb{F}$ solver as a cactus plot; Table 1 splits the solved instances by property and status. We see that even for these small bit-widths, the field-based approach is already superior. The bit-vector solvers are more competitive on the soundness benchmarks, since these benchmarks include only half as many field operations as the determinism benchmarks.

For our benchmarks, Bitwuzla is the most efficient BV solver. We further examine the time that it and our solver take to solve the 9 benchmarks they can both solve at all bit-widths. Figure 7b plots the total solve time against b. While the field-based solver's runtime is nearly independent of field size, the bit-vector solvers slow down substantially as the field grows.

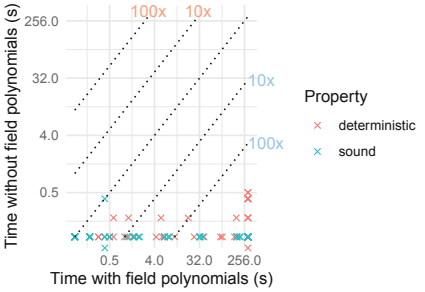
In sum, the BV approach scales poorly with field size and is already inferior on fields of size at least 2^{40}.

7.2 The Cost of Field Polynomials

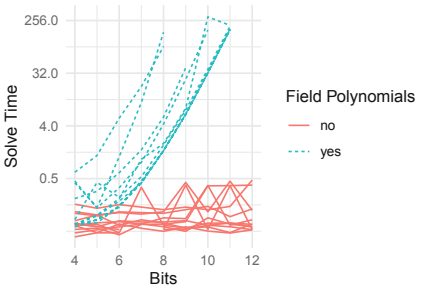
Recall that our decision procedure does not use field polynomials (§4.3), but our implementation optionally includes them (§5). In this experiment, we measure the cost they incur. We use propositional formulas in 2 variables with 4 terms, and we take $b \in \{4, \ldots, 12\}$, and include SAT and unknown benchmarks.

Figure 8a compares the performance of our tool with and without field polynomials. For many benchmarks, field polynomials cause a slowdown greater than 100×. To better show the effect of the field size, we consider the solve time for the SAT benchmarks, at varying values of b. Figure 8b shows how solve times change as b grows: using field polynomials causes exponential growth. For UNSAT benchmarks, both configurations complete within 1s. This is because (for these benchmarks) the GB is just $\{1\}$ and CoCoA's GB engine is good at discovering that (and exiting) without considering the field polynomials.

This growth is predictable. GB engines can take time exponential (or worse) in the degree of their inputs. A simple example illustrates this fact: consider computing a Gröbner basis with $X^{2^b} - X$ and $X^2 - X$. The former reduces to 0 modulo the latter, but the reduction takes $2^b - 1$ steps.

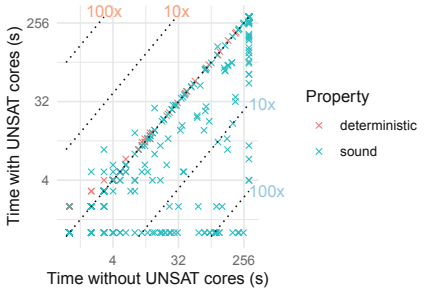


(a) All benchmarks, both configurations.

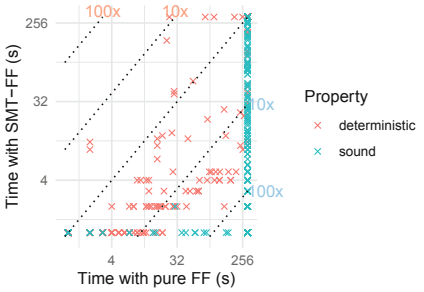


(b) Each series is one property at different numbers of bits.

Fig. 8. Solve times, with and without field polynomials. The field size varies from 4 to 12 bits. The benchmarks are all SAT or unknown.



(a) Our SMT solver with and without UNSAT cores.



(b) Our SMT solver compared with a pure computer algebra system.

Fig. 9. The performance of alternative algebra-based approaches.

