Rectification of Finite Field Arithmetic Circuits using Computer Algebra Techniques

VIKAS RAO, University of Utah, USA
HADEN ONDRICEK, University of Utah, USA
PRIYANK KALLA, University of Utah, USA
FLORIAN ENESCU, Georgia State University, USA

This article presents a complete and scalable symbolic computer algebra approach for rectification of faulty finite field arithmetic circuits at multiple nets. Contemporary approaches that utilize SAT solving, and Craig interpolation is infeasible in rectifying arithmetic circuits. Our approach represents the circuit as a system of polynomials and rectifies it against a polynomial specification by applying Gröbner basis (GB) based algorithms. Given a set of m candidate nets as rectification targets, first, we utilize algebra-based techniques to derive the necessary and sufficient conditions for the existence of a rectification function at the targets. Then, upon confirmation, we compute the patch functions collectively for the targets with variable support in primary inputs. For patch function computation, we present two approaches: a greedy approach which resolves the rectification functions for the targets, and an approach which explores a subset of don't care conditions for the targets. In this regard, we show how the algebraic computing model allows to explore the space of admissible rectification patches, collectively, for the m targets. Our approach is implemented as a custom software and utilizes existing open-source symbolic algebra libraries for computations. The core GB-based reduction computation on circuits is performed using the Boolean data-structure of Zero-suppressed Binary Decision Diagrams (ZDDs). We substantiate the approach with experimental results demonstrated over large operand width benchmarks up to 571 bits, including those that conform to the NIST-standard for ellyptic curve cryptography.

CCS Concepts: • **Hardware** → Electronic design automation.

Additional Key Words and Phrases: Logic Synthesis, Rectification, Arithmetic Circuits

1 INTRODUCTION

Debugging and rectification of digital logic circuits aims to correct a given defective circuit implementation to match its intended specification. As opposed to a complete redesign of the circuit, it is desirable to synthesize rectification sub-functions with minimal topological changes to the existing design – a problem often termed as *partial synthesis*. The process constitutes identifying candidate nets in the circuit as potential targets for rectification, followed by a check to ascertain the rectifiability of the circuit at these targets. If the check confirms that the targets admit correction, corresponding rectification functions are computed and synthesized to patch the circuit at these targets.

It is akin to performing synthesis for Engineering Change Order (ECO), wherein a highly optimized implementation is minimally modified to match the updated specification in a cost-effective way. This is achieved by reusing prior design efforts and avoiding rerunning the entire synthesis flow while adhering to the resource constraints and the physical design limitations.

The rectification problem has witnessed much research over the years – some of the earliest being [18, 28, 30]. Owing to a manifold improvement in the efficiency of SAT solvers, there has been a renewed interest in the problem over the last decade from the logic synthesis, testing, and verification communities [6, 11, 12, 31]. These techniques generally employ SAT, Quantified Boolean Formula (QBF) solving, and Craig Interpolation (CI) based techniques for rectification. While

Authors' addresses: Vikas Rao, University of Utah, Salt Lake City, USA, vikas.k.rao@utah.edu; Haden Ondricek, University of Utah, Salt Lake City, USA, haden.ondricek@utah.edu; Priyank Kalla, University of Utah, Salt Lake City, USA, kalla@ece.utah.edu; Florian Enescu, Georgia State University, Atlanta, USA, fenescu@gsu.edu.

successful for control-dominated applications, these techniques are computationally infeasible for the rectification of arithmetic circuits. Symbolic Computer Algebra (SCA) techniques are found to be more suitable for formal analysis and verification of arithmetic circuits. However, utilization of the various facets and capabilities of the SCA techniques for post-verification debugging and rectification has only recently begun to be addressed [8, 13, 14, 21, 25–27].

1.1 Motivation

This article addresses the problem of *multi-target rectification of faulty finite field arithmetic circuits*. Such circuits find applications in cryptography, error-control codes, RFID tags, testing of VLSI circuits, among others. Specifically, Elliptic Curve Cryptography (ECC) is one of the most important usage models where the public-key cryptography is designed on the algebraic structure of elliptic curves over finite fields. Due to its shorter key length and efficiency [20], ECC is fast becoming the encryption standard with applications in cryptocurrency transaction signing and securing web traffic. The main building blocks of ECC hardware implementations are fast, custom-built finite field arithmetic circuits, thus raising the potential for errors in the implementation, which have to be eventually rectified. It was shown [2] that incorrect cryptography hardware can lead to full leakage of the security key and even counterfeiting [32]. To secure data privacy and avoid vulnerabilities in crypto-systems, their rectification is imperative.

1.2 Problem Statement and Objective

We are given the following:

- As the specification, a multivariate polynomial f with coefficients in a finite field of 2^n elements (denoted $\mathbb{F}_{2^n}$), for a given $n \in \mathbb{Z}_{>0}$.
- An irreducible polynomial $P_n(x)$ of degree n with coefficients in $\{0, 1\}$ used to construct $\mathbb{F}_{2^n}$.
- A faulty circuit implementation C, with no assumptions on the number or the type of bugs present in C.
- A set $W = \{w_1, \ldots, w_m\}$ of m candidate targets from C, pre-specified or selected using contemporary signal selection heuristics [6, 16, 17].

The objective of our approach is to:

- Ascertain that C admits rectification at these m targets.
- Compute a set of individual rectification functions $U = \{u_1, \ldots, u_m\}$ for the corresponding targets.
- Derive *don't care conditions* corresponding to the m rectification functions.
- Synthesize the rectification polynomials into logic sub-circuit patches.

1.3 Approach and Contribution

We model the rectification problem using concepts from algebraic geometry and use symbolic computer algebra algorithms to determine rectifiability and to compute rectification functions. The specification f and implementation C are modeled in terms of polynomial ideals. The rectifiability check is formulated on these ideals using the Strong Nullstellensatz over finite fields. Subsequently, the rectification functions are computed using the Gröbner bases [1] of these ideals. In this regard, our approach goes beyond the Nullstellensatz-based results produced from the m-target rectifiability check presented in [26], and *computes* m individual rectification functions, altogether. We show how, in our algebraic model the don't cares correspond to *varieties of polynomial ideals*, and how they can be computed with Gröbner bases. The rectification functions are computed as polynomials whose common zeros correspond to the minterms of the functions. Overall, our contributions are as follows:

- Formulate a check to determine the rectifiability of C at the m targets.
  - This check implies the existence of a polynomial function over $\mathbb{F}_{2^n}$ with mapping u_i : $\mathbb{F}_2^{|X_{PI}|} \longrightarrow \mathbb{F}_2$ at individual targets w_i. Here, $1 \leq i \leq m$, $\mathbb{F}_2 = \{0, 1\}$, and X_{PI} denotes the set of primary input variables of C.
  - Substituting the polynomial function at the corresponding targets, $w_i = u_i[X_{PI}]$, rectifies the circuit.
- While there may exist multiple rectification functions for each of the m targets, we compute one such individual function for each of the targets. We present two approaches for patch function computation:
  - First, a heuristic which greedily resolves the rectification functions for the targets.
  - A second heuristic which explores a subset of *don't care conditions* for the targets.
    - We present efficient techniques to explore the space of various admissible rectification functions, in turn, computing subsets of don't care conditions which helps in simplifying the rectification patches.
    - Synthesis of the corresponding polynomial patch functions along with don't cares, demonstrates the efficacy of our approach in terms of improved area and delay characteristics of the patches.
- We further propose a synthesis procedure that can translate such polynomial functions to Boolean rectification functions. Subsequently, these functions are translated to Boolean logic by converting the algebraic multiplication and addition operations to the Boolean AND and XOR operations, respectively.
- We present theoretical concepts and algorithms, and their implementations, to provide a scalable and efficient solution for the rectification of finite field arithmetic circuits.
- We substantiate the efficacy of our techniques by rectifying large operand-width finite field arithmetic circuits, where conventional SAT-solver based rectification approaches are infeasible.
  - We reason about the presence or absence of solutions and other properties of a system of polynomials without explicitly solving them. In contrast, the contemporary approaches explicitly solve for a solution at each step and hence are infeasible towards rectification of arithmetic circuits.

