EPiC Series in Computing
Volume 94, 2023, Pages 238–256
Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

SMT Solving over Finite Field Arithmetic

Thomas Hader, Daniela Kaufmann, and Laura Kovács

TU Wien, Vienna, Austria
{thomas.hader,daniela.kaufmann,laura.kovacs}@tuwien.ac.at

Abstract

Non-linear polynomial systems over finite fields are used to model functional behavior of cryptosystems, with applications in system security, computer cryptography, and post-quantum cryptography. Solving polynomial systems is also one of the most difficult problems in mathematics. In this paper, we propose an automated reasoning procedure for deciding the satisfiability of a system of non-linear equations over finite fields. We introduce zero decomposition techniques to prove that polynomial constraints over finite fields yield finite basis explanation functions. We use these explanation functions in model constructing satisfiability solving, allowing us to equip a CDCL-style search procedure with tailored theory reasoning in SMT solving over finite fields. We implemented our approach and provide a novel and effective reasoning prototype for non-linear arithmetic over finite fields.

1 Introduction

Solving a system of polynomial equations is one of the hardest problems in mathematics, with emerging applications in cryptography, software security, code optimizations, control theory, and many other areas of computer science. Computing solutions to polynomial equations is known to be decidable and algorithmically solvable over algebraically closed fields thanks to the fundamental theory of algebra and Buchberger's algorithms for Gröbner basis computation [8,37]. Yet, when restricting the algorithmic study of solving polynomial equations over integers, the problem becomes undecidable [29].

Until recently, the algorithmic study of solving polynomial constraints, and hence automated reasoning in polynomial arithmetic, was the sole domain of computer algebra systems [1,28,36,43]. These systems are very powerful in computing the set of all solutions of polynomial constraints, but generally suffer from high computational overhead, such as doubly exponential computation complexities in terms of number of variables [11].

With the purpose of scaling non-linear reasoning, especially for solving satisfiability instances of polynomial arithmetic, exciting new developments in boolean satisfiability (SAT)/satisfiability modulo theory (SMT) reasoning arose by combining a Conflict-Driven Clause Learning (CDCL)-style search for a feasible assignment, called Model Constructing Satisfiability (MCSat), with algebraic decompositions and projections over the solution space of polynomial inequalities [12,25]. Unlike the classic CDCL(T) approach of SMT-solvers, MCSat [12,24,25] combines the capabilities of a SAT solver and a theory solver into a single procedure while keeping the search principles theory independent. To the best of our knowledge, SMT solving over finite fields lacks a dedicated approach for reasoning over finite fields. Encoding the problem in existing theories (e.g. NIA) are inefficient [33].

In this paper we address this challenge and introduce a CDCL-style search procedure extended with zero decomposition techniques for explaining and resolving (variable) conflicts while solving polynomial constraints over finited fields.

Need for Finite Fields. Finite fields provide natural ground to model bounded machine arithmetic, for example when considering modern cryptosystems with applications in system security and post-quantum cryptography. Existing approches build for example private and secure systems from Zero-Knowledge Proofs [18] or verify blockchain technologies, such as smart contracts [38], with all these efforts implementing finite field arithmetic. Elliptic curve cryptography [21] exploits polynomials over finite fields, with further use in TLS encryption [30], SSH [35] and digital signatures [23]. Polynomial equations over finite fields are also used in coding theory [27,31], decoding error-correcting codes of large error rates. In addition, solving polynomials over finite fields has applications in finite biological models, such as modeling cycles of biological networks as continuous dynamical systems [31,32].

SMT Solving over Finite Fields. In this paper we introduce an MCSat-based decision procedure for solving polynomial constraints over finite fields, extending thus the landscape of SMT solving with finite field arithmetic. We formalize *SMT solving over finite fields* as follows (see Section 2 for relevant notation).

Given a finite field $\mathbb{F}_q$ with order $q = p^k$, where p is a prime number and $k \geq 1$, let F be a set of polynomial constraints in $\mathbb{F}_q[X]$ and $\mathcal{F}$ a formula following the logical structure:

$$\mathcal{F} \quad = \quad \bigwedge_{C \subseteq F} \bigvee_{f \in C} f \quad = \quad \bigwedge_{C \subseteq F} \bigvee_{f \in C} \mathsf{poly}(f) \rhd 0 \quad \text{with } \rhd \in \{=, \neq\}.$$

SMT solving over finite fields: Does an assignment $\nu : \{x_1, \ldots, x_n\} \rightarrow \mathbb{F}_q$ exists that satisfies $\mathcal{F}$?

Example 1. *We show an instance of the SMT solving problem over finite fields, by considering the finite field $\mathbb{F}_5$ whose elements are $\{0, 1, 2, 3, 4\}$. Note that -1 is 4 in $\mathbb{F}_5$. Let $\mathcal{F}$ be the formula representing the conjunction of the polynomial constraints $\{x_1^2 - 1 = 0, x_1x_2 - x_2 - 1 = 0\}$ over $\mathbb{F}_5[x_1, x_2]$. In our work we address SMT solving of $\mathcal{F}$ over $\mathbb{F}_5[x_1, x_2]$, deriving that $\mathcal{F}$ is satisfiable using the variable assignment $\{x_1 \mapsto 4, x_2 \mapsto 2\}$.*

To the best of our knowledge, existing SMT-based approaches lack the necessary theory for reasoning over finite fields, and therefore assertions that model the behavior of finite fields must be included in the input problem formalization (i.e. F). As a workaround, one may use so-called *field polynomials* ($\{x_k^q - x_k \mid 1 \leq k \leq n\}$ for a ring $\mathbb{F}_q[X]$) to characterize finite fields and thus restrict the solution space of $\mathbb{F}_q[X]$ to the finite domain of the field $\mathbb{F}_q$. Unfortunately, using field polynomials is practically inefficient, as already witnessed in our initial attempts from [19,20]: when used during variable elimination, field polynomials yield new polynomials as logical consequences of the initial set F of polynomials and at the same time hugely increase the degree and size of the newly derived polynomials in the search space.

Our contributions. In this paper we do not rely on field polynomials but extend the theory-dependent rules of MCSat to natively support finite fields arithmetic. The main difficulty in MCSat-based reasoning comes with generating so-called *explanation clauses* for resolving conflicting variable assignments during SMT solving. We therefore develop a novel *theory propagation* rule for finite fields that admits propagation of theory literals (Section 4). Our method exploits zero decomposition techniques [40] to prove that polynomial constrains over finite fields yield finite basis explanation functions (Theorem 2), implying computabilty of such functions. We use single polynomial projections and adjust subresultant regular subchains [42] to calculate greatest common divisors with regard to partial variable assignments (Section 5), allowing us to avoid the use of field polynomials when deriving explanation clauses during solving polynomial constraints (Theorems 3–4). Our explanation clauses are integrated within MCSat, restricting the search space of SMT solving over $\mathbb{F}_q[X]$. We implement our approach in a new prototype for SMT solving over finite fields (Section 6) and experimentally demonstrate the applicability of SMT solving over finite fields (Section 7).

2 Preliminaries

We provide a brief summary of the relevant algebraic concepts of finite fields [15].

Fields and Polynomials. A *field* $\mathbb{F}$ consists of a set S on which two binary operators addition "+" and multiplication "·" are defined. Both operators are commutative, associative, have a neutral element in S (denoted as *zero* (0) and *one* (1), respectively), and each element in S has additive and multiplicative inverses. Furthermore, distributivity holds. Informally speaking, a *field* is a set S with well-defined operations addition, subtraction, multiplication, and division (with the exception of division by zero). Field examples include $\mathbb{Q}$ and $\mathbb{R}$.

Let X be the set of variables $\{x_1, \dots, x_n\}$. We sort the variables in X according to their index $x_1 < x_2 < \cdots < x_n$. Since x_i is the i-th variable in the order, we say it is of of *class* i, denoted by $\mathsf{cls}(x_i) = i$. We have $X_k = \{x_i \in X \mid i \leq k\}$.

By $\mathbb{F}[X]$ we denote the ring of polynomials in variables X with coefficients in $\mathbb{F}$. A *term* $\tau = x_1^{d_1} \cdots x_n^{d_n}$ is a product of powers of variables for $d_i \in \mathbb{N}$. If all $d_i = 0$, we have $\tau = 1$. A multiple of a term $c\tau$ with $c \in \mathbb{F} \setminus \{0\}$ is a *monomial*. A *polynomial* is a finite sum of monomials with pairwise distinct terms.

The *degree of a term* τ is the sum of its exponents $\sum_{i=0}^{n} d_n$. The *degree of a polynomial* p is the highest degree of its terms. We write $\mathsf{deg}(p, x_i)$ to denote the highest *degree of* x_i in p.

For a polynomial p, the set of variables of p is denoted by $\mathsf{vars}(p)$. If $\mathsf{vars}(p) = \emptyset$, then p is *constant*. If $|\mathsf{vars}(p)| = 1$, p is *univariate*, and otherwise it is *multivariate*. For a set of polynomials P, we define $\mathsf{vars}(P) = \bigcup_{p \in P} \mathsf{vars}(p)$.

An order $\leq$ is fixed on the set of terms such that for all terms τ, σ_1, σ_2 it holds that $1 \leq \tau$ and further $\sigma_1 \leq \sigma_2 \Rightarrow \tau\sigma_1 \leq \tau\sigma_2$. One such order is the *lexicographic term order*: If $x_1 < x_2 < \dots x_n$, then for two terms $\sigma_1 = x_1^{d_1} \cdots x_n^{d_n}$, $\sigma_2 = x_1^{e_1} \cdots x_n^{e_n}$ it holds $\sigma_1 < \sigma_2$ iff there exists an index i with $d_j = e_j$ for all $j > i$, and $d_i < e_i$.

For a polynomial p, the *leading variable* $\mathsf{lv}(p)$ is the variable x_i of $\mathsf{vars}(p)$ with the highest class. Let $\mathsf{cls}(p) = \mathsf{cls}(\mathsf{lv}(p))$. We define the coefficient of $x_i^{\mathsf{deg}(p,x_i)}$ as the *leading coefficient* of p with respect to x_i and write it as $\mathsf{lc}(p, x_i)$. We denote $\mathsf{red}(p, x_i) = p - \mathsf{lc}(p, x_i)x_i^{\mathsf{deg}(p,x_i)}$ as the *reductum* of p with respect to x_i.