7.3 The Benefit of UNSAT Cores

Section 4.2 describes how we compute unsatisfiable (UNSAT) cores in the $\mathbb{F}$ solver by instrumenting our Gröbner basis engine. In this experiment, we measure the benefit of doing so. We generate Boolean formulas with 2, 4, 6, 8, 10, and 12 variables; and 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, and 2^7 intermediate terms, for a 255-bit field. We vary the number of intermediate terms widely in order to generate benchmarks of widely variable difficulty. We configure our solver with and without GB instrumentation.

Figure 9a shows the results. For many soundness benchmarks, the cores cause a speedup of more than 10×. As expected, only the soundness benchmarks benefit. Soundness benchmarks have non-trivial boolean structure, so the SMT core makes many queries to the theory solver. Returning good UNSAT cores shrinks the propositional search space, reduces the number of theory queries, and thus reduces solve time. However, determinism benchmarks are just a conjunction

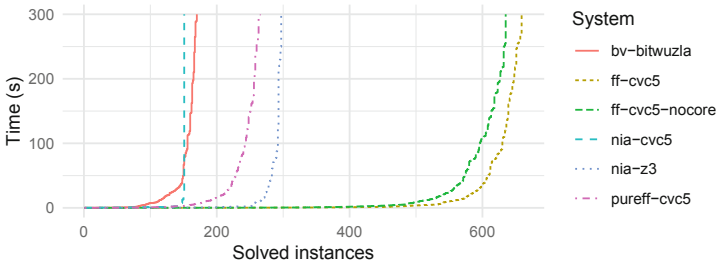


Fig. 10. A comparison of all approaches.

of theory literals, so the SMT core makes only one theory query. For them, returning a good UNSAT core has no benefit—but also induces little overhead.

7.4 Comparison to Pure Computer Algebra

In this experiment, we compare our SMT-based approach (which integrates computer-algebra techniques into SMT) against a stand-alone use of computer-algebra. We encode the Boolean structure of our formulas in $\mathbb{F}_p$ (see Appendix C). When run on such an encoding, our SMT solver makes just one query to its field solver, so it cannot benefit from the search optimizations present in CDCL(T). For this experiment, we use the same benchmark set as the last.

Figure 9b compares the pure $\mathbb{F}$ approach with our SMT-based approach. For benchmarks that encode soundness properties, the SMT-based approach is clearly dominant. The intuition here is is that computer algebra systems are not optimized for Boolean reasoning. If a problem has non-trivial Boolean structure, a cooperative approach like SMT has clear advantages. SMT's advantage is less pronounced for determinism benchmarks, as these manifest as a single query to the finite field solver; still, in this case, our encoding seems to have some benefit much of the time.

7.5 Main Experiment

In our main experiment, we compare our approach against all reasonable alternatives: a pure computer-algebra approach (§7.4), a BV approach with Bitwuzla (the best BV solver on our benchmarks, §7.1), an NIA approach with cvc5 and z3, and our own tool without UNSAT cores (§7.3). We use the same benchmark set as the last experiment; this uses a 255-bit field.

Figure 10 shows the results as a cactus plot. Table 2 shows the number of solved instances for each system, split by property and status. Bitwuzla quickly runs out of memory on most of the benchmarks. A pure computer-algebra approach outperforms Bitwuzla and cvc5's NIA solver. The NIA solver of z3 does a bit better, but our field-aware SMT solver is the best by far. Moreover, its best configuration uses UNSAT cores. Comparing the total solve time of ff-cvc5 and

Table 2. Solved benchmarks by tool, property, and status.

	determinism			soundness			total		
system	unsat	unk.	sat	unsat	unk.	sat	timeout	memout	solved
bv-bitwuzla	7	8	16	34	52	52	127	568	169
ff-cvc5	94	78	78	135	137	137	168	37	**659**
ff-cvc5-nocore	94	78	78	123	125	136	193	37	634
nia-cvc5	1	29	41	8	25	46	714	0	150
nia-z3	2	30	55	66	70	73	568	0	296
pureff-cvc5	84	74	75	6	15	10	532	68	264
all benchmarks	144	144	144	144	144	144			864

nia-z3 on commonly solved benchmarks, we find that ff-cvc5 reduces total solve time by 6×. In sum, the techniques we describe in this paper yield a tool that substantially outperforms all alternatives on our benchmarks.