The article is organized as follows: The following section covers preliminary background. Section 3.1 reviews the polynomial modeling concepts. Rectification check formulation is described in Section 4, followed by the rectification function and don't care computations in Section 5. Section 6 discusses the implementation details, and experimental results are described in Section 7, and Section 8 concludes the article.

2 REVIEW OF PRIOR WORK

Contemporary approaches formulate rectification using QBF solving [12], using CI or iterative SAT solving [31, 33]. The rectification techniques in [9, 16, 31, 34] iteratively and incrementally compute multiple single-fix functions that partially patch the circuit in each iteration. They ensure that, in each iteration, erroneous minterms are resolved and no new errors are introduced, eventually converging the circuit to the given specification. Recent techniques incorporate resource awareness in patch generation by re-expressing the obtained Skolem functions in terms of internal signals [6], employ improved heuristics for target selection [16], or resolve a combination of such objectives, such as the symbolic sampling approach of [17]. In [17] the authors propose a robust ECO approach to derive patches with minimal impact on the heavily optimized existing implementation against a structurally dissimilar ECO-evolved specification. They enumerate rectification points functionally by simulation and match the circuitry of patches implicitly to maximize the reuse of existing logic in the implementation. To achieve scalability, the method proposes modeling and analyzing its computations in symbolic sampling domain. However, circuits that implement polynomial computations over large bit-vector operands are hard to rectify using models based on Boolean function, SAT/SMT-solvers, etc. Our experiments show that the contemporary SAT solvers fail to rectify finite field circuits beyond 16-bit operands.

2.1 Symbolic Computer Algebra

In the context of arithmetic circuits, symbolic computer algebra techniques for integer arithmetic [7, 8, 21] and finite field circuits [13, 14, 25] have been considered for rectification. The rectification approaches presented in [7, 8] rely heavily on the structure of the arithmetic circuit. Further, if the circuit contains redundancies, their approach fails [24] to resolve the rectification question. Further, the approach of [21] is limited by the fault model, which addresses only a gate misplacement fault. The authors in [13, 14] present a Weak Nullstellensatz based rectification formulation to determine the existence of rectification functions and solve them using GB-based techniques. Subsequently, the rectification function is computed by the application of Craig Interpolation in polynomial algebra over finite fields. The authors show how the lattice of all algebraic interpolants correspond to don't care conditions for the rectification function, and synthesize rectification functions from algebraic interpolants. As an alternative to CI, in [25] we presented an approach where rectification was formulated using ideal membership and extended GB theory. However, all these algebraic approaches address only *single-fix rectification* – where irrespective of the type or number of bugs in the circuit, rectification is attempted at a single net. This is too restrictive, and depending on the nature of the bugs, the circuit may not admit single-fix rectification at all. In such cases, the correction has to be attempted at multiple targets.

Recently, we proposed a word-level SCA based approach [26] to *decide multi-target rectifiability* in finite field circuits. Given a set of m-targets within a n-bit operand width circuit, the approach presents a Strong Nullstellensatz based decision procedure to determine the existence of rectification functions at these targets. The efficiency of the approach is derived by interpreting the m-targets as a m-bit-vector. The problem was modeled at a word-level to enable the use of higher level abstraction in rectification and synthesis. However, enabling such word-level reasoning might induce computation issues across elements from the n-bit-vector space (circuit) and the m-bit-vector space (rectification patch). To resolve this incompatibility, we presented theory and mathematical derivations that facilitate algebraic computations over a unified domain modeled over $k = LCM(n, m)$. However, for finite field circuits, with practical applications in cryptography, the operand width n as dictated by the National Institute of Standards and Technology (NIST) is generally a prime number. For example, NIST-ECC recommends n to be 163 or larger (n= 163, 233, 283, 409, 571, etc.). Thus, for any given target size m, $k = LCM(n, m) = n * m$ becomes extremely large. This results in algebraic computations involving substantially large primitive elements and constants. The large size coupled with the complicated arithmetic nature of these circuits increases the complexity of the rectification problem. Moreover, the proposed approach [26] can only ascertain whether *there exists* a set of functions that can patch the circuit at the targets. As it is only a decision procedure, the approach cannot *compute* rectification functions. Thus the problem of multi-fix rectification of data-path circuits remains unsolved, and theoretical and algorithmic solutions to compute and synthesize rectification patches for arithmetic circuits are still desired.

3 PRELIMINARIES

This section reviews some basic concepts from symbolic computer algebra and associated algorithms that are utilized in this article.

Finite fields: Let $\mathbb{B}$ denote the Boolean domain, and $\neg, \wedge, \vee, \oplus$ the NOT, AND, OR and XOR operators, respectively. Let $\mathbb{F}_2 = \{0, 1\}$ be the field of 2 elements, and let $\mathbb{F}_q = \mathbb{F}_{2^n}$ denote the finite field of $q = 2^n$ elements, for a given $n \in \mathbb{Z}_{>0}$. We denote $\overline{\mathbb{F}}_q$ as the algebraic closure of $\mathbb{F}_q$. $\mathbb{F}_{2^n}$ is the n-dimensional extension of $\mathbb{F}_2$, and it is constructed as $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$. Here $P_n(x) \in \mathbb{F}_2[x]$ is a given degree-n polynomial, *irreducible* in $\mathbb{F}_2$, with a root γ, i.e. $P_n(\gamma) = 0$, which is a generator of $\mathbb{F}_{2^n}$. Then γ is called a primitive element (PE) of $\mathbb{F}_{2^n}$ and it generates the entire field: $\mathbb{F}_{2^n} = \{0, 1 = \gamma^{2^n-1}, \gamma, \gamma^2, \ldots, \gamma^{2^n-2}\}$. Note: $\forall \gamma \in \mathbb{F}_{2^n}, \gamma^{2^n-1} = 1$. An element $A \in \mathbb{F}_{2^n}$ can be written as $A = a_0 + a_1 \cdot \gamma + \cdots + a_{n-1} \cdot \gamma^{n-1}$, where $a_0, \ldots, a_{n-1} \in \mathbb{F}_2$. In $\mathbb{F}_{2^n}$, the addition $(''+'')$ and multiplication $(''\cdot'')$ operations are performed in the base field $\mathbb{F}_2$ and reduced modulo the corresponding primitive polynomial $P_n(x)$. For $n, k \in \mathbb{Z}_{>0}$, if n divides k, then $\mathbb{F}_{2^n} \subset \mathbb{F}_{2^k}$. Thus, $\mathbb{F}_2 \subset \mathbb{F}_{2^n}, \forall n > 1$. All fields of the type $\mathbb{F}_{2^n}$ have characteristic 2, and therefore $-1 = +1$ in $\mathbb{F}_{2^n}$.

Polynomial representation: Let $R = \mathbb{F}_q[x_1, \ldots, x_d]$ be the polynomial ring in variables $x_1, \ldots, x_d$ with coefficients in $\mathbb{F}_q$. A polynomial $p \in R$ is written as a finite sum of terms $f = c_1M_1 + c_2M_2 + \cdots + c_rM_r$, where $c_1, \ldots, c_r$ are coefficients from $\mathbb{F}_q$ and $M_1, \ldots, M_r$ are monomials, i.e. power products of the type $x_1^{e_1} \cdot x_2^{e_2} \cdots x_d^{e_d}$, $e_j \in \mathbb{Z}_{\geq 0}$. To systematically manipulate the polynomials, a monomial order $>$ (or a term order) is imposed on R such that the monomials of all polynomials are ordered according to $>$. Subject to $>$, we have that $M_1 > M_2 > \cdots > M_r$. Then, $lc(p) = c_1$, $lm(p) = M_1$, and $lt(p) = c_1M_1$ denote the *leading coefficient, leading monomial*, and *leading term* of p, respectively. Also, for $p \in R$, $tail(p) = p - lt(p)$. This work employs *lexicographic* (lex) term orders.

