On Satisfiability of Polynomial Equations over Large Prime Fields

Lucas Clemente Vella[1], Leonardo Alt[1]

[1]*Ethereum Foundation*

Abstract

The importance of zkSNARKs [1] is ever-increasing in the cryptocurrency and smart contracts ecosystem. Due to the significant threat of financial loss a bug represents in such applications, there is also a surge of interest in the formal verification of zkSNARK programs. These programs are ultimately encoded as a system of polynomial equations over large finite prime fields, and to prove statements about such a program is to prove statements about its system of equations. In this ongoing work, we investigate algebraic techniques with the goal of writing a mixed algebraic-SMT decision procedure to compute satisfiability of a new theory of polynomials over large prime fields. Ideally, the new theory and decision procedure could be implemented in existing SMT solvers as well as a standalone tool, in order to perform verification tasks over real world applications of zkSNARKs.

Keywords

large prime fields, zkSNARKs, satisfiability modulo theories, arithmetic circuits, ethereum

1. Introduction

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARKs) [1] were originally designed so that a prover entity could convince a verifier entity that they know something, without revealing private data. More specifically, given a computable function[1] f and a value b such that $f(a) = b$, a zkSNARK cryptographic scheme allows Alice to prove she knows the value of a to Bob, who knows f and b, without disclosing the actual value. For example, given a hash function f and the hash value b, using zkSNARKs Alice could prove that she knows a preimage value a without revealing it.

The need for secrecy over publicly available data makes blockchains a fertile ground for zkSNARKs applications. This technology is fundamental, for example, for the ZCash [2] cryptocurrency, which allows completely private transactions. It is also important for an increasing number of smart contract applications deployed on public permissionless blockchains, such as Ethereum [3], where zkSNARKs are used both for privacy, as in Tornado Cash [4], maci [5] and Dark Forest [6], and to "outsource" heavy computation to outside the Ethereum network (off-chain), since it can be expensive (as in money) to execute complex programs on-chain. In these cases, only the zkSNARK verification of computation is done on-chain [7, 8].

SMT 2022: Satisfiability Modulo Theories, August 11–12, 2022, Haifa, Israel

lucas.vella@ethereum.org (L. C. Vella); leo@ethereum.org (L. Alt)

0000-0002-4998-9740 (L. C. Vella)

[1]Not any Turing computable function, because zkSNARKs encoding admits no loops or recursion.

```
def main(private u8 y, public u8 x) -> bool:
    return (y * y) == x
```

Figure 1: A very simple ZoKrates program.

A zkSNARK program is often written in a high level DSL, such as ZoKrates [9], Circom [10], Cairo [11], Zinc [12], Leo [13], Noir [14], and AirScript [15]. Compiling such a program corresponds to creating a constraint system that is equisatisfiable to $\exists x.f(x) = y$, consisting of a system of polynomials over a large finite prime field of the specific zkSNARK scheme being used. This constraint system is referred as an arithmetic *circuit* in the literature, as a loose analogy to digital circuits, where the variables are wires — some are considered inputs, some are internal and some are outputs. The commonly used proof systems for zkSNARKs use elliptic curves specifically built for this purpose [16, 17], which employ very large prime fields. For example, the order of the field used by elliptic curve BN128 has 254 bits[2].

The most complex zkSNARK applications have systems on the order of thousands of variables and millions of polynomial constraints, at which point the execution cost for generating a zkSNARK proofs becomes a limiting factor. Naturally, verifying properties of such systems is not any easier. Some verification tasks that are of particular interest to zkSNARKs application developers are:

- given a constraint system, check whether it represents a function, i.e. whether the output variables are uniquely determined by the input;
- given two constraint systems, check whether they are equisatisfiable;
- more generically, check whether a constraint system conforms to a specification.

In this extended abstract, we present our work in progress trying to build a decision procedure that can be used to perform automated verification of such systems and their properties in practice, mixing algebraic and SMT approaches. We survey the existing solutions considering the challenges that our context enforces, and how they may be handled.

1.1. Motivating Example

As an example, take the ZoKrates program in Figure 1, and consider there are two interacting persons: the verifier and the prover[3].

The verifier publishes a certain unsigned 8-bit integer x, and is interested in whether someone can prove that they know an unsigned 8-bit integer y such that y is the square root of x. ZoKrates compiles this code into the R1CS [18] form, which consists of a set of polynomial constraints of the form $a * b - c = 0$, where a, b and c are linear combinations of prime field elements. The full constraint system for the example above can be seen in [19].