Example 2. *Given the polynomial* $p = 2x_3^2x_1 + 4x_3x_2^4 + x_3x_2 + 7x_1 \in \mathbb{Q}[x_1, x_2, x_3]$, *we have* $\mathsf{vars}(p) = \{x_1, x_2, x_3\}$, $\mathsf{lv}(p) = x_3$ *and* $\mathsf{red}(p, x_3) = 4x_3x_2^2 + x_3x_2 + 7x_1$. *Furthermore,* $\mathsf{lc}(p, x_1) = 2x_3^2 + 7$, $\mathsf{lc}(p, x_2) = 4x_3$, *and* $\mathsf{deg}(p, x_3) = 2$, $\mathsf{deg}(p, x_2) = 4$.

A polynomial $p \in \mathbb{F}[X]$ is *irreducible* if it cannot be represented as the product of two non-constant polynomials, i.e. there exist no $q, r \in \mathbb{F}[X]$ such that $p = q \cdot r$. A polynomial $g \in \mathbb{F}[X]$ is called a *greatest common divisor (gcd)* of polynomials $p_1, \ldots, p_s$ if g divides $p_1, \ldots, p_s$ and every common divisor of $p_1, \ldots, p_s$ divides g.

A tuple of values $\alpha \in \mathbb{F}^n$ is a *root* or *zero* of a polynomial $p \in \mathbb{F}[X]$ if $p(\alpha) = 0$. A field $\mathbb{F}$ is *algebraically closed* if every non-constant univariate polynomial in $\mathbb{F}[x]$ has a root in $\mathbb{F}$.

Let $\mathbb{K} \subseteq \mathbb{F}$ be a field with respect to the field operations inherited from $\mathbb{F}$. We call $\mathbb{F}$ a *field extension* of $\mathbb{K}$ and write $\mathbb{F}/\mathbb{K}$. An *algebraic extension* of $\mathbb{F}$ is a field extension $\mathbb{G}/\mathbb{F}$ such that every element of $\mathbb{G}$ is a root of a non-zero polynomial with coefficients in $\mathbb{F}$. An *algebraic closure* of a field $\mathbb{F}$ is an algebraic extension $\mathbb{G}$ that is algebraically closed; we call $\mathbb{F}$ the *base field*.

Finite Fields. In a *finite field* $\mathbb{F}_q$ the set S has only finitely many elements. The number of elements is denoted by q and is called the *order* of the finite field. We denote the algebraic closure of $\mathbb{F}_q$ as $\overline{\mathbb{F}}_q$. A finite field $\mathbb{F}_q$ exists iff q is the k-th power of a prime p, i.e. $q = p^k$. All finite fields with the same order are isomorphic, i.e. there exists a structure-preserving mapping between them. In case $k = 1$, $\mathbb{F}_q$ can be represented by the integers modulo p and we have $S = \{0, 1, \ldots, p-1\}$ with the standard integer addition and multiplication operation performed modulo p. For example for $\mathbb{F}_5$ we have $S = \{0, 1, 2, 3, 4\}$ and $2 + 3 = 0$ and $3 \cdot 4 = 2$.

The elements of $\mathbb{F}_q = \mathbb{F}_{p^k}$ with $k > 1$ are polynomials with degree $k-1$ and coefficients in $\mathbb{F}_p$. Addition and multiplication of the polynomials is performed modulo a univariate irreducible polynomial $g \in \mathbb{F}_q[a]$ with degree k. For example $\mathbb{F}_4 = \mathbb{F}_{2^2}$ is generated using the irreducible polynomial g: $a^2 + a + 1$. The elements are $\{0, 1, a, 1 + a\}$ and $((a) + (1)) \cdot (a + 1)$ evaluates to a.

Polynomial Constraints and Formulas. A *polynomial constraint* f *over* p in the ring $\mathbb{F}_q[X]$ is of the form $p \rhd 0$ where $p \in \mathbb{F}_q[X]$ and $\rhd \in \{=, \neq\}$. Since a total ordering with respect to the field operations on elements of a finite field $\mathbb{F}_q$ does not exist, we only consider inequality constraints of the form $p \neq 0$ and do not consider $<$ and $>$. We define $\mathsf{poly}(f) = p$, and extend $\mathsf{vars}(f) = \mathsf{vars}(\mathsf{poly}(f))$ and $\mathsf{cls}(f) = \mathsf{cls}(\mathsf{poly}(f))$. For a set of constraints F we define $\mathsf{vars}(F) = \bigcup_{f \in F} \mathsf{vars}(f)$. A polynomial constraint f is negated by substituting $\rhd$ in f with the other element, i.e. $\neg(\mathsf{poly}(f) = 0)$ is equivalent to $\mathsf{poly}(f) \neq 0$.

Let $\nu : X \to \mathbb{F}_q$ denote an (partial) *assignment* of variables X. We extend ν to an evaluation of polynomials in the natural way, i.e. $\nu(p) : \mathbb{F}_q[X] \to \mathbb{F}_q$. Given an assignment ν and a polynomial constraint $f = p \rhd 0$, we say ν *satisfies* f iff $\nu(p) \rhd 0$ holds. The function ν is also used to evaluate a constraint f. If ν does not assign all variables of $\mathsf{poly}(f)$, we define $\nu(f) = \mathsf{undef}$. If ν assigns all variables of $\mathsf{poly}(f)$, then $\nu(f) = \mathsf{true}$ if ν satisfies f, and $\nu(f) = \mathsf{false}$ otherwise. Given a set of polynomial constraints F, we have $\nu(F) = \mathsf{true}$ iff ν satisfies all elements in F. If such an ν exists, we say that F is *satisfiable* and ν *satisfies* F.

We refer to a single constraint as an *atom*. A *literal* is an atom or its negated form. A *clause* C is a disjunction of literals. If C contains only one literal it is a *unit clause*. A *formula* $\mathcal{F}$ is a set of clauses $\mathcal{C}$. Logically a formula represents a conjunction of disjunctions of literals. An *assignment* ν satisfies a clause C if at least one literal in C is satisfied by ν. Finally, ν satisfies a set of clauses if every clause is satisfied by ν.

3 Model Constructing Satisfiability (MCSat)

In this section, we summarize the MCSat approach [12, 24, 25] as presented in [25]. Our MCSat adjustments for finite fields are given in Sections 4 and 5.

MCSat Terminology. The MCSat procedure is a transition system with each state denoted by an indexed pair $\langle M, \mathcal{C}\rangle_k$ of a *trail* M and a set of clauses $\mathcal{C}$. The index k specifies the *level* of the state. In our case, a clause $C \in \mathcal{C}$ is a set of polynomial constraints over $\mathbb{F}_q[X]$. We require the following terminology:

– Each *trail element* of M is either a *decided* or *propagated literal*, or a *variable assignment*.
– A decided literal f is considered to be true. A propagated literal, indicated $C \to f$, denotes that the status of clause C implies that f is true. A variable assignment $x_i \mapsto \alpha$ maps a theory variable $x_i \in X$ to some $\alpha \in \mathbb{F}_q$.
– We say $f \in M$, if a constraint f is a trail element. Let $\mathsf{constr}(M) = \{f \in M\}$.
– A trail is *non-redundant* if it contains each constraint at most once.
– For a constraint f let $\mathsf{level}(f) = i \Leftrightarrow x_i \in \mathsf{vars}(f) \wedge \forall j > i : x_j \notin \mathsf{vars}(f)$ and define the level of a clause $\mathsf{level}(C) = \max_{f \in C} \mathsf{level}(f)$. Let $\mathcal{C}_i = \{C \in \mathcal{C} \mid \mathsf{level}(C) \leq i\}$.
– We have $\mathsf{level}(M) = k$, if $x_{k-1} \mapsto \alpha$ is the highest variable assignment in M, i.e. no variable assignment for $x_k, \ldots, x_n$ exists.
– A trail is *increasing in level*, if all variables but the highest level variable of a constraint f are assigned before f appears on the trail.
– Taking all theory variable assignments $x_1 \mapsto \alpha_1, \ldots, x_k \mapsto \alpha_k$ of a trail M with $\mathsf{level}(M) = k+1$, we define $\boldsymbol{\alpha}_M = \alpha_1, \ldots, \alpha_k$ and generate a (partial) assignment function $\nu_M : X \mapsto \mathbb{F}_q$. We overload ν_M to evaluate constraints and sets of constraints as discussed in Section 2.
– We further say that M is *feasible* if $\nu_M(\mathsf{constr}(M))$ has a solution for x_k. The set of possible values for x_k is denoted by $\mathsf{fsbl}(M)$.
– Given an additional constraint f, with $\mathsf{poly}(f) \in \mathbb{F}_q[X]$, we extend $\mathsf{fsbl}(f, M) = \mathsf{fsbl}([\![M, f]\!])$. If $\mathsf{fsbl}(f, M) \neq \emptyset$ we say that f is *compatible* with M, denoted by $\mathsf{comp}(f, M)$.
– A state $\langle M, \mathcal{C}\rangle_k$ is *well-formed* when M is non-redundant, increasing in level, $\mathsf{level}(M) = k$, $\mathsf{fsbl}(M) \neq \emptyset$, ν_M satisfies $\mathcal{C}_{k-1}$, $\forall f \in \mathsf{constr}(M) : \nu_M(f) = \mathsf{true}$, and all propagated literals $E \to f$ are implied, i.e. $f \in E$ and for all literals $f' \neq f$ in E, $\nu_M(f') = \mathsf{false}$ or $\neg f' \in \mathsf{constr}(M)$.
– Given a well-formed state with trail M, assume constraint f with $\mathsf{poly}(f) \in \mathbb{F}_q[X_k]$. Let:

$$\mathsf{val}(f, M) = \begin{cases} \nu_M(f) & x_k \notin \mathsf{vars}(f), \mathsf{level}(M) = k \\ \mathsf{true} & f \in \mathsf{constr}(M) \\ \mathsf{false} & \neg f \in \mathsf{constr}(M) \\ \mathsf{undef} & \text{otherwise} \end{cases}$$

We overload this function to handle clauses. As such, we define $\mathsf{val}(C, M) = \mathsf{true}$ if there exists $f \in C$ such that $\mathsf{val}(f, M) = \mathsf{true}$; $\mathsf{val}(C, M) = \mathsf{false}$ if $\mathsf{val}(f, M) = \mathsf{false}$ for al $f \in C$; and $\mathsf{val}(C, M) = \mathsf{undef}$ in all other cases.