Polynomial Reduction via division: Let $p,\ g$ be polynomials. If $lm(p)$ is divisible by $lm(g)$, then we say that p *is reducible to* r modulo g, denoted $p \xrightarrow{g} r$, where $r = p - \frac{lt(p)}{lt(g)} \cdot g$. This operation forms the core of polynomial division algorithms and it has the effect of canceling the leading term of p to obtain r. Similarly, p can be reduced w.r.t a set of polynomials $F = \{f_1, \ldots, f_s\}$. Then $p \xrightarrow{F}_+ r$ denotes the *reduction* of p modulo the set of polynomials F resulting in a remainder r, obtained by iteratively canceling terms in p by $lt(f_j)$, $f_j \in F$, via polynomial division. The remainder r is said to be *reduced* such that no term in r is divisible by the leading term of any $f_j \in F$.

Algorithm 1 (Alg. 1.5.1 [1]) presents the procedure which performs polynomial reduction by iteratively canceling one monomial at a time. Along with the remainder r, the algorithm also returns the set of quotients $\{u_1, \ldots, u_s\}$ of division of p by $\{f_1, \ldots, f_s\}$, respectively, such that $p = u_1 \cdot f_1 + \cdots + u_s \cdot f_s + r$.

Algorithm 1 Multivariate Reduction of p by $F = \{f_1, \ldots, f_s\}$

1: **procedure** *multi_var_division*($p, \{f_1, \ldots, f_s\}, f_j \neq 0$)
2: $\quad u_j \leftarrow 0;\ r \leftarrow 0,\ h \leftarrow p$
3: $\quad$ **while** $h \neq 0$ **do**
4: $\quad\quad$ **if** $\exists j$ s.t. $lm(f_j) \mid lm(h)$ **then**
5: $\quad\quad\quad$ choose j least s.t. $lm(f_j) \mid lm(h)$
6: $\quad\quad\quad u_j = u_j + \frac{lt(h)}{lt(f_j)}$
7: $\quad\quad\quad h = h - \frac{lt(h)}{lt(f_j)} f_j$
8: $\quad\quad$ **else**
9: $\quad\quad\quad r = r + lt(h)$
10: $\quad\quad\quad h = h - lt(h)$
11: $\quad$ **return** $(\{u_1, \ldots, u_s\}, r)$

Ideals and varieties: A given set of polynomials $F = \{f_1, \dots, f_s\}$ from R, generates the **ideal** $J = \langle F \rangle \subseteq R$, defined as $J = \langle f_1, \dots, f_s \rangle = \{h_1 \cdot f_1 + \dots + h_s \cdot f_s \mid h_1, \dots, h_s \in R\}$. Polynomials $f_1, \dots, f_s$ form the generators or the *basis* of ideal J.

Let $\boldsymbol{a} = (a_1, \dots, a_d) \in \mathbb{F}_q^d$ be a point in the affine space, and p a polynomial in R. If $p(\boldsymbol{a}) = 0$, we say that p *vanishes* on $\boldsymbol{a}$, or $\boldsymbol{a}$ is a *zero* of the polynomial p. In this work, we have to analyze the *set of all common zeros* of the polynomials of $F = \{f_1, \dots, f_s\}$ that lie within the field $\mathbb{F}_q$ – i.e. the set of all point $\boldsymbol{a} \in \mathbb{F}_q^d$ such that $f_1(\boldsymbol{a}) = \dots = f_s(\boldsymbol{a}) = 0$. This zero set is called the **variety**, which depends not just on the given set of polynomials in F, but rather on the ideal generated by polynomials. We denote it by $V(J)$, where: $V(J) = V_{\mathbb{F}_q}(J) = V_{\mathbb{F}_q}(\langle f_1, \dots, f_s \rangle) = \{\boldsymbol{a} \in \mathbb{F}_q^d : \forall f \in J, f(\boldsymbol{a}) = 0\}$.

Boolean functions comprise a finite set of points, so we can model them as varieties. If a point is an element of a variety, then that point can be considered either an on-, off-, or DC-set minterm of a corresponding Boolean function. Algebraic geometry analyzes ideals to reason about varieties without explicitly computing the varieties, which is infeasible in practice. In this paper, we compute the union, intersection, and set difference of functions, represented by varieties, by performing corresponding operations on ideals. The first two columns of Table 1 describe the correspondence between operations on Boolean functions and operations on varieties. We utilize this correspondence to formulate rectification check and to construct rectification functions by modeling the on-, off-, and DC-sets as varieties.

Table 1. Correspondences between algebraic operations and Boolean operations.

Boolean Functions	*Varieties*	*Ideal operations*
$f \vee g$	$V(J_1) \cup V(J_2)$	$J_1 * J_2$
$f \wedge g$	$V(J_1) \cap V(J_2)$	$J_1 + J_2$
$f - g$	$V(J_1) \setminus V(J_2)$	$J_1 : J_2$

Given two ideals $J_1 = \langle f_1, \dots, f_s \rangle$, $J_2 = \langle h_1, \dots, h_r \rangle$, the **sum of ideals** is denoted as $J_1 + J_2 = \langle f_1, \dots, f_s, h_1, \dots, h_r \rangle$, their **product** is given as $J_1 * J_2 = \langle f_i \cdot h_j : 1 \le i \le s, 1 \le j \le r \rangle$, and a **colon ideal** operation is defined as $J_1 : J_2 = \{f \in R \mid f \cdot g \in J_1, \forall g \in J_2\}$. Ideals and varieties are dual concepts: $V(J_1 + J_2) = V(J_1) \cap V(J_2)$, whereas $V(J_1 * J_2) = V(J_1) \cup V(J_2)$. A colon ideal operation corresponds the set difference of two varieties, $V_{\mathbb{F}_{2^n}}(J_1 : J_2) = V_{\mathbb{F}_{2^n}}(J_1) \setminus V_{\mathbb{F}_{2^n}}(J_2)$. These ideal operations are implemented in computer algebra tools, which we utilize.

Gröbner Bases: An ideal may have many different sets of generators such that their varieties are the same. A *Gröbner basis* (GB) is one such generating set with special properties that allows to solve many polynomial decision and quantification problems.

Definition 3.1. **[Gröbner Basis]** [1]: *For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner Basis (GB) of J $\iff$ $\forall f \in J$, $f \xrightarrow{g_1, \dots, g_t}_{+} 0$.*

The GB G for an ideal J can be computed using the Buchberger's algorithm [5]. The algorithm, shown in Alg. 2, takes the set of polynomials $F = \{f_1, \dots, f_s\}$ as input and computes their GB $G = \{g_1, \dots, g_t\}$., such that $J = \langle F \rangle = \langle G \rangle$. Moreover, the polynomials of F and G have the same solution-set, i.e. $V(J) = V(\langle F \rangle) = V(\langle G \rangle)$.

In the algorithm,

$$Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j \tag{1}$$

where $L = LCM(lm(f_i), lm(f_j))$, and LCM denotes the least common multiple.

Algorithm 2 Buchberger's Algorithm

Require: $F = \{f_1, \dots, f_s\}$
Ensure: $G = \{g_1, \dots, g_t\}$
1: $G := F$;
2: **while** $G' \neq G$ **do**
3: $\quad G' := G$
4: $\quad$ **for** each pair $\{f_i, f_j\}, i \neq j$ in G' **do**
5: $\quad\quad Spoly(f_i, f_j) \xrightarrow{G'}_{+} h$
6: $\quad\quad$ **if** $h \neq 0$ **then**
7: $\quad\quad\quad G := G \cup \{h\}$

Ideal Membership Test: The above definition inherently provides a decision procedure to test for membership of a polynomial in an ideal: f is a member of an ideal J *if and only if* $f \xrightarrow{GB(J)}_{+} 0$. When $f \notin J$, then division by $GB(J)$ results in a non-zero remainder r that is unique/canonical.