After the system is published by the verifier, the prover privately generates a *witness*, that is, an assignment for all the variables in the constraint system, including the private y in our

[2]The exact value is 21888242871839275222246405745257275088548364400416034343698204186575808495617.

[3]The word "prover" here designates a party in the cryptosystem. Not to be confused with an automated reasoning engine.

```
1. b1 * b1 == b1
2. b2 * b2 == b2
3. b3 * b3 == b3
4. b4 * b4 == b4
5. b5 * b5 == b5
6. b6 * b6 == b6
7. b7 * b7 == b7
8. b8 * b8 == b8
9. (128 * b1 + 64 * b2 + 32 * b3 + 16 * b4 +
    8 * b5 + 4 * b6 + 2 * b7 + 1 * b8) == x
```

Figure 2: The set of constraints ZoKrates generates to represent x as an 8-bit number.

example. Using the witness, the constraint system, and the proving key published by the verifier, the prover generates a zero knowledge proof that they know y and that they have executed the circuit computations specified by the constraint system. Finally, the prover can submit the proof to a smart contract deployed by the verifier for an on-chain check that the proof is valid, without revealing any private data.

Besides the expected constraints for multiplication and equality, when compiling the ZoKrates high level code into circuit-like polynomial constraints the compiler also conveniently adds constraints that ensure that x and y are indeed 8-bit numbers, represented in Figure 2 for x (analogous for y).

Since x is a prime field element, the compiler decomposes x as a binary number of 8 bits in constraint 9. However, the variables $b1, \ldots, b8$ that represent those bits are also prime field elements. Thus, the compiler also adds constraints to ensure that each of those variables can only be 0 or 1.

Using current SMT solvers, we can represent the decomposition above as a predicate

$$\text{bits}(x, b1, b2, b3, b4, b5, b6, b7, b8)$$

in the theory of Nonlinear Integer Arithmetic (NIA) applying $(\text{mod } P)$ to both sides of each equality, where P is the order of the large prime field employed by zkSNARK proof system, x is the number to be decomposed, and if the predicate is true then $b1, \ldots, b8$ are the bits that compose x. Then we can ask whether this decomposition is deterministic via the assertion in Figure 3.

The full smtlib2 file can be found in [20]. If the SMT query is unsatisfiable, the decomposition is deterministic. Neither cvc5 [21] nor z3 [22] could prove this to be unsatisfiable (the expected result) within 1 minute. If we change the definition of the predicate to, for example, not constrain bit 2 to be binary (i.e. remove the constraint "`2. b2 * b2 == b2`" from Figure 2), cvc5 proves that the problem is satisfiable, showing how to generate different bit decompositions that lead to x. Having two different bit compositions leading to the same x allows us, for example, to craft a proof that convinces the verifier that $128^2 = 7$. In this case we say that the system is *underconstrained* which is what causes the nondeterminism. It is possible for nondeterminism to be harmless, but usually when analyzing functions nondeterminism is not desired.

```
(assert (and
    (bits x b1 b2 b3 b4 b5 b6 b7 b8)
    (bits x c1 c2 c3 c4 c5 c6 c7 c8)
    (not (and
        (= b1 c1)
        (= b2 c2)
        (= b3 c3)
        (= b4 c4)
        (= b5 c5)
        (= b6 c6)
        (= b7 c7)
        (= b8 c8)
    ))
))
```

Figure 3: SMT query to whether the binary decomposition of x is unique.

1.2. Related Work

SMT solvers can often handle large software verification problems, but there is no theory specifically written for large prime fields, and applying NIA solvers [23] is impractical in this scenario. Bit-Vector solvers struggle quite quickly given the required maximum bit-width of 256 bits. In a similar context to SMT solvers, a CDCL-like approach for solving polynomial systems over finite fields is presented in [24], however the technique used was limited to small fields.

Ecne [25] is a verification tool for zkSNARK programs specialized for deciding whether a given constraint system represents a function. It is based on a set of heuristics and substitution rules that are fast and work well in practice. The tool we envision should be capable of verifying more complex properties, where determinism would be one potential property.

There are several algebraic approaches that can solve the systems of polynomials we propose in theory, however the complexity of the presented algorithms is prohibitively high to be used in practice. We discuss these works in detail in Sec. 3.

2. Problem Statement

Given a system of polynomial statements, such as the following:

$$\begin{aligned} &4x^3 + 2x^2y^2 + 12x - y = 5 \\ &y^3 - 3x^2 \neq 7xy^3 + 1 \\ &y \neq x, \end{aligned}$$

our goal is to find if there is an assignment of the variables (x and y in this case) in a field modulo a certain prime p where all the statements are true.