8 Discussion and Future Work

We've presented a basic study of the potential of an SMT theory solver for finite fields based on computer algebra. Our experiments have focused on translation validation for ZKP compilers, as applied to Boolean input computations. The solver shows promise, but much work remains.

As discussed (Sect. 5), our implementation makes limited use of the interface exposed to a theory solver for CDCL(T). It does no work until a full propositional assignment is available. It also submits no lemmas to the core solver. Exploring which lightweight reasoning should be performed during propositional search and what kinds of lemmas are useful is a promising direction for future work.

Our model construction (Sect. 4.3) is another weakness. Without univariate polynomials or a zero-dimensional ideal, it falls back to exhaustive search. If a solution over an extension field is acceptable, then there are $\Theta(|\mathbb{F}|^d)$ solutions, so an exhaustive search seems likely to quickly succeed. Of course, we need a solution in the base field. If the base field is closed, then every solution is in the base field. Our fields are finite (and thus, not closed), but for our benchmarks, they seem to bear some empirical resemblance to closed fields (e.g., the GB-based test for an empty variety never fails, even though it is theoretically incomplete). For this reason, exhaustive search may not be completely unreasonable for our benchmarks. Indeed, our experiments show that our procedure is effective on our benchmarks, including for SAT instances. However, the worst-case performance of this kind of model construction is clearly abysmal. We think that a more intelligent search procedure and better use of ideas from computer algebra [6,67] would both yield improvement.

Theory combination is also a promising direction for future work. The benchmarks we present here are in the `QF_FF` logic: they involve only Booleans and finite

fields. Reasoning about different fields in combination with one another would have natural applications to the representation of elliptic curve operations inside ZKPs. Reasoning about datatypes, arrays, and bit-vectors in combination with fields would also have natural applications to the verification of ZKP compilers.

Acknowledgements. We appreciate the help and guidance of Andres Nötzli, Andy Reynolds, Anna Bigatti, Dan Boneh, Erika Ábrahám, Fraser Brown, Gregory Sankaran, Jacob Van Geffen, James Davenport, John Abbott, Leonardo Alt, Lucas Vella, Maya Sankar, Riad Wahby, Shankara Pailoor, and Thomas Hader.

This material is in part based upon work supported by the DARPA SIEVE program and the Simons foundation. Any opinions, findings, and conclusions or recommendations expressed in this report are those of the author(s) and do not necessarily reflect the views of DARPA. It is also funded in part by NSF grant number 2110397.

A Proofs of IdealCalc Properties

This appendix is available in the full version of the paper [82].

B Proof of Correctness for *FindZero*

We prove that *FindZero* is correct (Theorem 3).

Proof. It suffices to show that for each branching rule that results in $\bigvee_j (X_{i_j} - r_j)$,

$$\mathcal{V}(\langle B \rangle) \subset \bigcup_j \mathcal{V}(\langle B \cup \{X_{i_j} - r_j\} \rangle)$$

First, consider an application of Univariate with univariate $p(X_i)$. Fix $z \in \mathcal{V}(\langle B \rangle)$. z is a zero of p, so for some j, $r_j = z$ and $z \in \mathcal{V}(\langle B \cup \{X_i - z\} \rangle)$.

Next, consider an application of Triangular to variable X_i with minimal polynomial $p(X_i)$. By the definition of minimal polynomial, any zero z of $\langle B \rangle$ has a value for X_i that is a root of p. Let that root be r. Then, $z \in \mathcal{V}(\langle B \cup \{X_i - z\} \rangle)$.

Finally, consider an application of Exhaust. The desired property is immediate.

C Benchmark Generation

This appendix is available in the full version of the paper [82].