A Gröbner basis can be further reduced. A reduced Gröbner basis (redGB) is canonical representation of the ideal J.

Definition 3.2. **[Reduced Gröbner Basis]**: *A redGB for a polynomial ideal J is a GB $G = \{g_1, \dots, g_t\}$ such that:*

- $lc(g_i) = 1, \forall g_i \in G$
- *$\forall g_i \in G$, no monomial of g_i lies in $\langle lt(G - \{g_i\})\rangle$*

The Strong Nullstellensatz in Finite Fields: For any element $\varphi \in \mathbb{F}_q$, $\varphi^q = \varphi$ holds. Therefore, the polynomial $x^q - x$ vanishes everywhere in $\mathbb{F}_q$, and we call it a *vanishing polynomial*. We denote by $F_0 = \{x_1^q - x_1, \dots, x_d^q - x_d\}$ the set of all vanishing polynomials in R, and $J_0 = \langle F_0 \rangle$ denotes the ideal of all vanishing polynomials in R. Then, $V_{\mathbb{F}_q}(J_0) = V_{\overline{\mathbb{F}}_q}(J_0) = \mathbb{F}_q^d$. Moreover, given any ideal J, $V_{\mathbb{F}_q}(J) = V_{\overline{\mathbb{F}}_q}(J) \cap \mathbb{F}_q^d = V_{\overline{\mathbb{F}}_q}(J) \cap V_{\overline{\mathbb{F}}_q}(J_0) = V_{\overline{\mathbb{F}}_q}(J + J_0) = V_{\mathbb{F}_q}(J + J_0)$.

Definition 3.3. *Given an ideal $J \subset R$ and $V(J) \subseteq \mathbb{F}_q^d$, the ideal of polynomials that vanish on $V(J)$ is $I(V(J)) = \{f \in R : \forall \boldsymbol{a} \in V(J), f(\boldsymbol{a}) = 0\}$.*

If f vanishes on $V(J)$, then $f \in I(V(J))$. The Strong Nullstellensatz, which has a special form over finite fields, characterizes the ideal $I(V(J))$.

Theorem 3.1 (*The Strong Nullstellensatz over finite fields (Theorem 3.2 in [10])*). *For any ideal $J \subset \mathbb{F}_q[x_1, \dots, x_d]$, $I(V_{\mathbb{F}_q}(J)) = J + J_0$.*

The computational complexity of GB algorithm is exponential in the number of variables d. For efficient GB computations on polynomials derived from a circuit, [19] proposed the use of a specialized term order that exploits the topology of the circuit and renders the set of polynomials for the gates of the circuit, a Gröbner basis itself. This term order '>' is called the *Reverse Topological Term Order (RTTO).*

Definition 3.4. **[Reverse Topological Term Order]** *[19]: Let C be an arbitrary combinational circuit. Let $\{x_1, \dots, x_d\}$ denote the set of all variables (signals) in C. Starting from the primary outputs, perform a reverse topological traversal of the circuit and order the variables such that $x_k > x_j$ if x_k appears earlier in the reverse topological order. Impose a lex term order $>$ to represent each gate as a polynomial f_j, s.t. $f_j = x_k + tail(f_j)$. Then set of polynomials $\{f_1, \dots, f_s\}$ corresponding to the gates of the circuits is a Gröbner basis when RTTO is used for ordering.*

3.1 Modeling Circuits with Polynomial Ideals

A multivariate polynomial f over $\mathbb{F}_{2^n}$ is given as a *Spec*, where n is the operand word-length (data-path size). A combinational circuit C is given as its (faulty) implementation. The function implemented by C is modeled with a system of polynomials over $R = \mathbb{F}_{2^n}[Z, A, X]$, where $X = \{x_1, \dots, x_d\}$ corresponds to all the bit-level variables (nets) in the circuit. Let $X_{PO} \subset X$, and $X_{PI} \subset X$ denote the set of all primary output variables, and primary input variables from C, respectively. Further, $Z = \{z_0, \dots, z_{n-1}\}$ and $A = \{a_0, \dots, a_{n-1}\}$ represent the output and input operand words of the circuit, respectively, where $z_i \in X_{PO}$ and $a_i \in X_{PI}$. As C comprises Boolean logic gates, the gates are represented by polynomials $\pmod 2$, i.e. over $\mathbb{F}_2(\subset \mathbb{F}_{2^n})$, using the mapping $\mathbb{B} \mapsto \mathbb{F}_2$:

$$\begin{aligned} z = \neg a \mapsto z + a + 1; \qquad & z = a \wedge b \mapsto z + a \cdot b; \\ z = a \vee b \mapsto z + a + b + a \cdot b; \qquad & z = a \oplus b \mapsto z + a + b; \end{aligned} \tag{2}$$

Algebraically, the correspondences between the bit-level and word-level variables of the circuit are represented as:

$$\begin{aligned} f_1 : Z = z_0 + \gamma \cdot z_1 + \cdots + \gamma^{n-1} \cdot z_{n-1}, \\ f_2 : A = a_0 + \gamma \cdot a_1 + \cdots + \gamma^{n-1} \cdot a_{n-1}, \end{aligned} \tag{3}$$

where $P_n(\gamma) = 0$. Since $\mathbb{F}_2 \subset \mathbb{F}_{2^n}$, the polynomials in Eqn. (2) can also be interpreted as polynomials over $\mathbb{F}_{2^n}$. Thus, the circuit is represented by a set of polynomials $F = \{f_1, \dots, f_s\} \subset R$. Let $J = \langle F \rangle$ be the ideal generated by this set. Let $F_0 = \{x_i^2 - x_i, Y^{2^n} - Y \mid x_i \in$ bit-level variables, $Y \in$ word-level variables$\}$ be the set of all vanishing polynomials, and $J_0 = \langle F_0 \rangle$ the corresponding ideal. Then, ideal $J + J_0 = \langle F \cup F_0 \rangle$ models the functionality of *Impl* C.

One can verify the correctness of the circuit C by checking if the given *Spec* is implied by the ideal representing C. In other words, $f \equiv C \iff f \xrightarrow{GB(J+J_0)}_+ 0$ [19]. In the manuscript, we use the circuit of Fig. 1, which is borrowed from [26], as a running example to demonstrate our algebraic approach for MFR.

Example 3.1. *The circuit C in Fig. 1 is a faulty implementation of a 3-bit (n=3) Mastrovito multiplier. The field $\mathbb{F}_{2^3}$ is constructed using $P_3(x) = x^3 + x + 1$ with γ as a root, i.e. $P_3(\gamma) = 0$. The Spec polynomial is $f : Z + A \cdot B$, where Z is the output word, and A, B the input words. Impose RTTO $>$ on the polynomials, i.e. a lex term order on all polynomials with the variables of C ordered reverse topologically from POs to PIs: $\{Z\} > \{A > B\} > \{z_0 > z_1 > z_2\} > \cdots > \{d_1 > d_2 > d_3 > r_0 > d_5 > rr_1\} > \{r_1 > rr_3 > rr_2\} > \{r_2 > r_3 > rr_4\} > \{r_4 > d_4\} > \{a_0 > a_1 > a_2 > b_0 > b_1 > b_2\}$.*