MCSat Calculus. The MCSat calculus is given in Figure 1 and detailed next. Given a set of clauses $\mathcal{C}$, in our case clauses of polynomial constraints over $\mathbb{F}_q[X]$, the goal is to move from an initial state of $\langle [\![\,]\!], \mathcal{C}\rangle_1$ to one of the two termination states, namely $\langle \mathsf{sat}, \nu\rangle$ or unsat, by continuously applying transition rules. A termination and correctness proof that is independent of the used theory, is given in [25, Thm. 1].

The search rules either select a clause for further processing (Sel-Clause), detect a conflict (Conflict), detect satisfiability (Sat), or assign a variable while increasing the level before performing another search step (Lift-Level).

Clause satisfaction rules determine how a clause C is absorbed into the trail M given a state $\langle M, \mathcal{C}\rangle_k \vDash C$ through semantic reasoning on the theory. The first two rules are similar to classical DPLL-style propagation and differ in whether we meet a single compatible literal (B-Prop) or can choose between multiple yet undetermined compatible literals (Decide-Lit).

Search Rules

[Sel-Clause]	$\langle M,\mathcal{C}\rangle_k \rightarrow \langle M,\mathcal{C}\rangle_k \vDash C$	$C \in \mathcal{C}_k \wedge \mathsf{val}(C,M) = \mathsf{undef}$
[Conflict]	$\langle M,\mathcal{C}\rangle_k \rightarrow \langle M,\mathcal{C}\rangle_k \vdash C$	$C \in \mathcal{C}_k \wedge \mathsf{val}(C,M) = \mathsf{false}$
[Sat]	$\langle M,\mathcal{C}\rangle_k \rightarrow \langle \nu_M, \mathsf{sat}\rangle$	$x_k \notin \mathsf{vars}(\mathcal{C})$
[Lift-Level]	$\langle M,\mathcal{C}\rangle_k \rightarrow \langle [\![M, x_k \mapsto \alpha]\!], \mathcal{C}\rangle_{k+1}$	$x_k \in \mathsf{vars}(\mathcal{C}) \wedge \alpha \in \mathsf{fsbl}(M) \wedge \mathsf{val}(\mathcal{C}_k, M) = \mathsf{true}$

Clause Satisfaction Rules

[B-Prop]	$\langle M,\mathcal{C}\rangle_k \vDash C \rightarrow \langle [\![M, C \rightarrow f]\!], \mathcal{C}\rangle_k$	$C = \{f_1, \dots, f_m, f\} \wedge \mathsf{val}(f,M) = \mathsf{undef} \wedge$ $\mathsf{comp}(f,M) \wedge \forall i : \mathsf{val}(f_i, M) = \mathsf{false}$
[Decide-Lit]	$\langle M,\mathcal{C}\rangle_k \vDash C \rightarrow \langle [\![M, f_1]\!], \mathcal{C}\rangle_k$	$\{f_1, f_2, \dots\} \subseteq C \wedge \mathsf{comp}(f_1, M) \wedge$ $\forall f_i : \mathsf{val}(f_i, M) = \mathsf{undef}$
[T-Prop]	$\langle M,\mathcal{C}\rangle_k \vDash C \rightarrow \langle [\![M, E \rightarrow f]\!], \mathcal{C}\rangle_k$	$f \in \{L, \neg L \mid L \in C\} \wedge \neg\mathsf{comp}(\neg f, M) \wedge$ $\mathsf{val}(f,M) = \mathsf{undef} \wedge E = \mathsf{exp}(f,M)$

Conflict Resolution Rules

[Resolve-Prop]	$\langle [\![M, E \rightarrow f]\!], \mathcal{C}\rangle_k \vdash C$	$\rightarrow \langle M,\mathcal{C}\rangle_k \vdash R$	$\neg f \in C \wedge$ $R = \mathsf{resolve}(C, E, f)$
[Resolve-Dec]	$\langle [\![M, f]\!], \mathcal{C}\rangle_k \vdash C$	$\rightarrow \langle M, \mathcal{C} \cup \{C\}\rangle_k \vDash C$	$\neg f \in C$
[Consume-Prop]	$\langle [\![M, E \rightarrow f]\!], \mathcal{C}\rangle_k \vdash C$	$\rightarrow \langle M,\mathcal{C}\rangle_k \vdash C$	$\neg f \in C$
[Consume-Dec]	$\langle [\![M, f]\!], \mathcal{C}\rangle_k \vdash C$	$\rightarrow \langle M,\mathcal{C}\rangle_k \vdash C$	$\neg f \in C$
[Drop-Level-1]	$\langle [\![M, x_{k+1} \mapsto \alpha]\!], \mathcal{C}\rangle_{k+1} \vdash C$	$\rightarrow \langle M,\mathcal{C}\rangle_k \vdash C$	$\mathsf{val}(C,M) = \mathsf{false}$
[Drop-Level-2]	$\langle [\![M, x_{k+1} \mapsto \alpha]\!], \mathcal{C}\rangle_{k+1} \vdash C$	$\rightarrow \langle M, \mathcal{C} \cup \{C\}\rangle_k \vDash C$	$\mathsf{val}(C,M) = \mathsf{undef}$
[Unsat]	$\langle [\![\,]\!], \mathcal{C}\rangle_1 \vdash C$	$\rightarrow \mathsf{unsat}$	

Figure 1: Transition Rules of MCSat

The T-Prop rule is the core component of any MCSat procedure. It utilizes theory knowledge to propagate literals during the search. The explanation function exp generates a valid lemma E that justifies the propagation. This rule was dubbed "R-Propagation" in [25] since the focus was merely real arithmetic. However, the rule itself does not rely on reals, only the explanation function does. For our purpose, we refer to this rule as "Theory-Propagation", in short T-Prop. In Section 4 we prove that explain functions for polynomials over finite fields always exists. Moreover, in Section 5 we show that explanation functions are also computable using zero decomposition procedures, avoiding the applications of field polynomials within MCSat.

The conflict resolution rules of Figure 1 rely on standard boolean conflict analysis [34], using the standard boolean resolution function resolve. We either resolve propagation or decision steps (Resolve-Prop, Resolve-Dec) or backtrack if there is no conflicting literal in the trail's top literal (Consume-Prop, Consume-Dec). The only theory-specific aspects of the conflict resolution are the rules Drop-Level-1 and Drop-Level-2, where we undo theory variable assignments. We add the conflict clause C to the clause set to avoid assignment repetition.

4 Theory Propagation for Polynomials over Finite Fields

The key challenge in designing an MCSat-based decision procedure for a particular theory is developing theory propagation in the respective theory to be used within the T-Prop rule of the MCSat calculus of Figure 1. In this section, we introduce zero decomposition procedures over polynomial constraints (Section 4.1) in support of theory propagation over finite fields, allowing us to prove the existence of explanation clauses within MCSat over finite fields (Section 4.2).

Upon application of the T-Prop rule, a literal is selected for propagation and justified by a newly generated explanation clause E. In our work we focus on propagating polynomial constraint literals f and define their respective explanations using so-called theory lemmas.

Definition 1 (Polynomial Explanation)**.** *Let f be a constraint, M a trail, and E a clause of constraints. E is a* valid (theory) lemma *if for any arbitrary assignment ν, $\nu(E) \neq \mathsf{false}$. The clause E* justifies f in M *iff $f \in E$ and $\forall f' \in E : f \neq f' \Rightarrow \mathsf{val}(f', M) = \mathsf{false}$. E is an* explanation clause *for f in M if E is a valid theory lemma and justifies f in M.*

Note that the explanation clauses E for f are generated using an explanation function exp during the applications of the T-Prop rule of Figure 1. We define the exp function as follows.

Definition 2 (Polynomial Explanation Function exp)**.** *A function $\mathsf{exp} : \{constraint\} \times \{trail\} \to \{clause\}$ is an* explanation function $\mathsf{exp}(f, M) = E$ *iff $f \notin M$, $\neg\mathsf{comp}(\neg f, M)$, and E is an explanation clause for f in M.*

Example 3. *A most trivial explanation function propagates f by excluding the current trail M of level k via $E = \{f\} \cup \{\neg f' \mid f' \in M \text{ and } x_{k-1} \in \mathsf{vars}(f')\} \cup \{x \neq \alpha \mid (x \mapsto \alpha) \in M\}$.*

As any (non-trivial) explanation function may introduce new literals, the termination of a general MCSat procedure requires that all newly introduced literals are taken from a finite basis.

Definition 3 (Finite-basis Polynomial Explanation)**.** *The function $\mathsf{exp}(f, M)$ is a* finite basis explanation function *if it returns an explanation clause E for f in M and all new literals in E are taken from a finite basis.*

In conclusion, if a theory admits a finite basis explanation function, then MCSat-based reasoning in that respective theory is terminating. Yet, providing a finite basis explanation function is not trivial. A key piece is an efficient procedure to decompose polynomial sets. We present our tailored procedures in Section 5.

4.1 Zeros in Polynomials over Finite Fields

Let us arbitrarily fix the sets of polynomials $P, Q \subset \mathbb{F}_q[X_k]$. We assume that P contains polynomials from equality constraints, and Q consists of inequality constraints.

We define the following sets of solutions: $\mathsf{zero}(P) = \{\boldsymbol{\alpha} \in \overline{\mathbb{F}}_q^k \mid p(\boldsymbol{\alpha}) = 0 \text{ for all } p \in P\}$ and $\mathsf{zero}_q(P) = \{\boldsymbol{\alpha} \in \mathbb{F}_q^k \mid p(\boldsymbol{\alpha}) = 0 \text{ for all } p \in P\}$. Clearly, $\mathsf{zero}_q(P) \subseteq \mathsf{zero}(P)$. For simplicity, we use set subtraction to define $\mathsf{zero}(P/Q) = \mathsf{zero}(P) \backslash \mathsf{zero}(Q)$ and $\mathsf{zero}_q(P/Q) = \mathsf{zero}_q(P) \backslash \mathsf{zero}_q(Q)$. We further write $\hat{P}$ for $P \setminus \mathbb{F}_q[X_{k-1}]$. We use the tuple $\mathcal{S} = (P, Q)$ to mean a *(polynomial) system* and write $\mathsf{zero}(\mathcal{S}) = \mathsf{zero}(P/Q)$. We finally define the projection set $\mathsf{proj}_k\mathsf{zero}_q(P/Q) = \{\boldsymbol{\alpha} \in \mathbb{F}_q^{k-1} \mid \exists \beta \in \mathbb{F}_q \text{ such that } (\boldsymbol{\alpha}, \beta) \in \mathsf{zero}_q(P/Q)\}$. Intuitively, projection sets are used to reduce the problem of solving polynomials over k variables into the smaller problem of solving polynomials over $k-1$ variables, providing thus means for eliminating the variable x_k.