It seems that inequations of the kind $x^2 \leq y$ do not make sense in a prime field because the values wrap around, so a total ordering among the elements is not quite meaningfully defined for

most applications of a satisfiability solver. Alternatively, ranges of the kind $a \leq x^2 - y < b$ look much more useful, but they are not being investigated in the scope of this work.

2.1. Description as a System of Equations

The problem above can be reduced to the problem of finding the zeros of a set of polynomials, i.e. the value of $\mathbf{x}$ such that $f_1(\mathbf{x}) = f_2(\mathbf{x}) = ... = f_m(\mathbf{x}) = 0$, where $\mathbf{x}$ is the vector $(x_1, x_2, ..., x_n)$. For equalities this is simple, as one can just rearrange all terms to the left-hand side, leaving 0 on the right-hand side. The equality in the example above becomes the polynomial

$$f_1 = 4x^3 + 2x^2y^2 + 12x - y - 5.$$

For disequalities, we can use the "Rabinowitsch trick" [26] (originally used in a proof of Hilbert's Nullstellensatz), to turn all the differences into a single equation. We first isolate all terms on the left-hand side, introduce a new variable z, and produce the equation

$$zg_1g_2...g_k - 1 = 0$$

where g_i are the polynomials such that $g_i \neq 0$. This works because the equation only has a solution if z is the inverse of $g_1g_2...g_k$, and zero has no inverse. In other words, if any of the g_i is zero, the term $zg_1g_2...g_k$ vanishes and the remaining equation $-1 = 0$ has no solutions.

In the above example, the disequalities produce the polynomial

$$f_2 = z(y^3 - 3x^2 - 7xy^3 - 1)(y - x) - 1.$$

Thus, we ultimately obtain a set of polynomials over a finite field, and the problem is reduced to finding their common roots in that same field. Like the set of real numbers $\mathbb{R}$, a finite field is not algebraically closed, and like $x^2 + 1 = 0$ over $\mathbb{R}$ have roots i and $-i$ in $\mathbb{C}$ (which is an extension field of $\mathbb{R}$), a polynomial over a finite field might have roots that only exists in an extension field. We are not interested in such roots.

2.1.1. Optimization Idea

Every polynomial over a field has a unique factorization of the kind $f = h_1h_2^2h_3^3 \dots h_k^k$, where the h_i are square-free (not divisible by any non-constant polynomial square) and pairwise coprime (have no common non-constant polynomial divisors). This is called square-free decomposition and is cheap to compute. It is easy to see that the polynomial $h_1h_2h_3 \dots h_k$ (the square-free component of f) has the same roots of f.

It might prove beneficial to replace the polynomials with their square-free components, (and $g_1g_2...g_k$ with its square-free component, for the disequality polynomial). This is because the square-free decomposition may reduce the degree and size of the polynomial, thus simplifying the solving procedure. The solution would be identical, except for the introduced variable z, which is not important to the problem.

3. Decision Procedure Discussion

Once we reduce our problem to a system of polynomial equations, we have to solve it. We now describe the approaches considered, and what we have already implemented. In this discussion, we assume the degrees of the input polynomials are small (not much larger than 10) and the prime field is large (around 2^{254}). For a definition of the basic algebra concepts used in this section (rings, ideals and irreducibility), check [27]. For a definition of the more advanced commutative algebra and algebraic geometry concepts (Lasker-Noether theorem, primary decomposition, algebraic sets, varieties and the classification of ideals) consult [28].

Before we try any expensive solving algorithm, let us consider some special cases that are easily handled. It is convenient to first try to reduce the set of polynomials against itself (reduction is a step of any Gröbner Basis algorithm, see Sec. 3.3), so that the system is sanitized (all trivial zeros and obviously equivalent polynomials are removed), and some easy to spot relations might be exposed, thus reducing the degree of some polynomials.

Special case #1: if the set of polynomials is empty, we consider it vacuously satisfiable and finish. This is consistent with the fact that the 0 polynomial is satisfiable, and that the initial reduction step removes all 0 polynomials from the set, so an empty set is equivalent to a set with only zero polynomials.

Special case #2: non-zero constant polynomial case: if the self-reduction returns a set with a non-zero constant polynomial (the reduction definition guarantees it will be the only polynomial left), it obviously cannot be zero, so the system is unsatisfiable. In the current implementation of reduction, if a non-zero constant polynomial is ever encountered, the algorithm is cut short and returns immediately.

We then iterate through all the polynomials and count 3 things: the maximum degree of the system d, the number of variables n, and the total number of zero degree terms t_z.