Under RTTO $>$, the following polynomials represent C:

$$\begin{aligned} f_1 : Z + z_0 + \gamma \cdot z_1 + \gamma^2 \cdot z_2; \quad & f_{22} : rr_1 + rr_3 + rr_2; \\ f_2 : A + a_0 + \gamma \cdot a_1 + \gamma^2 \cdot a_2; \quad & f_{23} : r_1 + r_2 + r_3; \\ f_3 : B + b_0 + \gamma \cdot b_1 + \gamma^2 \cdot b_2; \quad & f_{26} : r_3 + r_4 + d_4; \\ f_4 : z_0 + d_0 + e_1; \quad & f_{27} : rr_3 + rr_4 + b_2; \dots \\ \dots \quad & f_{30} : rr_4 + a_2 + b_2 + a_2 b_2; \end{aligned}$$

where the polynomials f_{26}, f_{27} correspond to the introduced bugs. Then $F = \{f_1, \dots, f_{30}\}$, $F_0 = \{a_0^2 - a_0, \dots, z_2^2 - z_2, A^8 - A, B^8 - B, Z^8 - Z\}$. So, ideal $J + J_0 = \langle F \cup F_0 \rangle$ encapsulates the function implemented by C. Computing $f : Z + A \cdot B \xrightarrow{F, F_0}_+ r$ results in $r = \gamma^2(a_1 a_2 b_1 b_2 + a_1 a_2 b_2 + a_2 b_1 b_2) + \gamma^1(a_1 a_2 b_1 b_2 + a_1 a_2 b_2) + \gamma^0(a_2 b_1 b_2)$. Since $r \neq 0$, the circuit is buggy.

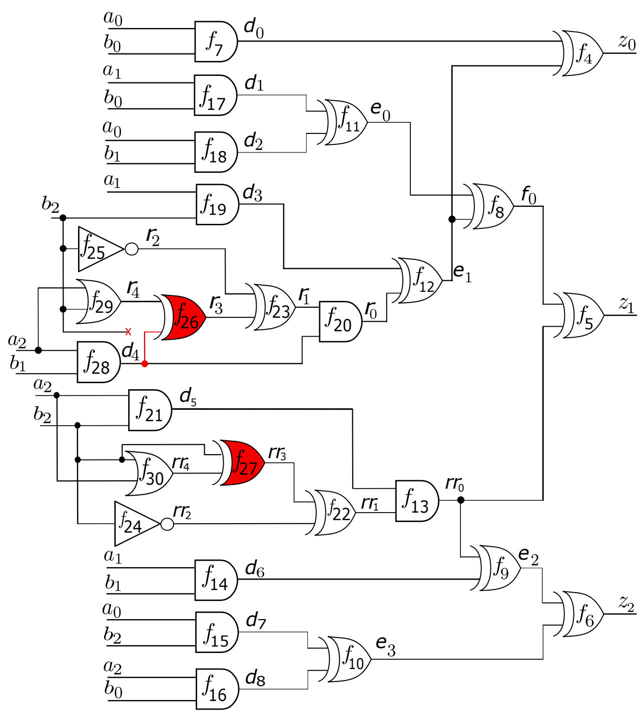

Fig. 1. A faulty *Impl* of the circuit C: a 3-bit finite field multiplier (n=3) with bugs introduced at net r_3 (AND gate replaced with an XOR gate and one of the inputs mis-connected to d_4 instead of b_2) and net rr_3 (AND gate replaced with an XOR gate).

4 RECTIFICATION CHECK

In [26], the authors presented techniques which utilize the aforementioned polynomial ideal setup to derive the necessary and sufficient conditions for the existence of a multi-fix rectification at the given set of targets. We rephrase and restate Thm. V.1 from [26], and briefly discuss its key aspects. Subsequently, we formulate the computation of rectification functions by utilizing the outcome of their decision procedure.

THEOREM 4.1. **[Multi-fix Rectification Theorem]** *A Spec polynomial f, a faulty Impl C represented using the ideal $J + J_0 = \langle F \cup F_0 \rangle \subset R$, and a set of targets $W = (w_1, \dots, w_m) \subset \{X - X_{PI}\}$ are given. RTTO $>$ is imposed on R. Let $W_c = \{(0, 0, .., 0), \dots, (1, 1, .., 1)\}$, $|W_c| = 2^m$, denote the set of all possible $\{0, 1\}$ assignments to targets W. This is akin to computing cofactors of the circuit functions with respect to the targets W. Each cofactor tuple $W_c[l]$ serves as one set of assignments to m targets at their respective indexes in W. The following ideals are constructed:*

- $J_l = \langle F_l \rangle = \langle f_1, \dots, f_{w_1} : w_1 + W_c[l][1], \dots, f_{w_m} : w_m + W_c[l][m], \dots, f_s \rangle, \forall l \in 1, \dots 2^m.$

*Reduce f by $F_l \cup F_0$ to obtain remainders rem_l: $f \xrightarrow{F_l \cup F_0}_+ rem_l$, for $1 \le l \le 2^m$. Then, the circuit C is rectifiable at the target set W **if and only if** union of varieties $\bigcup_{l=1}^{2^m} V(rem_l) = \mathbb{F}_2^{|X_{PI}|}$.*

PROOF 4.1. *As the correction at target W makes C match f, f should vanish on $V_{\mathbb{F}_{2^k}}(J')$. Moreover, each rem_l comprises only X_{PI} variables. This is because WRTO $>_R$ ensures that each non primary input* *variable (each gate output and word-level variable) appears as the leading term of some polynomial in F'. Thus each non primary input variable is canceled in the reduction $f \xrightarrow{F'_l, F'_0}_+ rem_l$. Furthermore, as X_{PI} take values in $\mathbb{F}_2$, $x^2 = x, \forall x \in X_{PI}$. Hence, $V(rem_l) \subseteq \mathbb{F}_2^{|X_{PI}|}$. Thus, the rectification theorem can be equivalently stated as: "f vanishes on $V_{\mathbb{F}_{2^k}}(J') \iff \bigcup_{l=1}^{2^m} V(rem_l) = \mathbb{F}_2^{|X_{PI}|}$".*

(i) **To prove "$\Rightarrow$"**: *Let $x_{PI} \in \mathbb{F}_2^{|X_{PI}|}$ be an assignment to the primary input variables of C. Every assignment x_{PI} results in a corresponding assignment x_{int} to rest of the variables in C. For each such point $(x_{PI}, x_{int}) \in \mathbb{F}_{2^k}$, the target W evaluates to one of the values in the list δ, i.e. $(0, 1, \beta, \ldots, \beta^{2^m-2})$. When $W = 0$, J'_1 vanishes on the point (x_{PI}, x_{int}). Likewise, J'_2 vanishes on (x_{PI}, x_{int}) when $W = 1$, and so on. Since $f \xrightarrow{F'_l \cup F'_0}_+ rem_l, 1 \le l \le 2^m$, and f vanishes on the point (x_{PI}, x_{int}) to begin with, we obtain that for every primary input assignment x_{PI}, one of the rem_l vanishes. This implies that $\bigcup_{l=1}^{2^m} V(rem_l) = \mathbb{F}_2^{|X_{PI}|}$.*

(ii) **To prove "$\Leftarrow$"**: *Say there exists an assignment to the primary inputs $x_{PI} \in \mathbb{F}_2^{|X_{PI}|}$ such that rem_1 vanishes on x_{PI}, i.e. $rem_1(x_{PI}) = 0$. For the given point x_{PI}, the rest of the variables of C get a corresponding assignment x_{int}. As $f \xrightarrow{F'_1 \cup F'_0}_+ rem_1$, we have that f is a member of the ideal $J'_1 + J'_0 + \langle rem_1 \rangle$. Therefore, when $rem_1(x_{PI}) = 0$, the ideal J'_1 also vanishes on $(x_{PI}, x_{int}) \in \mathbb{F}_{2^k}$ because the tuple (x_{PI}, x_{int}) is a valid evaluation of the circuit. Further, J'_0 by definition vanishes everywhere in R'. This implies that $f(x_{PI}, x_{int}) = 0$. The argument similarly holds for each rem_l vanishing on some x_{PI}. This proves that for all primary inputs, if any $rem_l : 1 \le l \le 2^m$ vanishes, then f vanishes too; and that completes the proof.*

The above check for union of varieties can be performed as product of ideals, i.e. by checking if $\prod_{l=1}^{2^m} rem_l \xrightarrow{J_0}_+ 0$. RTTO > is known to have the property that makes the division $f \xrightarrow{F_l \cup F_0}_+ rem_l$ mimic gate level substitution in polynomial algebra. Thus, after reduction, all non-primary input variables in the circuit are canceled and the final remainder has only X_{PI} variables in its support.