Definition 4 (Zero Decomposition)**.** *A* zero decomposition procedure *is an algorithm that given $P, Q \subset \mathbb{F}_q[X_k]$ generates a set of systems $\Delta = \{(P_1, Q_1), \ldots, (P_m, Q_m)\}$ such that*

$$\mathsf{zero}_q(P/Q) = \bigcup_{(P',Q')\in\Delta} \mathsf{zero}_q(P'/Q'). \tag{1}$$

The zero decomposition procedure is projecting *in case $P', Q' \in \mathbb{F}_q[X_{k-1}]$ for all $(P', Q') \in \Delta$ and*

$$\mathsf{proj}_k \mathsf{zero}_q(P/Q) = \bigcup_{(P',Q')\in\Delta} \mathsf{zero}_q(P'/Q').$$

Given additionally $\boldsymbol{\alpha} \in \mathbb{F}_q^{k-1}$ which cannot be extended to a zero, i.e. there is no $\beta \in \mathbb{F}_q$ such that $(\boldsymbol{\alpha}, \beta) \in \mathsf{zero}_q(P/Q)$, we say that an algorithm is a weak projecting zero decomposition procedure for $\boldsymbol{\alpha}$ *if*

$$\mathsf{proj}_k \mathsf{zero}_q(P/Q) \subseteq \bigcup_{(P',Q')\in\Delta} \mathsf{zero}_q(P'/Q') \tag{2}$$

with $P', Q' \in \mathbb{F}_q[X_{k-1}]$ and $\boldsymbol{\alpha} \notin \mathsf{zero}_q(P'/Q')$ for all $(P', Q') \in \Delta$.

For many zero decomposition procedures [42], the *pseudo division* operation plays an important role, as follows. Consider polynomials $f, g \in \mathbb{F}_q[X_k]$, with $f \neq 0$. Let $r, o \in \mathbb{F}_q[X_k]$ denote polynomials. We define the *pseudo-remainder formula* (in x_k) as

$$l^d \cdot g = o \cdot f + r$$

where $l = \mathsf{lc}(f, x_k)$, $d = \max(\mathsf{deg}(g, x_k) - \mathsf{deg}(f, x_k) + 1, 0)$, and $\mathsf{deg}(r, x_k) < \mathsf{deg}(f, x_k)$. The *pseudo-remainder* r and *pseudo-quotient* o of g with respect to f in x_k are denoted as $\mathsf{prem}(g, f, x_k)$ and $\mathsf{pquo}(g, f, x_k)$, respectively. The polynomials o and r are uniquely determined by f and g and are computable [42].

Example 4. *Let $f = x_2 + x_1$ and $g = 3x_2x_1^2 + x_1$ in $\mathbb{F}_5[x_1, x_2]$. Noting that $-2 = 3$ in $\mathbb{F}_5$, we have eliminated x_2 in both the pseudo remainder and pseudo quotient by*

$$\underbrace{(3x_2x_1^2 + x_1)}_{g} = \underbrace{(-2x_1^2)}_{\mathsf{pquo}(g,f,x_2)} \cdot \underbrace{(x_2 + x_1)}_{f} + \underbrace{(2x_1^3 + x_1)}_{\mathsf{prem}(g,f,x_2)}.$$

Calculating gcds is another method for reducing the degree of x_k and thereby eliminating it. We employ subresultant regular subchains (SRSs) to calculate gcds with respect to a partial assignment, as shown in Lemma 2.4.2 of [42]. Given two polynomials $f, g \in \mathbb{F}_q[X_k]$ with $\mathsf{deg}(f, x_k) \geq \mathsf{deg}(g, x_k) > 0$, we denote by $\mathsf{srs}(f, g, x_k) = h_2, \ldots, h_r$ the SRS of f and g with regard to x_k. Let $l = \mathsf{lc}(g, x_k)$ and $l_\ell = \mathsf{lc}(h_\ell, x_k)$. Then for $2 \leq \ell \leq r$, we have

$$\gcd(f(\boldsymbol{\alpha}, x_k), g(\boldsymbol{\alpha}, x_k)) = h_\ell(\boldsymbol{\alpha}, x_k)$$

if $\boldsymbol{\alpha} \in \mathsf{zero}(\{l_{\ell+1}, \ldots, l_r\}/\{l, l_\ell\})$. When f, g, and h_ℓ are partially evaluated w.r.t. $\boldsymbol{\alpha}$, the above gcd is equivalent to computing a univariate gcd.

Example 5. *Let $f = x_3^2 + x_3x_2 + 4$ and $g = x_3x_2 + x_1$ in $\mathbb{F}_5[x_1, x_2, x_3]$. Then $\mathsf{srs}(f, g, x_3) = [h_2, h_3] = [x_3x_2 + x_1, -x_2^2x_1 - x_2^2 + x_1^2]$. Using the assignment function $\nu = \{x_2 \mapsto 1, x_1 \mapsto 3\}$ we have $\nu(f) = x_3^2 + x_3 - 1$ and $\nu(g) = \nu(h_2) = x_3 - 2$ which is indeed the gcd of $\nu(f)$ and $\nu(g)$.*

4.2 Explaining Propagated Literals

We now show that polynomial constraints over finite fields have finite basis explanation functions.

Our MCSat-based theory propagation works as follows. Let M be a level k trail, implying that variable x_k is not yet assigned. Suppose f is a constraint such that $f \notin \mathsf{constr}(M)$ and $\neg\mathsf{comp}(\neg f, M)$. We derive polynomial constraints that are false for $[\![M, \neg f]\!]$ to generate an explanation clause E in order to justify propagating f. To enable an instant application of T-Prop, we ensure that for all $e \in E$ either $e = f$, $\mathsf{level}(e) < k$, or $\neg e \in \mathsf{constr}(M)$ must hold.

Finite Basis Explanations. Towards generating explanation clauses E, we consider the polynomial constraint systems

$$A = \{f' \in \mathsf{constr}(M) \mid \mathsf{level}(f') = k\} \cup \{\neg f\} \text{ and } A_{\rhd} = \{p \mid (p \rhd 0) \in A\} \text{ for } \rhd \in \{=, \neq\}. \quad (3)$$

Further, we fix the system $\mathcal{A} = (A_=, A_{\neq})$. From $\neg\mathsf{comp}(\neg f, M)$ follows that $(\boldsymbol{\alpha}_M, \beta) \notin \mathsf{zero}_q(\mathcal{A})$ for all $\beta \in \mathbb{F}_q$. Based on Definition 4, we use a weak projecting zero decomposition procedure for $\boldsymbol{\alpha}_M$ and decompose $\mathcal{A}$ into multiple systems $\mathcal{A}_1, \ldots, \mathcal{A}_r$ such that for every $1 \leq \ell \leq r$ we have that $\boldsymbol{\alpha}_M \notin \mathsf{zero}(\mathcal{A}_\ell)$. Then each $\mathcal{A}_\ell = (P_\ell, Q_\ell)$ contains (at least) one polynomial u in $\mathbb{F}_q[X_{k-1}]$ that excludes $\boldsymbol{\alpha}_M$. Depending on whether $u \in P_\ell$ or $u \in Q_\ell$, we generate an appropriate constraint c_ℓ as $u = 0$ or $u \neq 0$, respectively, to ensure that $c_\ell(\boldsymbol{\alpha}_M) = \mathsf{false}$. As a result, we set $C = \{c_1, \ldots, c_r\}$.

For any $(\boldsymbol{\alpha}, \beta) \in \mathsf{zero}_q(\mathcal{A})$, by Definition 4 we have that $\boldsymbol{\alpha} \in \mathsf{zero}_q(\mathcal{A}_\ell)$ for some $1 \leq \ell \leq r$ and thus $c_\ell(\boldsymbol{\alpha}) = \mathsf{true}$. Hence, anytime an assignment function fulfills all constraints from A, it also fulfills at least one constraint of C. We generate the explanation clause $E = \{\neg a \mid a \in A\} \cup C$.

Theorem 1 (Explanation Clause E)**.** *Given a trail M of level k. Let f be a constraint such that $f \notin \mathsf{constr}(M)$ and $\neg\mathsf{comp}(\neg f, M)$. Further let $A = \{f' \in \mathsf{constr}(M) \mid \mathsf{level}(f') = k\} \cup \{\neg f\}$ and $C = \{c_1, \ldots, c_r\}$ constructed as defined above. Then $E = \{\neg a \mid a \in A\} \cup C$ is an explanation clause for f in M.*

Proof. By Definition 1 we show that E is a valid theory lemma and justifies f in M.

By construction we have that $\neg f \in A$ and thus $f \in E$. Let $a \in A$ be a constraint such that $a \neq \neg f$. Then $\neg a \in \mathsf{constr}(M)$, therefore, $\mathsf{val}(a, M) = \mathsf{false}$. Let $c \in C$, from the construction of C it immediately follows that $\mathsf{level}(c) < k$ and $c(\boldsymbol{\alpha}_M) = \mathsf{false}$, thus $\nu_M(c) = \mathsf{false}$. Since all constraints in E but f evaluate to false under M, we derive that E *justifies* f *in* M.

Let ν be an arbitrary assignment. We distinguish the following two cases:
Case 1: Assume $\nu(\neg a) = \mathsf{false}$ for all $a \in A$. Let $\mathcal{A} = (A_=, A_{\neq})$ as defined in (3) and $\boldsymbol{\alpha}$ be ν represented as a k-tuple. Since $\nu(a) = \mathsf{true}$ for all $a \in A$, we have $\boldsymbol{\alpha} \in \mathsf{zero}(\mathcal{A})$. Since $\mathcal{A}$ was zero decomposed into systems $\mathcal{A}_1, \ldots, \mathcal{A}_r$, there exists $1 \leq i \leq r$ such that $\boldsymbol{\alpha} \in \mathsf{zero}(\mathcal{A}_i)$. Thus $c_i(\alpha) = \mathsf{true}$ for $c_i \in C$. As $C \subseteq E$, it follows that $\nu(E) = \mathsf{true}$.
Case 2: Assume $\nu(\neg a) \neq \mathsf{false}$ for some $a \in A$. As $\neg a \in E$, we obtain $\nu(E) \neq \mathsf{false}$.