Special case #3: if $t_z = 0$, there are no zero degree terms. Thus $(0, 0, ..., 0)$ is a solution of the system, so we can stop.

Special case #4: if $d = 1$ (it cannot be 0 because that was checked in special cases #1 and #2) then the system is linear. Reduction is the generalization of Gaussian elimination, so whatever remained of the system is necessarily solvable (otherwise it would have finished in #2) and already in row echelon form — from which it is trivial to compute the unique solution — and we can stop.

Special case #5: if $n = 1$, reduction guarantees there will be only one polynomial left. Since it is monic, we can extract a root by using the Cantor-Zassenhaus probabilistic algorithm [29] (not yet implemented).

Special case #6: this is not yet implemented, and we are not sure if it is worth it. In case of a single polynomial with $n > 1$, [30] suggests factoring into irreducible factors: if any of the factors is absolutely irreducible (i.e. irreducible over any extension field, and there is a test for that [31]) then we consider it satisfiable. Plenty of solutions are guaranteed if d and n are small enough compared to the field size [30, 32, 33]. Otherwise, each factor f can then be handled independently: recursively test if the system $\{f = df/dx = 0\}$ is satisfiable, for some variable x where df/dx_i does not vanish. The problem is satisfiable iff at least one of these subsystems spawned by the factors is satisfiable.

We now discuss the alternatives being explored for the general case.

3.1. Fast Algebraic Geometry Method

If the set of polynomials has some particular properties, namely, it defines a prime ideal (an ideal that is both primary and radical), and it defines an absolutely irreducible algebraic set (a variety), then the algorithm in [32] allows for finding one common root in probabilistic polynomial time.

The problem is, of course, that this method is incomplete, as not all problems will obey these properties. It is unclear how many problems this technique would be able to solve on its own. It is also unclear what happens if the algorithm is executed on a polynomial set without those properties. Moreover, it is also unknown how to test whether an ideal is a radical without a full Gröbner Basis, and how to test whether an algebraic set is a variety. Nothing on this front has been implemented.

3.2. Robust Algebraic Geometry Method

There is another class of algorithms based on algebraic geometry (original probabilistic version given in [30], deterministic version given in [34]). Compared to the Fast Algebraic Geometry Method, instead of expecting the problem to be given as a variety, it recursively decomposes the problem's algebraic set until it either eventually reaches a variety (then deciding for satisfiable) or it is no longer possible to decompose, deciding for unsatisfiable.

This algorithm's complexity is double exponential on the number of variables, and only works if $p > d^{n^{O(n)}}$, where p is the order of the prime field, d is the maximum degree of the polynomials and n is the number of variables. Using our assumptions of $p \approx 2^{254}$ and $d \approx 10$, this algorithm does not work in general for much more than 3 variables. It is unclear for how many instances of the problem this algorithm might work in practice, but at least for the probabilistic version, this algorithm does not look practical at all.

Nothing of the algebraic set decomposition algorithm has been implemented. From here, we only implemented a test that tells whether a single multivariate polynomial that happens to be absolutely irreducible has any roots, originally from [33].

3.3. Methods based on Gröbner Basis

Finding the reduced Gröbner Basis [35] of the polynomial set is often the first step for solving the system. The best known algorithms for finding Gröbner Basis are Faugère's F4 [36] and F5 [37] algorithms, and they have worst case exponential space complexity. The Gröbner Basis is defined for a total order among the monomials (i.e. the variable product power parts such as x^3y^2), which implies a total order among the variables themselves. Each ordering has some useful properties: reverse degree lexicographical order (degrevlex), also called graded reverse lexicographic order (grevlex) is the fastest to calculate (exponential time instead of double exponential). Lexicographical ordering leaves the system in the triangular format suitable for variable elimination.

At the moment, we have implemented only a naive version of Buchberger's algorithm, which is much worse in practice than Faugère's F4 and F5 (but has the same worst case complexity). For a very small system it is capable of computing the Gröbner Basis in grevlex ordering, but we do not believe it would perform well for lexicographical ordering.

3.3.1. Triangular System

For this approach, we need the Gröbner Basis in lexicographical order, so that the system is in a triangular format where we can eliminate one variable at a time. However Gröbner Basis is much faster to compute on grevlex, so much that in practice lexicographical order is obtained by transforming from grevlex via Gröbner Walk [38] or similar. According to [39], this order conversion step is the slowest in this approach.