Example 4.1. *Continuing on with the Ex. 3.1, we demonstrate the rectification check presented in [26] for $W = (r_3, rr_3)$.*

Constructing the J_l ideals:

- $J_1 = \langle F_1 \rangle$, *where* $F_1[f_{26} : r_3 + 0], F_1[f_{27} : rr_3 + 0], (r_3 = 0, rr_3 = 0)$
- $J_2 = \langle F_2 \rangle$, *where* $F_2[f_{26} : r_3 + 0], F_2[f_{27} : rr_3 + 1], (r_3 = 0, rr_3 = 1)$
- $J_3 = \langle F_3 \rangle$, *where* $F_3[f_{26} : r_3 + 1], F_3[f_{27} : rr_3 + 0], (r_3 = 1, rr_3 = 0)$
- $J_4 = \langle F_4 \rangle$, *where* $F_4[f_{26} : r_3 + 1], F_4[f_{27} : rr_3 + 1], (r_3 = 1, rr_3 = 1)$

Reducing the Spec $f : Z + A \cdot B$ modulo these ideals, we get:

- $rem_1 = f \xrightarrow{F_1 \cup F_0}_+ (\gamma + 1)a_2b_1b_2 + (\gamma^2 + \gamma)a_2b_2$
- $rem_2 = f \xrightarrow{F_2 \cup F_0}_+ (\gamma + 1)a_2b_1b_2$
- $rem_3 = f \xrightarrow{F_3 \cup F_0}_+ (\gamma + 1)a_2b_1b_2 + a_2b_1 + (\gamma^2 + \gamma)a_2b_2$
- $rem_4 = f \xrightarrow{F_4 \cup F_0}_+ (\gamma + 1)a_2b_1b_2 + a_2b_1$

When we compute $\prod_{l=1}^{2^m} rem_l \xrightarrow{J_0}_+$, we obtain remainder 0, thus confirming that the target set $W = (r_3, rr_3)$ indeed admits correction.

The concepts presented in [26] are limited to proving the *existence* of rectification functions. However, our investigation further reveals that their result can be extended to characterize the desired rectification functions. Intuitively, the concept can be elaborated as follows. The variety of rem_l for any l corresponds to the set of all assignments to primary inputs X_{PI} (minterms) where the *Spec* f agrees with the *Impl* C. Thus, the condition of Thm. 4.1 implies that the union of individual varieties of rem_l's comprises the set of all minterms where f and C evaluate the same. Thus, for every primary input assignment, *there exists* a cofactor tuple assignment $W_c[l]$ to W such that f and C match. Consequently, there exists a set of functions $U = (u_1, \ldots, u_m)$ that can be computed to rectify every error minterm. We exploit and explore this concept to compute rectification functions in the following section.

5 COMPUTING RECTIFICATION FUNCTIONS

For a given set of targets W, due to the presence of don't cares (DC), there may exist more than one set U of rectification functions which will rectify the circuit. In this section, we describe the notion of DC in the MFR setup which can be exploited for the simplification of rectification patches. Exploring all the DC conditions for m targets might be computationally infeasible; we present two different approaches to overcome this. First, we present an approach to compute an on- and off-set for each rectification function by heuristically resolving all the DC conditions. Following this, we present an approach to explore and compute a subset of the DC conditions, along with on- and off-sets, for each rectification function.

5.1 Greedy Approach for MFR

To illustrate the greedy approach, consider the case with $m = 2$, where $W_c = \{(0,0), (0,1), (1,0), (1,1)\}$, and we must compute rectification functions u_1 and u_2. For brevity, let $V_{W_c[i]} = V(rem_i)$, for $1 \leq i \leq 2^m$; in this case, $V_{(0,0)} = V(rem_1)$, $V_{(0,1)} = V(rem_2)$, and so on.

Recall that $V_{(0,0)}$ comprises the set of points where the *Impl* and *Spec* evaluate the same for the corresponding assignments $(0,0)$ to the targets. This implies that at these points, the rectification functions u_1 and u_2 should evaluate to 0. Table 2 shows the required evaluations of u_1 and u_2 for the points in each variety, following the same reasoning. The on-set of the rectification function for a target corresponds to the union of the varieties (sets) where the function evaluates to 1, and the off-set corresponds to the union of the varieties (sets) where the function evaluates to 0. In this case, the on-set of u_1 consists of the set of points in $V_{(1,0)} \cup V_{(1,1)}$, and the off-set of u_1 consists of the set of points in $V_{(0,0)} \cup V_{(0,1)}$. Similarly, the on-set and off-set of u_2 consists of the points in $V_{(0,1)} \cup V_{(1,1)}$ and $V_{(0,0)} \cup V_{(1,0)}$, respectively. The functions u_1 and u_2 could be synthesized using these on- and off-sets.

Table 2. Function evaluations

$Variety$	u_1	u_2
$V_{(0,0)}$	0	0
$V_{(0,1)}$	0	1
$V_{(1,0)}$	1	0
$V_{(1,1)}$	1	1

However, the above argument is only correct when $V_{(0,0)}, V_{(0,1)}, V_{(1,0)}, V_{(1,1)}$ are pairwise disjoint, which may not be true in practice. For example, for a point contained in $V_{(0,0)} \cap V_{(0,1)}$, (u_1, u_2) must evaluate either to $(0,0)$, or to $(0,1)$ in order for the *Impl* to evaluate to the same value as the *Spec*; this point would be in both the on- and off-set of u_2 in the method previously described. A decision procedure is necessary to determine the evaluation of (u_1, u_2) at these intersections. We present a greedy approach which resolves such ambiguities by imposing an order on the sets. An example of our greedy approach to evaluate (u_1, u_2) for an order $V_{(0,0)} > V_{(0,1)} > V_{(1,0)} > V_{(1,1)}$ is as follows:

First, we place all the points from $V_{(0,0)}$ into the off-sets of (u_1, u_2). Next, we place all the points from $V_{(0,1)} \setminus V_{(0,0)}$ into the off-set of u_1 and the on-set of u_2. We perform the set difference to avoid placing the points in $V_{(0,0)} \cap V_{(0,1)}$ into both the on-set and off-set of u_2. Next, we place all the points from $V_{(1,0)} \setminus (V_{(0,0)} \cup V_{(0,1)})$ into the on-set of u_1, and the off-set of u_2. Finally, we place the remaining points from $V_{(1,1)} \setminus (V_{(0,0)} \cup V_{(0,1)} \cup V_{(1,0)})$ into the on-set of (u_1, u_2). The resulting on- and off-sets for u_1 and u_2 are shown below.

$$
\begin{aligned}
V(u_{1_{on}}) &= (V_{(1,1)} \setminus (V_{(0,0)} \cup V_{(0,1)} \cup V_{(1,0)})) \cup (V_{(1,0)} \setminus (V_{(0,0)} \cup V_{(0,1)})) \\
V(u_{1_{off}}) &= (V_{(0,0)}) \cup (V_{(0,1)} \setminus V_{(0,0)}) \\
V(u_{2_{on}}) &= (V_{(0,1)} \setminus V_{(0,0)}) \cup (V_{(1,1)} \setminus (V_{(0,0)} \cup V_{(0,1)} \cup V_{(1,0)})) \\
V(u_{2_{off}}) &= (V_{(0,0)}) \cup (V_{(1,0)} \setminus (V_{(0,0)} \cup V_{(0,1)}))
\end{aligned}
$$

This approach with the given order greedily places points into the off-sets of the rectification functions (u_1, u_2) where possible, and only places points into the on-sets of the rectification functions when necessary. Subject to the given order, the on-sets of the rectification functions are thus minimized. For the experiments in this paper, we always use the order $V_{W_c[i]} > V_{W_c[j]}$ for $i < j$, as in the above example, though any order would yield valid rectification functions.