As $\nu(E) \neq \mathsf{false}$ in both of the cases above, we conclude that E is a valid lemma. □

Example 6. *(Example 1) We have $\mathbb{F}_5[x_1, x_2]$ and two unit clauses $C_1 = \{c_1\} = \{x_1^2 - 1 = 0\}$ and $C_2 = \{c_2\} = \{x_1x_2 - x_2 - 1 = 0\}$. Assume the current trail is $M = [\![x_1^2 - 1 = 0,\ x_1 \mapsto 1]\!]$. We cannot add c_2 as we have $\neg\mathsf{comp}(c_2, M)$. Towards a conflict, we propagate $\neg c_2$. Then $A = \{x_1x_2 - x_2 - 1 = 0\}$, $A_= = \{x_1x_2 - x_2 - 1\}$, and $A_{\neq} = \emptyset$. Using a weak zero decomposition procedure (cf. Example 7), we derive the zero decomposition $\Delta = \{(\emptyset, \{x_1 - 1\})\}$ and generate $E = \{\neg c_2, x_1 - 1 \neq 0\}$ to justify $\neg c_2$ on M. However, $\neg c_2$ on M results in a conflict with C_2. Thus, we resolve E with C_2 and learn that $x_1 - 1 \neq 0$ must hold. We backtrack the assignment* *of x_1 and end up with $M = [\![x_1^2 - 1 = 0, x_1 - 1 \neq 0]\!]$. Assigning $x_1 \mapsto 4$, we eventually reach* Sat *with $M = [\![c_1, x_1 - 1 \neq 0, x_1 \mapsto 4, c_2, x_2 \mapsto 2]\!]$ and $\nu_M = \{x_1 \mapsto 4, x_2 \mapsto 2\}$.*

We next show that our explanations from Theorem 1 can be turned into explanations with finite basis. As such, using our explanation functions in the T-Prop rule ensures that MCSat terminates for our theory (Section 5).

Theorem 2 (Finite Based Explanations)**.** *Every explanation function for theory of $\mathbb{F}_q[X]$ can be finite based.*

Proof. The proof relies on application of Fermat's little theorem. We show that every polynomial p in the explanation clause can be translated to an equivalent polynomial p' from a finite basis. Given $\mathbb{F}_q$, by the generalized Fermat's little theorem, every element $a \in \mathbb{F}_q$ satisfies $a^q \equiv a$. Let $t = c \prod_{i=1}^{r} x_i^{p_i}$ be a term of p. Then by Fermat's little theorem an equivalent term t' can be found such that $p_i \leq q$ for all $1 \leq i \leq r$. When $c \in \mathbb{F}_q$, r is finite, and $d_i \leq q$ for $1 \leq i \leq r$, there is only a finite set T of different terms. As a polynomial is a sum of terms, there are $2^{|T|} - 1$ different polynomials that can be constructed from T. By replacing all terms of p by an equivalent term from T, we have an equivalent polynomial p' from a finite basis. □

5 Explanation Functions over Finite Fields in MCSat

Section 4 established the generation of explanation clauses for the theory of polynomials over finite fields (Theorem 1) and proved the existence of a finite basis explanation function (Theorem 2).

The primary component of the provided explanation generation procedure is a weak projecting zero decomposition procedure. It remains to show in this section that such a procedure exists. We provide a novel method for giving such a decomposition that does not rely on field polynomials, as in [20]. This is the last piece to providing a finite basis explanation function and thus to turning the MCSat calculus of Figure 1 into an SMT solving approach over finite fields.

Projecting Zero Decomposition. Recall the generation of finite basis explanations in Section 4.2 for a trail M of level k. Again, we have $\boldsymbol{\alpha}_M \in \mathbb{F}_q^{k-1}$ and $\mathcal{A} = (A_=, A_{\neq})$ be the polynomial system of constraint set A as defined in (3), such that $(\boldsymbol{\alpha}_M, \beta) \notin \mathsf{zero}(\mathcal{A})$ for all $\beta \in \mathbb{F}_q$. Depending on $|A|$, $|A_=|$, and $|A_{\neq}|$, we utilize different projecting procedures to find explanation clauses E. Each procedure takes $\mathcal{A}$ and $\boldsymbol{\alpha}_M$ as input and decomposes $\mathcal{A}$ in the set of systems $\Delta = \{\mathcal{A}_1, \ldots, \mathcal{A}_r\}$, according to Definition 4. By the construction of E, it thus suffices to return one constraint f_ℓ of each system $\mathcal{A}_\ell \in \Delta$ such that $f_\ell(\boldsymbol{\alpha}_M) = \mathsf{false}$.

Based on the structure of A, we use single polynomial projections (Section 5.1) or SRS-based projections (Section 5.2) to derive the explanation constraints f_ℓ of each system $\mathcal{A}_\ell$.

5.1 Single Polynomial Projection for Deriving Explanation Constraints

In case $|A| = 1$ the coefficients of the polynomial constraint $f \in A$ can be used for projecting. By the construction of A we have that $A = \{\neg f\}$, $\mathsf{level}(\neg f) = k$ and $\boldsymbol{\alpha}_M$ cannot be extended to satisfy $\neg f$. We write $\mathsf{poly}(\neg f) = c_1 \cdot x_k^{d_1} + \cdots + c_m \cdot x_k^{d_m}$. By the definition of this polynomial, we have that each $c_i \in \mathbb{F}_q[X_{k-1}]$ and thus c_i can be fully evaluated by $\boldsymbol{\alpha}_M$. Let $\gamma_i = c_i(\boldsymbol{\alpha}_M)$ for $1 \leq i \leq m$ and set $F = \{c_i - \gamma_i \neq 0 \mid 1 \leq i \leq m\}$. Each $f_\ell \in F$ represents one (single-polynomial) system which is returned by a zero decomposition procedure. We denote this procedure as $\mathsf{Proj}_{\mathsf{Coeff}}$ and prove the following.

Theorem 3 (Single Polynomial Weak Projection)**.** *Let $\mathcal{A}$ be a polynomial system with a single polynomial $a \in \mathbb{F}_q[X_k]$ and let $\boldsymbol{\alpha} \in \mathbb{F}_q[X_{k-1}]$ be an assignment that cannot be extended to be a zero of $\mathcal{A}$. Then $\mathsf{Proj}_{\mathsf{Coeff}}(\mathcal{A}, \boldsymbol{\alpha})$ is a weak projecting zero decomposition procedure for $\boldsymbol{\alpha}$.*

Proof. Termination of $\mathsf{Proj}_{\mathsf{Coeff}}$ is obvious. Let $a = c_1 \cdot x_k^{d_1} + \cdots + c_m \cdot x_k^{d_m}$. Furthermore, there is no $\beta \in \mathbb{F}_q$ such that $(\boldsymbol{\alpha}, \beta) \in \mathsf{zero}_q(\mathcal{A})$. Then $\mathsf{Proj}_{\mathsf{Coeff}}(\mathcal{A}, \boldsymbol{\alpha})$ returns a set of systems $\Delta = \{(\emptyset, \{c_i - \gamma_i\}) \mid 1 \leq i \leq m\}$ where $\gamma_i \in \mathbb{F}_q$ is $c_i(\boldsymbol{\alpha})$. Let $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_k) \in \mathsf{zero}_q(\mathcal{A})$. Towards a contradiction, assume that $\boldsymbol{\xi} \notin \mathsf{zero}_q(\mathcal{S})$ for all $\mathcal{S} \in \Delta$. Then for all c_i, we have that $c_i(\boldsymbol{\xi}) = \gamma_i = c_i(\boldsymbol{\alpha})$, i.e. all coefficients of a evaluate to the same value for $\boldsymbol{\alpha}$ and $\boldsymbol{\xi}$. As there is no value to be assigned to x_k such that $\boldsymbol{\alpha}$ can be extended to a zero of $\mathcal{A}$, $\boldsymbol{\xi}$ cannot exist, as ξ_k would extend $\boldsymbol{\alpha}$. Therefore, $\boldsymbol{\xi} \in \mathsf{zero}_q(\mathcal{S})$ for some $\mathcal{S} \in \Delta$. Furthermore, note that $\boldsymbol{\alpha}$ is excluded from all systems in Δ by construction. □

Example 7. *Filling the gap in Example 6, we use $\mathsf{Proj}_{\mathsf{Coeff}}$ to decompose $\mathcal{A} = (\{x_1x_2 - x_2 - 1\}, \emptyset)$ with $\boldsymbol{\alpha} = (1)$. Let a be the polynomial in $\mathcal{A}$. We write $a = (x_1 - 1)x_2^1 + (-1)x_2^0$. Evaluating the coefficients, we get $\gamma_1 = 0$, $\gamma_0 = -1$ and generate $F = \{(x_1 - 1) - 0 \neq 0, (-1) - (-1) \neq 0\}$. As there are no zeros in the second system, we return $\Delta = \{(\emptyset, \{x_1 - 1\})\}$.*

5.2 SRS-Based Projection for Deriving Explanation Constraints

For $|A| > 1$, we use the procedure $\mathsf{P}_{\mathsf{Reg}}(\mathcal{A}, \boldsymbol{\alpha})$ as shown in Algorithm 1 and described next. Algorithm 1 is a weak projecting zero decomposition procedure for $\boldsymbol{\alpha}$ that decomposes the system $\mathcal{A}$. It utilizes SRS chains to calculate gcds that reduce the degree of x_k. This idea is based on the algorithm RegSer presented in [41, 42]. While this original work presents RegSer for polynomials over fields with characteristic 0 only, the work of [26] claims validity of the approach also over finite fields. Algorithm 1 relies on this result and proceeds as follows.