References

1. Abbott, J., Bigatti, A.M.: CoCoALib: A C++ library for computations in commutative algebra... and beyond. In: International Congress on Mathematical Software (2010)

2. Abbott, J., Bigatti, A.M., Palezzato, E., Robbiano, L.: Computing and using minimal polynomials. J. Symbolic Comput. **100** (2020)
3. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Logical Algebraic Methods in Programm. **119** (2021)
4. Anderson, B., McGrew, D.: Tls beyond the browser: Combining end host and network data to understand application behavior. In: IMC (2019)
5. Archer, D., O'Hara, A., Issa, R., Strauss, S.: Sharing sensitive department of education data across organizational boundaries using secure multiparty computation (2021)
6. Aubry, P., Lazard, D., Maza, M.M.: On the theories of triangular sets. J. Symbolic Comput. **28**(1) (1999)
7. Barbosa, H., et al.: cvc5: A versatile and industrial-strength SMT solver. In: TACAS (2022)
8. Barlow, R.: Computational thinking breaks a logjam. https://www.bu.edu/cise/computational-thinking-breaks-a-logjam/ (2015)
9. Barnett, M., Chang, B.Y.E., DeLine, R., Jacobs, B., Leino, K.R.M.: Boogie: A modular reusable verifier for object-oriented programs. In: FMCO (2005)
10. Barrett, C., et al.: CVC4. In: CAV (2011)
11. Bayer, D., Stillman, M.: Computation of hilbert functions. J. Symb. Comput. **14**(1), 31–50 (1992)
12. Bayless, S., Bayless, N., Hoos, H., Hu, A.: SAT modulo monotonic theories. In: AAAI (2015)
13. Baylina, J.: Circom. https://github.com/iden3/circom
14. bellman. https://github.com/zkcrypto/bellman
15. Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. Springer Science & Business Media (2013)
16. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 103–112 (1988)
17. Bogetoft, P., et al.: Secure multiparty computation goes live. In: FC (2009)
18. Bosma, W., Cannon, J., Playoust, C.: The magma algebra system i: the user language. J. Symb. Comput. **24**(3–4), 235–265 (1997)
19. Braun, D., Magaud, N., Schreck, P.: Formalizing some "small" finite models of projective geometry in coq. In: International Conference on Artificial Intelligence and Symbolic Computation (2018)
20. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: TACAS (2010)
21. Buchberger, B.: A theoretical basis for the reduction of polynomials to canonical forms. SIGSAM Bulletin (1976)
22. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: CCS (2017)
23. Caviness, B.F., Johnson, J.R.: Quantifier elimination and cylindrical algebraic decomposition. Springer Science & Business Media (2012)
24. Chin, C., Wu, H., Chu, R., Coglio, A., McCarthy, E., Smith, E.: Leo: A programming language for formally verified, zero-knowledge applications. Cryptology ePrint Archive (2021)

25. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. In: ACM TOCL **19**(3) (2018)
26. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: TACAS (2013)
27. Cimatti, A., Mover, S., Tonetta, S.: Smt-based verification of hybrid systems. In: AAAI (2012)
28. Cohen, C.: Pragmatic quotient types in coq. In: ITP (2013)
29. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: `SMT-RAT`: an open source `C++` toolbox for strategic and parallel SMT solving. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 360–368. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_26
30. Cox, D., Little, J., OShea, D.: Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra. Springer Science & Business Media (2013)
31. Davenport, J.: The axiom system (1992)
32. developers, M.: Monero technical specs. https://monerodocs.org/technical-specs/ (2022)
33. D'silva, V., Kroening, D., Weissenbacher, G.: A survey of automated techniques for formal software verification. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. **27**(7) (2008)
34. Dummit, D.S., Foote, R.M.: Abstract algebra, vol. 3. Wiley Hoboken (2004)
35. Dutertre, B.: Yices 2.2. In: CAV (2014)
36. Eberhardt, J., Tai, S.: ZoKrates–scalable privacy-preserving off-chain computations. In: IEEE Blockchain (2018)
37. Eisenbud, D., Grayson, D.R., Stillman, M., Sturmfels, B.: Computations in algebraic geometry with Macaulay 2, vol. 8. Springer Science & Business Media (2001)
38. Enderton, H.B.: A mathematical introduction to logic. Elsevier (2001)
39. Erbsen, A., Philipoom, J., Gross, J., Sloan, R., Chlipala, A.: Systematic Generation Of Fast Elliptic Curve Cryptography Implementations. Tech. rep, MIT (2018)
40. Erbsen, A., Philipoom, J., Gross, J., Sloan, R., Chlipala, A.: Simple high-level code for cryptographic arithmetic: With proofs, without compromises. ACM SIGOPS Oper. Syst. Rev. **54**(1) (2020)
41. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: ISSAC. ACM (2002)
42. Faugere, J.C., Gianni, P., Lazard, D., Mora, T.: Efficient computation of zero-dimensional Gröbner bases by change of ordering. J. Symb. Comput. **16**(4) (1993)
43. Faugére, J.C.: A new efficient algorithm for computing gröbner bases (f4). J. Pure Appl. Algebra **139**(1), 61–88 (1999)
44. Finance, Y.: Monero quote. https://finance.yahoo.com/quote/XMR-USD/ (2022) Accessed 30 June 2022
45. Finance, Y.: Zcash quote. https://finance.yahoo.com/quote/ZEC-USD/ (2022). Accessed 30 June 2022
46. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.: Practical accountability of secret processes. In: USENIX Security (2018)
47. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisfiability, Boolean Modeling and Comput. 1(3–4) (2006)
48. Gao, S.: Counting zeros over finite fields with Gröbner bases. Ph.D. thesis, Master's thesis, Carnegie Mellon University (2009)

49. GAP – Groups, Algorithms, and Programming, Version 4.13dev. www.gap-system.org (this year)
50. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: STOC (1985)
51. Gonthier, G., et al.: A machine-checked proof of the odd order theorem. In: ITP, pp. 163–179 (2013)
52. Greuel, G.M., Pfister, G., Schönemann, H.: Singular-a computer algebra system for polynomial computations. In: Symbolic computation and automated reasoning, pp. 227–233. AK Peters/CRC Press (2001)
53. Grubbs, P., Arun, A., Zhang, Y., Bonneau, J., Walfish, M.: {Zero-Knowledge} middleboxes. In: USENIX Security (2022)
54. Hader, T.: Non-Linear SMT-Reasoning over Finite Fields. Ph.D. thesis, TU Wien (2022), mS Thesis
55. Hader, T., Kovács, L.: Non-linear SMT-reasoning over finite fields. In: SMT (2022). http://ceur-ws.org/Vol-3185/extended3245.pdf extended Abstract
56. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (2020)
57. Heath, L.S., Loehr, N.A.: New algorithms for generating conway polynomials over finite fields. J. Symb. Comput. (2004)
58. Heck, A., Koepf, W.: Introduction to MAPLE, vol. 1993 (1993)
59. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification. https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf (2016)
60. Jovanović, D.: Solving nonlinear integer arithmetic with MCSAT. In: VMCAI (2017)
61. Jovanović, D., De Moura, L.: Solving non-linear arithmetic. ACM Commun. Comput. Algebra 46(3/4) (2013)
62. Jovanović, D., Moura, L.d.: Cutting to the chase solving linear integer arithmetic. In: CADE (2011)
63. Kamara, S., Moataz, T., Park, A., Qin, L.: A decentralized and encrypted national gun registry. In: IEEE S&P (2021)
64. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-aided reasoning: ACL2 case studies, vol. 4. Springer Science & Business Media (2013)
65. Komendantsky, V., Konovalov, A., Linton, S.: View of computer algebra data from coq. In: International Conference on Intelligent Computer Mathematics (2011)
66. Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J.: Coming of age: A longitudinal study of tls deployment. In: IMC (2018)
67. Lazard, D.: A new method for solving algebraic systems of positive dimension. Discr. Appl. Math. **33**, 1–3 (1991)
68. Lazard, D.: Solving zero-dimensional algebraic systems. J. Symb. Comput. **13**(2), 117–131 (1992)
69. libsnark. https://github.com/scipr-lab/libsnark
70. Maréchal, A., Fouilhé, A., King, T., Monniaux, D., Périn, M.: Polyhedral approximation of multivariate polynomials using handelman's theorem. In: VMCAI (2016)
71. McEliece, R.J.: Finite fields for computer scientists and engineers, vol. 23. Springer Science & Business Media (2012)
72. Meurer, A., et al.: Sympy: symbolic computing in python. PeerJ Comput. Sci. **3**, e103 (2017)