Generalizing our greedy approach for m targets, we first construct the following composite sets (varieties):

$$
S_l = \begin{cases} V_{W_c[1]}, & \text{if } l = 1 \\ V_{W_c[l]} \setminus (\bigcup_{j=1}^{l-1} V_{W_c[j]}), & 2 \leq l \leq 2^m \end{cases} \tag{4}
$$

The resulting on-set and off-set functions for each target i, where $1 \leq i \leq m$ are:

$$
\begin{aligned}
V(u_{i_{on}}) &= \bigcup S_l, \ \forall l \mid W_c[l][i] = 1 \\
V(u_{i_{off}}) &= \bigcup S_l, \ \forall l \mid W_c[l][i] = 0
\end{aligned} \tag{5}
$$

5.2 Don't Care Conditions for MFR

Let $U_d \subseteq U$ denote a subset of the target rectification functions. We are interested in the DC conditions which arise for these functions at points where they may evaluate to any value, for some fixed evaluation of the remaining functions in the set $\{U \setminus U_d\}$. For example, consider a point in $V_{(0,0)} \cap V_{(0,1)}$ for a circuit with two targets. As discussed previously, u_1 must evaluate to 0 at this point, but $U_d = \{u_2\}$ may evaluate either to 0 or to 1, so this is a *DC* point for u_2.

Not every intersection of varieties yields *DC* points which follow the conditions described above. Consider a point in $V_{(0,0)} \cap V_{(1,1)}$. Here, (u_1, u_2) must evaluate either to $(0,0)$ or to $(1,1)$. If this point were assigned to the *DC* set of u_2, for example, the *Spec* and *Impl* would only evaluate the same if u_1 evaluated to the same value as u_2. Thus, u_1 would become a function of u_2 at this point. This point cannot be placed into the on-set, off-set, or DC-set of u_1 before u_2 is evaluated. To avoid inter-dependencies between the rectification functions, we do not classify points in such intersections as *DC* points. We rely on our greedy heuristic to evaluate these points.

Finally, consider a point in $V_{(0,0)} \cap V_{(0,1)} \cap V_{(1,0)}$. This point cannot be a DC point for both targets simultaneously since the evaluation $(1, 1)$ here will result in an incorrect rectification function. However, because $V_{(0,0)} \cap V_{(0,1)} \cap V_{(1,0)} \subset V_{(0,0)} \cap V_{(0,1)}$, we could treat this point as a *DC* point for u_2 and evaluate u_1 to 0. Alternatively, because $V_{(0,0)} \cap V_{(0,1)} \cap V_{(1,0)} \subset V_{(0,0)} \cap V_{(1,0)}$, we could treat this point as a *DC* point for u_1 and evaluate u_2 to 0. Thus, we have a choice to place this point in the *DC*-set of either targets, but not both.

Finding every intersection containing *DC* points for every target can be very expensive for circuits with more than a few targets. We therefore propose an approach to compute a subset of the *DC* points by considering only the set of pairwise intersections of varieties which contain *DC* points for exactly one target, denoted as DC_{pair}.

Let $d(W_c[j], W_c[k])$ denote the Hamming distance between the cofactor tuples $W_c[j]$ and $W_c[k]$. We compute the set of varieties which contain *DC* points for one target, denoted DC_{pair}, from the equation below, where $1 \leq j, k \leq 2^m$.

$$DC_{pair} = \{V_{W_c[j]} \cap V_{W_c[k]} \mid d(W_c[j], W_c[k]) = 1\} \tag{6}$$

Since the Hamming distance $d = 1$ between the cofactor tuples $W_c[j]$ and $W_c[k]$ for each intersection of varieties in DC_{pair}, exactly one rectification function may evaluate either to 0 or to 1. The remaining rectification functions require fixed evaluations of 1 or 0. Therefore, each intersection of varieties in DC_{pair} yields *DC* points for exactly one rectification function in U, and either on- or off-set points for the remaining rectification functions in U. We use DC_{pair} to compute the *DC* points for each rectification function, as described in the next section.

5.2.1 Computing Rectification Functions with Don't Cares. Once the set DC_{pair} has been found, a few steps remain to compute the on-, off-, and don't-care sets for each target. First, we follow an approach identical to the greedy approach to evaluate points not located inside DC_{pair}. We construct new composite sets S_l^d for $1 \leq l \leq 2^m$, which are identical to the composite sets (varieties) created for the previous approach, except that all the points from DC_{pair} set are removed.

$$S_l^d = \begin{cases} V_{W_c[1]} \setminus DC_{pair}, & \text{if } l = 1 \\ V_{W_c[l]} \setminus ((\bigcup_{j=1}^{l-1} V_{W_c[j]}) \cup DC_{pair}), & 2 \leq l \leq 2^m \end{cases} \tag{7}$$

Points in these composite sets are assigned to the on- and off-set for each rectification function in the same way as Eqn. (5), substituting S_l with S_l^d.

Next, we must evaluate points within DC_{pair} as on-, off-, or *DC* points for each rectification function. We select the first pairwise intersection of varieties in DC_{pair} and assign the points according to the cofactor tuples, as described previously. Since the intersections within DC_{pair} may not be disjoint, we then take the next pairwise intersection, remove the points from the first intersection which have already been assigned, then assign these points according to the cofactor tuples. We continue this for each subsequent intersection of varieties from DC_{pair}, remembering to remove all previously assigned points at each step.

For example, for a circuit with two targets, $DC_{pair} = \{V_{(0,0)} \cap V_{(0,1)}, V_{(0,0)} \cap V_{(1,0)}, V_{(0,1)} \cap V_{(1,1)}, V_{(1,0)} \cap V_{(1,1)}\}$. We place the points in $V_{(0,0)} \cap V_{(0,1)}$ into the off-set of u_1 and the *DC* set of u_2. We then place the points in $V_{(0,0)} \cap V_{(1,0)} \setminus V_{(0,0)} \cap V_{(0,1)}$ into the *DC* set of u_1 and the off-set of u_2. We place points in $V_{(0,1)} \cap V_{(1,1)} \setminus ((V_{(0,0)} \cap V_{(0,1)}) \cup (V_{(0,0)} \cap V_{(1,0)}))$ into the *DC* set of u_1 and the on-set of u_2. Finally, we place points in $V_{(1,0)} \cap V_{(1,1)} \setminus ((V_{(0,0)} \cap V_{(0,1)}) \cup (V_{(0,0)} \cap V_{(1,0)}) \cup (V_{(0,1)} \cap V_{(1,1)}))$ into the on-set of u_1 and the *DC* set of u_2. Following this approach, we calculate on- off- and *DC* sets for each rectification function.

5.3 Synthesizing Rectification Functions

The above techniques show how to construct a rectification function by reasoning about the varieties of rem_l. However, algebraically, we compute these functions using their corresponding ideals. Specifically, we show how the remainders computed in Thm. 4.1 can be utilized for rectification function computation. Even though the remainders rem_l have coefficients in $\mathbb{F}_{2^n}$ (higher field), their varieties are in $\mathbb{F}_2^{|X_{PI}|}$ as they correspond to bit-level assignments to X_{PI}. However, in [13], it was shown that it is a property of such ideals ($\langle rem_l, J_0\rangle$) that their reduced Gröbner bases (Def. 3.2) have coefficients only in $\mathbb{F}_2$. Further, it was shown that, given an ideal I with coefficients in $\mathbb{F}_2$ with generators $\{g_1, \dots, g_t\}$, a polynomial p can always be constructed as $p = (1+g_1)(1+g_2)\dots(1+g_t)+1$, such that $V(p) = V(I)$. Consequently, the rectification function operations are restricted to algebraic computations in $\mathbb{F}_2[X_{PI}] \equiv \mathbb{B}$.