Consider two polynomials $p_1, p_2 \in \mathbb{F}_q[X_k]$ with $\mathsf{lv}(p_1) = \mathsf{lv}(p_2) = x_k$ and let $h_2, \ldots, h_r = \mathsf{srs}(p_1, p_2, x_k)$. Further, let $l_i = \mathsf{lc}(h_i, x_k)$ for all $2 \leq i \leq r$. Then, by case distinction over the evaluation of $l_2, \ldots, l_r \in \mathbb{F}_q[X_{k-1}]$, the set of zeros can be decomposed to guarantee that each h_i is a gcd of p_1 and p_2 in one newly generated system. The gcd property is then used to reduce $\mathsf{deg}(p_1, x_k)$ and $\mathsf{deg}(p_2, x_k)$ in this system. The original approach of [41, 42] splits the zero set for every h_i and thus generates exponentially many systems. In our setting, a full decomposition can be avoided by guiding the search using $\boldsymbol{\alpha}$. This is done by evaluating $l_2, \ldots, l_r$ with $\boldsymbol{\alpha}$ and not further exploring systems that already exclude $\boldsymbol{\alpha}$. Therefore, only a linear amount of systems is generated in Algorithm 1. This computation is performed until a polynomial is found that excludes $\boldsymbol{\alpha}$. In case there are polynomials in x_k left, we exclude them using $\mathsf{Proj}_{\mathsf{Coeff}}$.

The **return** *call* **if** c statements of Algorithm 1, where *call* is a recursive call and the *guard* c is a polynomial constraint, are used to track which path the search takes. If c evaluates to true under $\boldsymbol{\alpha}$ then the recursive call is performed and its result is returned. In addition, the procedure keeps a set of tracked constraints C that is empty in beginning. Whenever a guard c is reached but false under $\boldsymbol{\alpha}$, it is added to C, otherwise, $\neg c$ is added. The constraints in C are added to the returned sets accordingly. Thus, the constraints in C describe the search space that was not visited during the search.

Recall the notion of $\hat{P}, \hat{Q}$ from Section 4.1. Lines 1–2 of Algorithm 1 return an excluding polynomial in case one is found. Line 5 ensures that $\mathsf{lc}(p, x_2) \neq 0$ which is a requirement for any further gcd operation. Lines 6-12 are used to remove polynomials of P until only one is left, which is then used in lines 14-19 to remove polynomials from Q. Lines 10 and 17 handle the special case $\mathsf{lv}(h_r) < x_k$. By definition of SRS, $\mathsf{lv}(h_i) = x_k$ for $2 \leq i \leq r - 1$, but not necessarily for h_r. Roughly speaking, $\mathsf{lv}(h_r) < x_k$ denotes a constant gcd and, thus, divisor-free polynomials. Line 22 splits elements in Q to remove x_k in case all polynomials $p \in P$ are free of x_k.

Algorithm 1: $\mathsf{P}_{\mathsf{Reg}}(\mathcal{A} = (P, Q),\ \boldsymbol{\alpha})$

```
1  return $(\{p\}, \emptyset)$ for any $p \in P$ with $\mathsf{lv}(p) < x_k$ and $p(\boldsymbol{\alpha}) \neq 0$
2  return $(\emptyset, \{q\})$ for any $q \in Q$ with $\mathsf{lv}(q) < x_k$ and $q(\boldsymbol{\alpha}) = 0$
3  if $|\hat{P}| > 0$ then
4      select $p \in P$ with the smallest positive $\mathsf{deg}(p, x_k)$
5      return $\mathsf{P}_{\mathsf{Reg}}(P \setminus \{p\} \cup \{\mathsf{red}(p, x_k)\}, Q, \boldsymbol{\alpha})$ if $\mathsf{lc}(p, x_k)(\boldsymbol{\alpha}) = 0$
6      if $|\hat{P}| > 1$ then
7          select any $p' \in \hat{P} \setminus \{p\}$
8          compute $h_2, \dots, h_r = \mathsf{srs}(p', p, x_k)$ and let $l_i = \mathsf{lc}(h_i, x_k)$ for $2 \leq i \leq r$
9          if $\mathsf{lv}(h_r) < x_k$ then
10             return $\mathsf{P}_{\mathsf{Reg}}(P \setminus \{p, p'\} \cup \{h_r, h_{r-1}\}, Q, \boldsymbol{\alpha})$ if $l_{r-1}(\boldsymbol{\alpha}) \neq 0$
11             $r \leftarrow r - 2$
12         for $i = r, \dots, 2$ do return $\mathsf{P}_{\mathsf{Reg}}(P \setminus \{p, p'\} \cup \{h_i, l_{i+1}, \dots, l_r\}, Q, \boldsymbol{\alpha})$ if $l_i(\boldsymbol{\alpha}) \neq 0$
13     else if $|\hat{Q}| > 0$ and $\mathsf{lv}(p) = x_k$ then
14         select any $q \in \hat{Q}$
15         compute $h_2, \dots, h_r = \mathsf{srs}(q, p, x_k)$ and let $l_i = \mathsf{lc}(h_i, x_k)$ for $2 \leq i \leq r$
16         if $\mathsf{lv}(h_r) < x_k$ then
17             return $\mathsf{P}_{\mathsf{Reg}}(P \setminus \{p\} \cup \{\mathsf{pquo}(p, h_r, x_k)\}, Q \setminus \{q\}, \boldsymbol{\alpha})$ if $l_r(\boldsymbol{\alpha}) \neq 0$
18             $r \leftarrow r - 1$
19         for $i = r, \dots, 2$ do return $\mathsf{P}_{\mathsf{Reg}}(P \setminus \{p\} \cup \{\mathsf{pquo}(p, h_i, x_k), l_{i+1}, \dots, l_r\}, Q, \boldsymbol{\alpha})$ if $l_i(\boldsymbol{\alpha}) \neq 0$
20     return $\mathsf{P}_{\mathsf{Reg}}((P, Q) \cup \mathsf{Proj}_{\mathsf{Coeff}}(p, \boldsymbol{\alpha}), \boldsymbol{\alpha})$
21 else if $|\hat{Q}| > 0$ then
22     forall $q \in \hat{Q}$ do return $\mathsf{P}_{\mathsf{Reg}}(P, Q \setminus \{q\} \cup \{\mathsf{red}(q, x_k)\}, \boldsymbol{\alpha})$ if $\mathsf{lc}(q, x_k)(\boldsymbol{\alpha}) = 0$
23     return $\mathsf{P}_{\mathsf{Reg}}((P, Q \setminus \hat{Q}) \cup \mathsf{Proj}_{\mathsf{Coeff}}(\prod_{q' \in \hat{Q}} q', \boldsymbol{\alpha}), \boldsymbol{\alpha})$
```

Example 8. *Assume we have the system* $\mathcal{A} = (\{x_3^2 + x_3x_2 + 4\}, \{x_3x_2 + x_1\})$ *in* $\mathbb{F}_5[x_1, x_2, x_3]$ *and let* $\boldsymbol{\alpha} = (3, 1)$*. At line 15,* $\mathsf{P}_{\mathsf{Reg}}$ *we will calculate the first SRS according to Example 5. Eventually, the computation terminates with a zero decomposition represented by the constraints* $\{x_2 = 0, -x_2^2x_1 - x_2^2 + x_1^2 \neq 0, -x_2^4 + 2x_2^2x_1 \neq 0\}$*, each representing one generated system.*

Theorem 4 (SRS-Based Weak Projection)**.** *Let* $\mathcal{A}$ *be a polynomial system and let* $\boldsymbol{\alpha} \in \mathbb{F}_q^{k-1}$ *be an assignment that cannot be extended to be a zero of* $\mathcal{A}$*. Then* $\mathsf{P}_{\mathsf{Reg}}(\mathcal{A}, \boldsymbol{\alpha})$ *of Algorithm 1 is a weak projecting zero decomposition of* $\mathcal{A}$ *for* $\boldsymbol{\alpha}$*.*

Proof. We show that $\mathsf{P}_{\mathsf{Reg}}(\mathcal{A}, \boldsymbol{\alpha})$ terminates and is a weak projecting zero decomposition for $\boldsymbol{\alpha}$.

Termination: As the first two loops in Algorithm 1 are bound by the size of the SRS decomposition r and the size of a SRS is bound, both loops certainly terminate. The third and last loop iterates over the finite amount of elements in $\hat{Q}$ and thus terminates. It remains to show that the recursion depth of Algorithm 1 is bound. Note that for every recursive call of $\mathsf{P}_{\mathsf{Reg}}$ the degree in x_k for at least one polynomial in $\mathcal{A} = (P, Q)$ decreases. Once $\hat{P} = \hat{Q} = \emptyset$ no further recursive call is performed. We distinct two cases:

Case 1: Assume $|\hat{P}| > 0$. Then, the degree of x_k in polynomials of P is reduced in each recursive call of lines 4-19 of Algorithm 1. In case one polynomial $p \in \hat{P}$ remains, we use $\mathsf{Proj}_{\mathsf{Coeff}}$ to remove x_k.

Case 2: Assume $|\hat{P}| = 0$. Algorithm 1 proceeds by splitting polynomials in $\hat{Q}$ in line 22. For a given polynomial $q \in \hat{Q}$ the recursion depth of the call in line 22 is bound by the number of coefficients of q in x_k. For each call the leading coefficient is removed. Once x_k is removed, we have $\mathsf{lc}(q, x_k) = q$. Then, we either return in line 2 or the guard of the call in line 22 is false. As all recursive calls eventually terminate, Algorithm 1 terminates. Note that from the zero decomposition argument below follows that the recursion always ends in line 1 or 2. This usually happens before all x_k are eliminated.

Zero Decomposition: Results of [41, 42] imply that RegSer is a zero decomposition procedure that generates a sequence of regular systems. Besides other properties, for a regular system (P, Q) holds that either $\hat{P} = \emptyset$ or $\hat{Q} = \emptyset$. Furthermore, P is a triangular set, thus $|\hat{P}| \leq 1$. With a very similar argument can be proven that $\mathsf{P}_{\mathsf{Reg}}$ performs a zero decomposition towards regular systems, although the systems are not fully computed. In $\mathsf{P}_{\mathsf{Reg}}$ the decomposition ends after x_k has been fully processed as for generating explanations further decomposition is not required.