This yields a new set of polynomials with the same roots, but in a triangular format. Once the free variables are assigned, the system can be solved one variable at a time, so that the first polynomial has only one variable to be solved. This leaves the next polynomial with only one variable after replacing the variables already known, and so forth until the system is completely solved.

For instance, in 3 variable systems, if we define $x > y > z > 1$ in our ordering (1 must always be the smallest), then the first variable that can be solved is z, then y, then x, assuming the system is fully determined. If the system is not fully determined, and there are free variables, then the lowest variables are the ones to be assigned.

Having an algorithm to find the roots one variable at a time, we still have a branching problem, where each variable assignment leads to a new set of possible assignments for the next variable, and so on. Thus, the naive search expands into a typical NP problem.

Finding roots of a univariate polynomial in a prime field is relatively cheap, with the best known algorithm being factorizing the polynomial into unique factors $(x-a_1)(x-a_2)...(x-a_n)$ where a_i are the roots of the polynomial. The most efficient algorithm for factorization in a prime field is called the Cantor-Zassenhaus algorithm, with complexity $O(d^3 \log p)$ where d is the degree of the polynomial and p is the field's characteristic prime.

Besides the Gröbner Basis in grevlex, we have not yet implemented the techniques mentioned above, Gröbner Walk (or some alternative) and Cantor-Zassenhaus being the most notable absences.

The unsolved problems of this approach include how to choose the variable ordering optimally and a search for variable assignments in the triangular system that is better than exhaustive trial and error. It is unclear how to perform a local search in the triangular system, much less how to prove unsatisfiability.

3.3.2. Primary Decomposition Method

Given polynomials in the ring $\mathbb{F}[x_1, x_2, ..., x_n]$, every polynomial set defines an algebraic set, which is the set of points in $\mathbb{F}^n$ where all polynomials evaluate to zero. This is the set from where we seek solutions: by finding one point in the algebraic set, we have proven the problem to be satisfiable, and by proving the algebraic set to be empty, we have proven the problem to be unsatisfiable.

The set of polynomials we are trying to solve defines an ideal, and by Lasker-Noether theorem we know that such polynomial ideal is the intersection of a finite set of primary ideals (called the primary decomposition of the ideal), which implies any algebraic set is a union of irreducible algebraic sets, named varieties.

So the idea of this approach is to find the radical of the ideal defined by our polynomial set,

compute its primary decomposition, and for each prime ideal found (which is primary and radical), try to find a zero in its algebraic set. This seems to go back to the initial problem: we have a set of polynomials for which we must find a common zero point. The difference is that since we know we are dealing with an algebraic variety, we may be able to handle it with algebraic geometry methods such as the one described in [32].

This can also be the first approach of a hybrid method, where the bigger problem is reduced to smaller sub-problems that can then be handled in parallel. Each subproblem is attempted to be solved first by the Fast Algebraic Geometry method, and if that fails, we can apply different approaches such as the triangular system method, local search, CDCL or others.

Computing the Gröbner Basis is the first step of the primary decomposition and radical ideal algorithms, such as in [40].

The biggest issue of this method is that it is not complete: it decides for satisfiable if it is easy to extract a rational point from one of the algebraic sets (i.e. it is a variety) and it decides for unsatisfiable if the algebraic set is empty on the extension field (Gröbner Basis turns out to be a single non-zero constant). However, besides partitioning the problem, it does not help in case all the algebraic sets are reducible in the extension field.

3.3.3. Base Field Restriction

The satisfiability problem can be solved by finding the Gröbner Basis of the system extended with the equations

$$x^p - x = 0$$

for each variable x in the system (as per Fermat's little theorem, $x^p = x \mod p$). This removes all roots lying in an extension field but not in the base field.

For instance, for the system:

$$f_1 = 4x^3 + 2x^2y^2 + 12x - y - 5$$
$$f_2 = z(y^3 - 3x^2 - 7xy^3 - 1)(y - x) - 1$$

the new polynomials would be:

$$f_3 = x^p - x$$
$$f_4 = y^p - y$$
$$f_5 = z^p - z$$

where p is the field's characteristic prime.

The Gröbner Basis of the extended set contains a non-zero constant polynomial iff the system is unsatisfiable. Such Gröbner Basis can be calculated using any monomial ordering, such as degree reverse lexicographical order, which is cheaper to compute than the lexicographical order.

It is unclear how to turn this into a practical algorithm for the case where $p \gg 2$, as the procedure to compute the Gröbner Basis is prohibitively expensive due to the time complexity being $O(p^2)$ on the reduction step of $x^p - x = 0$. We suspect it can be done in $O(p \log p)$ by

using exponentiation by squaring, but even linear complexity on p is impractical, and would not help with the huge size that each reduced polynomial would have (on the order of $O(p)$).