73. Moura, L.d., Bjørner, N.: Z3: An efficient smt solver. In: TACAS (2008)
74. Moura, L.d., Jovanović, D.: A model-constructing satisfiability calculus. In: VMCAI (2013)
75. Moura, L.d., Kong, S., Avigad, J., Doorn, F.v., Raumer, J.v.: The lean theorem prover (system description). In: CADE (2015)
76. Niemetz, A., Preiner, M.: Bitwuzla at the SMT-COMP 2020. arXiv:2006.01621 (2020)
77. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC and Boolector 3.0. In: CAV (2018)
78. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract davis-putnam-logemann-loveland procedure to DPLL(T). J. ACM (2006)
79. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
80. Noir. https://noir-lang.github.io/book/index.html
81. Ozdemir, A., Brown, F., Wahby, R.S.: Circ: Compiler infrastructure for proof systems, software verification, and more. In: IEEE S&P (2022)
82. Ozdemir, A., Kremer, G., Tinelli, C., Barrett, C.: Satisfiability modulo finite fields (2023), https://eprint.iacr.org/2023/091, (Full version)
83. Parker, R.: Finite fields and conway polynomials (1990), talk at the IBM Heidelberg Scientific Center. Cited by Scheerhorn
84. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. Commun. ACM **59**(2), 103–112 (2016)
85. Philipoom, J.: Correct-by-construction finite field arithmetic in Coq. Ph.D. thesis, Massachusetts Institute of Technology (2018)
86. Pnueli, A., Siegel, M., Singerman, E.: Translation validation. In: TACAS (1998)
87. Rabin, M.O.: Probabilistic algorithms in finite fields. SIAM Journal on computing 9(2) (1980)
88. Rabinowitsch, J.L.: Zum hilbertschen nullstellensatz. Mathematische Annalen 102 (1930), https://doi.org/10.1007/BF01782361
89. Sasson, E.B., et al.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE S&P (2014)
90. Scheerhorn, A.: Trace-and Norm-compatible Extensions of Finite Fields. Applicable Algebra in Engineering, Communication and Computing (1992)
91. Schwabe, P., Viguier, B., Weerwag, T., Wiedijk, F.: A coq proof of the correctness of x25519 in tweetnacl. In: CSF (2021)
92. Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: Proceedings of the 8th ACM European Conference on Computer Systems, pp. 71–84 (2013)
93. Shankar, N.: Automated deduction for verification. CSUR **41**(4) (2009)
94. Thaler, J.: Proofs, Arguments, and Zero-Knowledge (2022)
95. Torlak, E., Bodik, R.: A lightweight symbolic virtual machine for solver-aided host languages. In: PLDI (2014)
96. Tung, V.X., Khanh, T.V., Ogawa, M.: raSAT: An smt solver for polynomial constraints. In: IJCAR (2016)
97. Vella, L., Alt, L.: On satisfiability of polynomial equations over large prime field. In: SMT (2022). http://ceur-ws.org/Vol-3185/extended9913.pdf extended Abstract
98. Weispfenning, V.: Quantifier elimination for real algebra-the quadratic case and beyond. Appl. Algebra Eng., Commun. Comput. **8**(2) (1997)

99. Wen-Tsún, W.: A zero structure theorem for polynomial-equations-solving and its applications. In: European Conference on Computer Algebra (1987)
100. Wolfram, S.: Mathematica: a system for doing mathematics by computer. Addison Wesley Longman Publishing Co., Inc. (1991)
101. Zimmermann, P., et al.: Computational mathematics with SageMath. SIAM (2018)
102. Zinc. https://zinc.matterlabs.dev/