Let $\mathcal{A}' = (P, Q)$ be one decomposed system from $\mathcal{A}$. We first show that $P, Q \in \mathbb{F}_q[X_{k-1}]$ and $\boldsymbol{\alpha} \notin \mathsf{zero}_q(\mathcal{A}')$. From the lemmas presented in [42] for RegSer, it follows that the each decomposition step in lines 4-19 of Algorithm 1 as well as line 22 performs a zero decomposition according to equation (1); thus, $\boldsymbol{\alpha}$ cannot be extended to a zero of any such generated system. In case $\mathcal{A}'$ is a systems that was not further expanded in a conditional recursive call, i.e. the negation of the guard is in $\mathcal{A}'$, then the desired property holds by construction. In case $\mathcal{A}'$ contains a polynomial from $\mathbb{F}_q[X_{k-1}]$ which excludes $\boldsymbol{\alpha}$ directly, Algorithm 1 stops in lines 1 or 2, returning only this one polynomial. In case the regular decomposition procedure has concluded for x_k and no such polynomials can be found, by definition of a regular system, we end up with either exactly one polynomial $p \in \hat{P}$ or $\hat{Q} \neq \emptyset$, but not both. We distinct two cases:

(a) Assume $\hat{Q} = \emptyset$, then $\hat{P} = \{p\}$. Since $\boldsymbol{\alpha}$ cannot be extended to a zero of $\mathcal{A}'$ but is not excluded by any other polynomial in $\mathcal{A}'$, we conclude that it cannot be extended to become a zero of p. Therefore, we may call $\mathsf{Proj}_{\mathsf{Coeff}}$ in line 20 to further decompose $\mathcal{A}'$. The weak projecting zero decomposition property of $\mathsf{Proj}_{\mathsf{Coeff}}$ concludes the proof.

(b) Assume $\hat{Q} \neq \emptyset$, then $\hat{P} = \emptyset$. Since the regular decomposition process has concluded, the recursive call in line 22 is not executed for any $q \in \hat{Q}$. We know that $\boldsymbol{\alpha}$ cannot be extended such that all $q \in \hat{Q}$ evaluate to a non-zero value, we have that the product of all $q \in \hat{Q}$ when evaluated with $(\boldsymbol{\alpha}, \beta)$ for all $\beta \in \mathbb{F}_q$. We thus use $\mathsf{Proj}_{\mathsf{Coeff}}$ to concludee the weak projecting zero decomposition property.

We finally show that $\mathsf{P}_{\mathsf{Reg}}$ fulfills equation 2. Let $\boldsymbol{\xi} \in \mathsf{zero}_q(\mathcal{A})$. As $\mathsf{P}_{\mathsf{Reg}}$ performs a zero decomposition, there is a system $\mathcal{A}'$ such that $\boldsymbol{\xi} \in \mathsf{zero}_q(\mathcal{A}')$. If Algorithm 1 returns in lines 1 or 2, then $\boldsymbol{\xi}$ is a zero of the returned single polynomial (sub-)system of $\mathcal{A}'$. If $\mathcal{A}'$ is not further expanded because of a guard c in an conditional recursive call, the polynomial of $\neg c$ is in the according set of A' and returned as a single polynomial sub-system of $\mathcal{A}'$. As $\boldsymbol{\xi}$ is a zero of $\mathcal{A}'$, it is also a zero of the returned sub-system. Finally, in case $\mathsf{P}_{\mathsf{Reg}}$ utilizes $\mathsf{Proj}_{\mathsf{Coeff}}$ to remove a polynomial in x_k from $\mathcal{A}'$, we have by Theorem 3 that $\boldsymbol{\xi}$ is a zero of one of the returned (projected) systems. In any case, $\boldsymbol{\xi}$ is a zero of the decomposition and thus equation 2 holds. □

6 Implementation

We have implemented our MCSat approach for SMT solving over finite fields in a new prototype[1], written in Python and using the computer algebra system Sage [36] for handling polynomials. While our work is not limited to a specific field order, practical implementation constraints (from our implementation as well as Sage) are a limiting factor in the prototype's ability to

[1]The source code of the prototype together with the generated test instances are available: https://github.com/Ovascos/ffsat

handle large(r) field. Besides the general procedure of MCSat and the theory specific details presented in Sections 4 and 5, the performance of our approach therefore depends on certain implementation details. In the sequel we discuss our design decisions.

Selecting literals for propagation. While the general MCSat framework does not restrict the application of theory propagation beyond the conditions of the T-Prop rule, it is up to the theory to determine whether a theory propagation is applicable and appropriate. For the theory of polynomials over finite fields, we utilize a similar propagation strategy as [25] uses for reals. Let $\langle M, \mathcal{C} \rangle_k$ be the current state with feasible trail M and $f \in C$ a literal of a previously selected (yet unsatisfied) clause $C \in \mathcal{C}$ such that $\mathsf{level}(f) = k$ and $\mathsf{val}(f, M) = \mathsf{undef}$. If this happens, we use T-Prop to add f to M in order to satisfy C.

Let $\mathcal{X}_k \subseteq \mathbb{F}_q$ be the set of possible values for x_k that satisfy $\nu[M](f)$. We can distinguish four different scenarios of propagation by comparing feasible values for x_k, namely:

(i) If $\mathcal{X}_k = \mathbb{F}_q$, then f is propagated.
(ii) If $\mathcal{X}_k = \emptyset$, then $\neg f$ is propagated.
(iii) If $\mathcal{X}_k \supseteq \mathsf{fsbl}(M)$, i.e. f does not restrict the feasible values of M, then f is propagated.
(iv) If $\mathcal{X}_k \cap \mathsf{fsbl}(M) = \emptyset$, then $\neg f$ is propagated.

Informally, a propagation of f demonstrates the (theory) knowledge that C is fulfilled by M. By propagating $\neg f$ with explanation E, it is very likely that a conflict will arise right away (cf. Example 6). The design of E results in an immediate resolution of E with C. Because generating explanation clauses is costly, they are not generated at the moment of propagation but only when they are needed for conflict analysis.

Storing feasible values. As we work with finite set $\mathbb{F}_q$ of theory values, feasible values for a reasonable small q can be enumerated. Even for larger field orders, the number of zeros of a polynomial given a partial assignment is still constrained by the length of the polynomial.

Variable Order. The order in which the theory variables are assigned in the trail can hugely influence the number of conflicts and thus generated explanations. Finding a beneficial variable order is a general consideration for both MCSat style approaches and computer algebra algorithms alike. While it is highly important for practical performance, our procedure is correct for any ordering; optimizing the variable order is an interesting task for future work.

7 Experiments and Discussion

We compare the performance of our Python prototype in solving polynomial systems to state-of-the-art Gröbner basis techniques provided by Sage. It is important to note that by design Gröbner basis techniques require polynomial systems (cf. Section 4.1) as inputs and are incapable of handling polynomial constraints (i.e. disjunctions and conjunctions of polynomials). We therefore limit our experimental comparison to polynomial benchmarks with a small subset of inputs, as Gröbner basis algorithms with general polynomial constraints over finite fields would involve exponential many calls (in the number of constraints) to the Gröbner basis algorithm.

We further note that SMT-LIB standard and repository [4] does not support finite field arithmetic. For this reason, we cannot yet directly compare our work to SMT-based solvers. To circumvent such limitations, we represent polynomial constraints directly in our Python framework and compare our work only to Gröbner basis approaches supporting such input.

Type	q	n	c	FFSat	GB	GB$_{\text{LEX}}$
Rand	3	8	8	**25**	**25**	**25**
Rand	3	16	16	**12**	11	0
Craft	3	32	32	**25**	**25**	0
Craft	3	64	64	**25**	24	0
Rand	13	8	4	**25**	0	0
Rand	13	8	8	**1**	0	0
Craft	13	32	16	**19**	18	1
Rand	211	8	4	**17**	0	0
Rand	211	8	16	0	0	0
Craft	211	16	8	24	**25**	**25**

Table 1: Instances solved by FFSat, GB, and GB$_{\text{LEX}}$, out of 25 polynomial systems per test set.

Experimental setup. For our experiments on SMT solving over finite fields, we created 250 polynomial systems over a range of finite field orders (3, 13, 211) and different numbers of variables (up to 64). To get better insights, we have utilized two different methods of polynomial system generation:

- Rand: All polynomials in this test set are fully random generated by Sage's `random_element` function. The degree of the polynomials is at most 4. These created systems are more frequently unsatisfiable and have fewer zeros on average. This category of tests has smaller systems and fewer variables since they are challenging to solve (for any strategy). It is ensured that at least one polynomial has a constant term to avoid trivial 0-solutions, as this would give our approach an unfair advantage.
- Craft: These polynomial systems are crafted to have multiple solutions by explicitly multiplying zeros. They tend to be easy to solve. Thus, these systems are considerable larger with a huge amount of variables. Polynomial constraints are restricted to up to 5 distinct variables with up to 3 zeros each.

Each test set consists of 25 polynomial systems with fixed field order and a fixed number of variables and constraints; see Table 1. Our experiments were run on an AMD EPYC 7502 CPU with a timeout of 300 seconds per benchmark instance.

We compare our procedure (FFSat) to a Gröbner basis approach (GB). The latter uses field polynomials to limit the solutions to those for the base field. To get an elimination ideal and thus ensure to get a satisfiable assignment from a calculated Gröbner basis, one typically relies on lexicographic term-ordering, which is especially expensive. However, to "only" check whether a polynomial system is satisfiable without returning an assignment, it suffices to calculate the Gröbner basis in any term ordering. In our experiments we therefore calculate two different bases. GB uses the (efficient) default ordering provided by Sage, while GB$_{\text{LEX}}$ uses a lexicographic term ordering.

Experimental results. For analyzing our experimental findings, let us note that the already highly engineered Gröbner basis algorithms written in C/C++ utilized by Sage have an inherent performance advantage compared to our Python implementation. Yet, Table 1 demonstrates that our approach works well for satisfiable cases. The number of instances that were resolved between FFSat and GB within the predetermined timeout of 300s for each instance is also compared in Table 1. Each test is identified by its type, the finite field size q, the number of variables n, and the number of constraints per system c.

Experimental analysis and discussions. We note the following key insights of our approach in comparison to Gröbner basis approaches:

- Our Python prototype can already keep up with highly engineered Gröbner basis approaches on some classes of instances but further engineering work is required to match existing Gröbner basis implementations consistently.
- The strength of our work comes with solving satisfiable instances. This is because we can often find a satisfying assignment without fully decomposing the polynomial system.
- While the MCSat approach is capable of detecting conflicts by deriving empty clauses, the point in time when an empty clause can be derived is highly dependent on the variable order.
- The lack of an order on finite fields leads to the generation of many inequality constraints when a partial assignment cannot be extended. Developing further optimizations to detect such cases, especially for unsat instance with large field orders, is a task for future work.
- Gröbner basis approaches are saturation based, thus they have a conceptually advantage on unsatisfiable instances. This is due to the fact that Gröbner basis methods terminate once a non-zero constant is determined, which is why we feel inventing and employing extremely efficient monomial orderings would aid our work.
- It is notable that our approach seems to show complementary performance characteristics to existing Gröbner basis techniques, indicating that a portfolio approach could be valuable.