3.4. Local Search Method

This is more a complementary approach than an alternative, as it cannot prove unsatisfiability, and it may find a solution quickly if they are abundant.

A local search is the standard way of approximating floating point solutions for most equation systems found in engineering and simulations. The problem is treated as a black box function (called objective function), and the input is varied (the solution you want) according to how it affects the output, usually following the gradient towards a local minimum. A solution is found when zero is reached.

The traditional way of building an objective function for systems of equations is to take some norm (usually L_2 or L_∞) of the residual vector. In case of polynomials, the elements of the residual vector are the evaluation of each polynomial at the given input. In case of prime fields, it is important to consider that the distance to 0 is at most $p/2$ when calculating the norm.

In SAT solving, discrete probabilistic methods (such as simulated annealing [41]) are used for local search, and that might be a good choice in our case, too. However, we also think it is worth to investigate classical mathematical methods, such as Powell, BFGS or even Newton's method, since the polynomials are differentiable (see [42] for these methods).

That said, the problem is much harder on the discrete case compared to continuous, because the search might narrow down to oscillate around zero, but never reach it because it is in between the minimum discrete steps. Such problem does not happen over reals. A metaheuristic such as Tabu Search [43] might be useful to exclude such convergent points, but a huge number of restarts might be unavoidable.

Some ideas that could be considered are: it might be beneficial to just use the equalities for the local search, removing the disequalities polynomial and considering each individual disequality as a constraint. If the disequalities happen to be useful in the search, it might be better to apply the Rabinowitsch trick individually for each one, because adding new variables is not a problem for this method, and this makes each restriction more explicit for the search algorithm.

3.5. CDCL-like Approach

The work in [23] describes a decision procedure for nonlinear constraints of polynomials over $\mathbb{Z}$ and its roots over $\mathbb{R}$. It is done by a CDCL-style procedure where a sequence of clauses and variable assignments is assembled one by one so that the total assignment is always consistent with the selected clauses.

When a conflict is detected, the set of conflicting polynomial constraints is identified, and Cylindrical Algebraic Decomposition (CAD) is used to isolate a whole region in the search space where this conflict can happen. This region is then described as a set of polynomial constraints that are added to the problem as learned clauses.

The difficulty of generalizing this technique for prime fields is that we need a way of generalizing a conflict in one particular variable evaluation to a whole region of the search space (the role of CAD in the original algorithm). Moreover, we also need to be able to describe it in a

meaningful way for the deduction system, i.e. define the new restrictions only in terms of $=$ and $\neq$, not with $>$, $\geq$, $<$ and $\leq$.

What also complicates the matter is that we are only interested in solutions that happen to be on the base field, not in the whole extension field, a restriction that [23] does not have to worry about.

The work in [24] attempts two different approaches to replace CAD for the prime field case: in one, a conflicting clause is found by calculating the Gröbner Basis on the remaining set of polynomials after evaluation on the partial variable assignment. The other approach uses elimination theory, which is able to handle constraints of form $f \neq 0$ directly, where f is a polynomial.

The problem of roots outside the base field is solved by including the polynomials $x^p - x$ for every variable x, which is most likely the main cause for the poor performance displayed in practice by the algorithms in [24], and limits the technique to very low values of p.

We suspect it might be possible to improve the approach in [24] by removing the $x^p - x$ polynomials from the conflict search, and handle such restrictions separately (as well as the common restriction for binary variables $x^2 - x$). If possible, this is a promising direction.

3.6. SIMPLEX Generalization

In linear systems, the search can be reduced by looking at the bounds of the variables with the SIMPLEX algorithm. It may be that there is a SIMPLEX generalization that works here to speed up the search. However, that is unlikely since SIMPLEX works by iteratively restricting the convex search space limited by straight lines/hyperplanes, but the curves designated by non-linear polynomials are not planes, so the limited space will not be convex.

This possibility is entertained for the sake of completeness and has not been considered in practice.

3.7. More Alternatives

An algorithm named Mutant Zhuang-Zi is given in [44] for solving a polynomial system over a finite field, and a somewhat promising algorithm for counting the solutions is given in [45]. However, at a first glance they do not seem to apply to large p due to including $x^p - x$ in the computations.

4. Conclusion

In this work in progress, we investigate how to decide satisfiability of a system of polynomials over a large prime field. We are also currently implementing a tool that will hopefully be used for the formal verification of zkSNARKs constraint systems using the investigated techniques.