In summary, our current experiments show the general effectiveness of our approach, indicating also how present weaknesses can be mitigated with existing techniques from MCSat and CDCL solving (e.g. heuristics on the variable order, restarts, pre- and improcessing techniques to reduce clause complexity, clause deletion, etc.).

8 Related Work

Gröbner bases [8] and triangular sets [2,3] have been introduced to compute the solution space of polynomial equations, by reducing the degree of polynomials through variable elimination. Solving polynomial equations in general entails finding all of its solutions in the algebraic closure of the underlining coefficient field. Yet, for the purpose of satifiability, solutions in the base field are usually of the most interest. Obviously, there are only a finite number of solutions if the base field is finite; yet, enumerating all of the finitely numerous possibilities is not practically viable.

To limit the solutions of Gröbner bases and triangular sets to finite fields, a common technique is to introduce and add the set of field polynomials to the set of polynomial equations [16,22] Using field polynomials though greatly impacts practical performance, as showcased in [20]. Specialized ways for computing Gröbner bases and triangular sets over finite fields have therefore been created, such as the XL algorithm [9], F4 [13], and F5 [14] for Gröbner bases. Although all of these strategies are aimed at solving polynomial systems over finite fields, none of them explicitly address inequalities even though inequalities may be converted into equalities using the Rabinowitsch trick [10, 4.2 Prop. 8].

Optimization concepts for triangular sets have been introduced in [42], including efficient characteristic set algorithms [17,22] and polynomial decomposition into simple sets [26]. Although these approaches integrate reasoning over inequalities, none of them considers systems of clauses with polynomial constraints as needed for our SMT solving problem. Furthermore, they all require the generation of exponentially many sets to fully describe the systems. Our approach only explores a linear sized decomposition on demand.

A related approach to our search procedure is given in the hybrid framework of [5,6]. Here, a partial evaluation of the system is performed by fixing some variables before starting multiple Gröbner bases computations. Instead, in our work we show that subresultant regular subchain computations allows us to avoid working with Gröbner bases (and hence their double-exponential computational complexities).

Substantial progress has also been devoted to the problem of dealing with specific boolean polynomials, i.e. finite fields with only two elements. PolyBoRi [6, 7] is fairly effective in this domain, but it does not generalize towards arbitrary finite fields, which is the focus of our work.

Recently, an algebraic SMT decision technique for computing satisfiability of polynomial equalities/inequalities over large prime fields has been introduced in [39]. As polynomial systems are a subset of our polynomial constraint clauses, our work complements this effort, by also establishing a computational approach for deriving explanation clauses within MCSat reasoning.

9 Conclusion

We introduce a novel reasoning approach for determining the satisfiability of a given system of non-linear polynomial constraints over finite fields. As a framework, we adopt an MCSat decision procedure and expand it with a specific theory propagation rule that allows variable propagation over finite fields by adding so-called explanation clauses. To show the existence of these explanation clauses over finite fields, we apply zero decomposition procedures over polynomial constraints. Based on the structure of the polynomial system, we construct explanation clauses to resolve conflicting variable assignments. We distinguish between single polynomial projections and projections of multiple polynomials using subresultant regular subchains. Our work avoids using field polynomials while reducing the size of the projected polynomials.

We aim to further optimize our prototype through specific design decisions. For example, we will investigate the effect the variable order has on SMT solving over finite fields. Furthermore, we wish to improve performance if the given polynomial system is unsatisfiable; in this case, we are also interested in generating proof certificates. Finally, integrating our prototype within a high-performance SMT solver is another line for future work.

Acknowledgements. We thank Nikolaj Bjørner for the fruitful discussion on this work. We acknowledge partial support from the ERC Consolidator Grant ARTIST 101002685, the TU Wien SecInt Doctoral College, and the FWF SFB project SpyCoDe F8504.

References

[1] Anthony C. Hearn and the REDUCE developers: Reduce, http://www.reduce-algebra.com/, accessed: 03-13-2023

[2] Aubry, P., Lazard, D., Maza, M.M.: On the Theories of Triangular Sets. Journal of Symbolic Computation **28**(1-2), 105–124 (1999)

[3] Aubry, P., Maza, M.M.: Triangular Sets for Solving Polynomial Systems: a Comparative Implementation of Four Methods. Journal of Symbolic Computation **28**(1-2), 125–154 (1999)

[4] Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)

[5] Bettale, L., Faugere, J.C., Perret, L.: Hybrid Approach for Solving Multivariate Systems over Finite Fields. Journal of Mathematical Cryptology **3**(3), 177–197 (2009)

[6] Bettale, L., Faugère, J.C., Perret, L.: Solving Polynomial Systems over Finite Fields: Improved Analysis of the Hybrid Approach. In: ISSAC. pp. 67–74 (2012)

[7] Brickenstein, M., Dreyer, A.: PolyBoRi: A framework for Gröbner-basis Computations with Boolean Polynomials. Journal of Symbolic Computation **44**(9), 1326–1345 (2009)

[8] Buchberger, B.: Bruno Buchberger's PhD Thesis 1965: An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. Journal of Symbolic Computation **41**(3-4), 475–511 (2006)

[9] Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: EUROCRYPT. pp. 392–407. Springer (2000)

[10] Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Springer-Verlag New York (1997)

[11] Davenport, J.H., Heintz, J.: Real Quantifier Elimination is Doubly Exponential. Journal of Symbolic Computation **5**(1/2), 29–35 (1988)

[12] De Moura, L., Jovanović, D.: A Model-Constructing Satisfiability Calculus. In: VMCAI. pp. 1–12. Springer (2013)

[13] Faugere, J.C.: A New Efficient Algorithm for Computing Gröbner Bases (F4). Journal of Pure and Applied Algebra **139**(1-3), 61–88 (1999)

[14] Faugere, J.C.: A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In: ISSAC. pp. 75–83 (2002)

[15] Gallian, J.A.: Contemporary Abstract Algebra. Chapman and Hall/CRC (2021)

[16] Gao, S., Platzer, A., Clarke, E.M.: Quantifier Elimination over Finite Fields using Gröbner Bases. In: Algebraic Informatics. pp. 140–157 (2011)

[17] Gao, X.S., Huang, Z.: Characteristic Set Algorithms for Equation Solving in Finite Fields. Journal of Symbolic Computation **47**(6), 655–679 (2012)

[18] Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM Journal on Computing **18**(1), 186–208 (1989)

[19] Hader, T.: Non-Linear SMT-Reasoning over Finite Fields. Master's thesis, TU Wien, Vienna (Feb 2022)

[20] Hader, T., Kovács, L.: An SMT Approach for Solving Polynomials over Finite Fields. In: SMT. CEUR Workshop Proceedings, vol. 3185, pp. 90–98. CEUR-WS.org (2022)

[21] Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag, Berlin, Heidelberg (2003)

[22] Huang, Z.: Parametric Equation Solving and Quantifier Elimination in Finite Fields with the Characteristic Set Method. Journal of Systems Science and Complexity **25**(4), 778–791 (2012)

[23] Johnson, D., Menezes, A., Vanstone, S.: The Elliptic Curve Digital Signature Algorithm (ECDSA). Int. Journal of Information Security **1**(1), 36–63 (2001)

[24] Jovanović, D., De Moura, L.: Cutting to the Chase Solving Linear Integer Arithmetic. In: CADE. pp. 338–353 (2011)

[25] Jovanović, D., De Moura, L.: Solving Non-Linear Arithmetic. In: IJCAR. pp. 339–354 (2012)

[26] Li, X., Mou, C., Wang, D.: Decomposing Polynomial Sets into Simple Sets over Finite Fields: The Zero-Dimensional Case. Computers & Mathematics with Applications **60**(11), 2983–2997 (2010)

[27] MacWilliams, F., Sloane, N.: The Theory of Error-Correcting Codes. North-holland Publishing Company, 2nd edn. (1978)

[28] Maplesoft, a division of Waterloo Maple Inc..: Maple, `https://hadoop.apache.org`, accessed: 03-13-2023

[29] Matiyasevich, Y.: Hilbert's Tenth Problem. The MIT Press, Cambridge (1993)

[30] Moeller, B., Bolyard, N., Gupta, V., Blake-Wilson, S., Hawk, C.: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (2006)

[31] Mou, C.: Solving Polynomial Systems over Finite Fields: Algorithms, Implementation and Applications. Phd thesis, Université Pierre et Marie Curie (May 2013)

[32] Niu, W., Wang, D.: Algebraic Approaches to Stability Analysis of Biological Systems. Mathematics in Computer Science **1**, 507–539 (03 2008)

[33] Ozdemir, A., Kremer, G., Tinelli, C., Barrett, C.: Satisfiability modulo finite fields. Cryptology

ePrint Archive (2023)

[34] Silva, J.P.M., Sakallah, K.A.: GRASP—A New Search Algorithm for Satisfiability. In: The Best of ICCAD, pp. 73–89. Springer (2003)

[35] Stebila, D., Green, J.: Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer. RFC 5656 (2009)

[36] Stein, W., et al.: Sage Mathematics Software, http://www.sagemath.org, accessed: 03-13-2023

[37] Sturmfels, B.: Solving Systems of Polynomial Equations. No. no. 97 in CBMS Regional Conference Series in Mathematics, AMS (2002)

[38] Szabo, N.: Smart Contracts: Building Blocks for Digital Markets (1996), [Online]. Available: http://www.fon.hum.uva.nl

[39] Vella, L.C., Alt, L.: On Satisfiability of Polynomial Equations over Large Prime Fields. In: SMT. CEUR Workshop Proceedings, vol. 3185, pp. 114–127. CEUR-WS.org (2022)

[40] Wang, D.: An Elimination Method for Polynomial Systems. Journal of Symbolic Computation **16**(2), 83–114 (1993)

[41] Wang, D.: Computing Triangular Systems and Regular Systems. Journal of Symbolic Computation **30**(2), 221–236 (2000)

[42] Wang, D.: Elimination Methods. Springer Science & Business Media (2001)

[43] Wolfram Research, Inc.: Mathematica, Version 13.2, https://www.wolfram.com/mathematica, accessed: 03-13-2023