It seems that the bane of large prime fields are the field equations $x^p - x = 0$. When used either with Gröbner Basis or other algorithms, these equations makes the running time (and space) of the algorithms a superlinear function of the prime number, which is so large that even $O(p)$ is impractical.

The role of field equations in the existing algorithms is to restrict the roots found on the prime field itself (i.e. filter out solutions found only on the extension field). Embedding them explicitly on the system to be solved might only work for small prime fields, but with creative indirect use, there are probabilistic algorithms such as Cantor-Zassenhaus and [32] that are able to sample roots over large prime fields (which have exponentially more roots than small prime fields [33]). So there is hope that at least proving the satisfiable cases is feasible for a large portion of the general case, whereas the unsatisfiable case might be much trickier.

Acknowledgement

The authors thank Barry Whitehat, Mahmut Levent Doğan, Nikolaj Bjørner and Thibaut Schaeffer for the valuable discussions and insights on Zero Knowledge Proofs, ZoKrates, polynomial constraint systems, Gröbner Basis and algebraic geometry.

References

[1] Zcash Team, What are zk-SNARKs?, 2022. URL: https://z.cash/technology/zksnarks/.
[2] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized Anonymous Payments from Bitcoin, in: 2014 IEEE Symposium on Security and Privacy, 2014, pp. 459–474. doi:10.1109/SP.2014.36.
[3] V. Buterin, Ethereum: A next-generation smart contract and decentralized application platform, 2014. URL: https://ethereum.org/en/whitepaper/.
[4] @ayefda, Introduction to Tornado Cash, 2022. URL: https://docs.tornado.cash/general/readme.
[5] B. WhiteHat, K. Tan, K. Gurkan, C.-C. Liang, , K. W. Jie, Minimum Anti-Collusion Infrastructure, 2022. URL: https://github.com/privacy-scaling-explorations/maci/blob/master/specs/01_introduction.md.
[6] D. F. Team, Announcing Dark Forest, 2020. URL: https://blog.zkga.me/announcing-darkforest.
[7] C. Smith, Zero Knowledge Rollups, 2022. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups.
[8] B. WhiteHat, Y. Tong, Initial Zero Knowledge Roll Up proposal, 2019. URL: https://github.com/barryWhiteHat/roll_up_token.
[9] ZoKrates contributors, ZoKrates documentation, 2022. URL: https://zokrates.github.io/.
[10] J. L. Muñoz-Tapia, Circom documentation, 2021. URL: https://docs.circom.io/.
[11] StarkWare Industries Ltd., Cairo documentation, 2021. URL: https://www.cairo-lang.org/docs/.
[12] A. Gluchowski, Release of Zinc v0.1, 2020. URL: https://blog.matter-labs.io/release-of-zinc-v0-1-8d949aa9a2f2.
[13] C. Chin, H. Wu, Leo documentation, 2022. URL: https://github.com/AleoHQ/leo.
[14] K. Wedderburn, Noir documentation, 2022. URL: https://noir-lang.github.io/book/index.html.

[15] B. Threadbare, AirScript repository, 2020. URL: https://github.com/GuildOfWeavers/AirScript.
[16] S. Bowe, BLS12-381: New zk-SNARK Elliptic Curve Construction, 2017. URL: https://electriccoin.co/blog/new-snark-curve/.
[17] Zcash Team, What is Jubjub?, 2019. URL: https://z.cash/technology/jubjub/.
[18] V. Buterin, Quadratic arithmetic programs: from zero to hero, 2016. URL: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.
[19] L. Alt, Example circuit in the ZoKrates intermediate representation, 2022. URL: https://gist.github.com/leonardoalt/05b0428bee4b09173b224e8c1b4fa15e.
[20] L. Alt, SMTLib2 query representing an 8-bit unsigned integer decomposition, 2022. URL: https://gist.github.com/leonardoalt/e4c6704be6ea3be288ddcef3c6924dfe.
[21] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A Versatile and Industrial-Strength SMT Solver, in: Tools and Algorithms for the Construction and Analysis of Systems, 2022, pp. 415–442. doi:10.1007/978-3-030-99524-9_24.
[22] L. de Moura, N. Bjørner, Z3: An efficient SMT solver, in: Tools and Algorithms for the Construction and Analysis of Systems, volume 4963 of *Lecture Notes in Computer Science*, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3_24.
[23] D. Jovanović, L. de Moura, Solving non-linear arithmetic, in: Automated Reasoning, 2012, pp. 339–354. doi:10.1007/978-3-642-31365-3_27.
[24] T. Hader, Non-Linear SMT-Reasoning over Finite Fields, Master's thesis, 2022. URL: https://repositum.tuwien.at/handle/20.500.12708/19502.
[25] F. Wang, Ecne: Automated verification of zk circuits, 2022. URL: https://0xparc.org/blog/ecne.
[26] W. D. Brownawell, Rabinowitsch trick, in: M. Hazewinkel (Ed.), Encyclopaedia of Mathematics, volume supplemental III, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, p. 323. URL: https://encyclopediaofmath.org/wiki/Rabinowitsch_trick.
[27] J. A. Gallian, Contemporary Abstract Algebra, CRC Press, Abingdon, Oxfordshire, England, 2021.
[28] D. Eisenbud, Commutative Algebra with a View Toward Algebraic Geometry, Graduate Texts in Mathematics, Springer, New York, NY, 1995. doi:10.1007/978-1-4612-5350-1.
[29] D. G. Cantor, H. Zassenhaus, A new algorithm for factoring polynomials over finite fields, Mathematics of Computation 36 (1981) 587–592. doi:10.1090/S0025-5718-1981-0606517-5.
[30] M.-D. Huang, Y.-C. Wong, Solving systems of polynomial congruences modulo a large prime, in: Proceedings of 37th Conference on Foundations of Computer Science, 1996, pp. 115–124. doi:10.1109/SFCS.1996.548470.
[31] E. Kaltofen, Fast parallel absolute irreducibility testing, Journal of Symbolic Computation 1 (1985) 57–67. doi:10.1016/S0747-7171(85)80029-8.
[32] A. Cafure, G. Matera, Fast computation of a rational point of a variety over a finite field, Mathematics of Computation 75 (2006) 2049–2085. doi:10.1090/S0025-5718-06-01878-3.

[33] W. M. Schmidt, A lower bound for the number of solutions of equations over finite fields, Journal of Number Theory 6 (1974) 448–480. doi:10.1016/0022-314X(74)90043-2.
[34] N. Kayal, Derandomizing some algebraic and number-theoretic algorithms, Ph.D. thesis, 2006. URL: https://www.microsoft.com/en-us/research/publication/derandomizing-some-algebraic-and-number-theoretic-algorithms/.
[35] B. Buchberger, M. Kauers, Groebner basis, 2010. URL: http://www.scholarpedia.org/article/Gr%C3%B6bner_basis.
[36] J.-C. Faugére, A new efficient algorithm for computing Gröbner bases (F4), Journal of Pure and Applied Algebra 139 (1999) 61–88. doi:10.1016/S0022-4049(99)00005-5.
[37] C. Eder, J.-C. Faugère, A survey on signature-based algorithms for computing Gröbner basis computations, Journal of Symbolic Computation (2016) 1–75. doi:10.1016/j.jsc.2016.07.031.
[38] S. Collart, M. Kalkbrener, D. Mall, Converting bases with the gröbner walk, Journal of Symbolic Computation 24 (1997) 465–469. doi:10.1006/jsco.1996.0145.
[39] C. Mou, Solving Polynomial Systems over Finite Fields: Algorithms, Implementation and Applications, Ph.D. thesis, Université Pierre et Marie Curie, 2013. URL: https://tel.archives-ouvertes.fr/tel-01110887.
[40] P. M. Gianni, B. M. Trager, G. Zacharias, Gröbner bases and primary decomposition of polynomial ideals, J. Symb. Comput. 6 (1988) 149–167. doi:10.1016/S0747-7171(88)80040-3.
[41] F. Martinez-Rios, J. Frausto-Solis, A simulated annealing algorithm for the satisfiability problem using dynamic markov chains with linear regression equilibrium, in: M. de Sales Guerra Tsuzuki (Ed.), Simulated Annealing, IntechOpen, Rijeka, 2012. doi:10.5772/46175.
[42] G. N. Vanderplaats, Numerical Optimization Techniques For Engineering Design, Vanderplaats Research & Development, Inc, 1999.
[43] F. Glover, M. Laguna, Tabu Search, Springer US, Boston, MA, 1998, pp. 2093–2229. doi:10.1007/978-1-4613-0303-9_33.
[44] J. Ding, D. S. Schmidt, Mutant Zhuang-Zi Algorithm, in: N. Sendrier (Ed.), Post-Quantum Cryptography, 2010, pp. 28–40. doi:10.1007/978-3-642-12929-2_3.
[45] S. Gao, Counting Zeros over Finite Fields with Gröbner Bases, 2009. URL: https://www.cs.cmu.edu/~sicung/papers/MS_thesis.pdf.