To compute the patch u_i, we perform the following steps:

- Compute reduced Gröbner bases of $\langle rem_l, J_0\rangle$.
- Construct a singleton polynomial p such that $V(p) = V(\langle rem_l, J_0\rangle)$.
- Impose an order on the sets for DC_{pair} and composite set computations.
- Compute DC_{pair} using Eqn. (6), and then obtain the composite sets in Eqn. (7) which are then assigned to DC-, on- and off-sets of the rectification functions (Sec. 5.2).
- In order to perform the variety union, intersection, and difference operations, we use ideal product, sum, and colon operations, respectively, on the singleton polynomial p representation of the ideal.
- The above procedure delivers $u_{i_{DC}}$ and $u_{i_{on}}$ as singleton polynomials in $\mathbb{F}_2[X_{PI}]$. Translate $u_{i_{DC}}$ and $u_{i_{on}}$ into Boolean functions by interpreting the product, sum, and '+1' as Boolean AND, XOR, and INV gates, respectively. Optimize the on-set $u_{i_{on}}$ w.r.t. to the DC-set $u_{i_{DC}}$.

Example 5.1. *Continuing with Ex. 4.1:*

- $rem_3 = (\gamma + 1)a_2b_1b_2 + a_2b_1 + (\gamma^2 + \gamma)a_2b_2$
- $redGB(\langle rem_3, J_0\rangle) = \{a_2b_1, a_2b_2\}$
- $p_{rem_3} = (1 + a_2b_1) * (1 + a_2b_2) + 1 = a_2b_1b_2 + a_2b_1 + a_2b_2$
  - *Here,* $V(p_{rem_3}) = V(\langle rem_3, J_0\rangle)$
- *Similarly, polynomials* p_{rem_l} *for each* rem_l *can be computed.*

The rectification polynomials for the targets (r_3, rr_3) *computed using the greedy approach (Sec. 5.1)*

$$u_{1_{on}} = a_2b_1b_2; \quad u_{1_{off}} = a_2b_1b_2 + 1; \quad r_3 = u_{1_{patch}} = (a_2 \wedge b_1 \wedge b_2);$$
$$u_{2_{on}} = a_2b_2; \quad u_{2_{off}} = a_2b_2 + 1; \quad rr_3 = u_{2_{patch}} = (a_2 \wedge b_2).$$

The rectification polynomials for the targets (r_3, rr_3) *computed using the on-set and don't care simplification (Sec. 5.2.1)*

$$u_{1_{dc}} = a_2b_1b_2 + a_2b_2; \quad u_{1_{on}} = a_2b_1b_2; \quad r_3 = u_{1_{patch}} = a_2 \wedge b_2;$$
$$u_{2_{dc}} = a_2b_2 + 1; \quad u_{2_{on}} = a_2b_2; \quad rr_3 = u_{2_{patch}} = 1.$$

The techniques described in rectification check and function computation predominantly involve Gröbner basis computations and polynomial reductions. By virtue of RTTO >, the complexity of remainder (rem_l) generation in the rectification check was moved from one of computing GB to that of GB-reduction by way of multivariate polynomial division. However, the function computation operation also involves computing a reduced GB, which can be especially prohibitive when applied to circuits with large size operands. Hence, there is a need for efficient representation and to overcome the infeasibility of these operations using conventional computer algebra tools.

to overcome this complexity. As the GB algorithms are employed to compute rectification function patches via CI on the circuit, we analyze the circuit and its topology to derive specific information which we use to efficiently guide the computation of patches. Moreover, we further show how this information can be effectively utilized by our ZDD based algorithms to compute rectification patches. By combining our theories and algorithms we can explore the space of all possible rectification functions through their ON-set, OFF-set and DC-set of corresponding Boolean functions.

6 IMPROVING RECTIFICATION SETUP EFFICIENCY USING ZDDS

Taking inspiration from [15], we utilize PolyBoriâĂŹs [4] reduction procedure with ZDDs [22, 23] as the underlying data structure to improve our proposed approach. PolyBori proposed the use of ZDDs to compute Gröbner bases for Boolean polynomials. PolyBori is a generic Boolean GB computational engine that caters to many permissible term orders. Its division algorithm is also generally based on the conventional concept of canceling one monomial in every step of reduction. In contrast, our algorithms are tailored for GB-reduction under the RTTO $>$. The efficiency of our approach stems from the observation that the RTTO $>$ imposes a special structure on the ZDDs, which allows for multiple monomials to be canceled in one division-step, along with simplifying the search for divisors. The efficiency is derived by treating the polynomials as unate cube sets and checking isomorphism between the implementation and specification graphs. We reason about the presence or absence of solutions and other properties of a system of polynomials without explicitly solving them. The Boolean values of the nets of a circuit for all possible input assignments is a set of points, which can be construed as solutions to a set of polynomials. We further improve upon the rectification approach by exploiting the circuit topology and ZDD based GB reductions.

7 EXPERIMENTAL RESULTS

The benchmark suit includes two modular multipliers (Mastrovito and Montgomery), and a circuit that performs Point Addition over NIST standard Elliptic curves. These benchmarks are taken from [19] and synthesized using the *abc* tool with a gate library comprising two input gates. Table 3 presents the results of our approach when performing MFR against their respective polynomial specifications. We introduce bugs by means of gate and wiring modifications in the synthesized netlists such that multiple output bits are affected in the design (column #BO). We introduce multiple such modifications at various topological levels: some closer to PIs, some in the middle of the circuit, and some near POs. In our experiments, the number of targets is chosen from $m = \{2, 3, 5\}$. Our approach isn't limited by this set m and can perform MFR for any given number of targets.

Our approach is implemented as a custom software in Python programming language. We use PolyBori's [4] ZDD based API to implement the division, $f \xrightarrow{F'_l \cup F'_0}_+ rem_l$, $1 \leq l \leq 2^m$. Subsequently, the remainders generated from these divisions are utilized in the decision procedure, as well as the function computations. Further, we use *sis* [29] and *abc* [3] to perform logic optimization and synthesis. Specifically, in *sis* we run a script to perform *kernel extraction* and *full simplify* to optimize the rectification functions computed in Sec.5.2. We use *abc* to perform *structural hashing, balancing, refactoring, rewriting, etc.*. Finally, we *map* the computed functions using a library of AND-XOR-INV gates and extract the synthesis results for *area* and *delay*. The experiments are performed on a 3.5GHz Intel(R) Core$^{\text{TM}}$ i7-4770K Quad-Core CPU with 32 GB RAM.

Table 3 presents the characteristics of the benchmark suit and the execution time for the computations using our approach. Column PBS denotes the time taken to build the respective ZDD models (commensurates with the operand word-length n). Execution time for rectification check (RC) and function computation (GFC and DFC) depend on various factors such as: i) the number of

Table 3. Time is in seconds; I = Benchmark Index, n = Datapath Size, m = target word size, AM = Maximum resident memory utilization in Mega Bytes (Average across benchmarks), #G = Number of gates $\times 10^3$, , #BO = Number of faulty outputs, PBS = Required time for PolyBori setup (ring declaration/poly collection/spec collection), RC = Required time for verification, multi-fix setup and rectifiability check, GFC = Required time for function computation using the greedy approach, DFC = Required time for function computation with don't cares

				Mastrovito						Montgomery						Point Addition					
I	***n***	***m***	**AM**	**#G**	**#BO**	**PBS**	**RC**	**GFC**	**DFC**	**#G**	**#BO**	**PBS**	**RC**	**GFC**	**DFC**	**#G**	**#BO**	**PBS**	**RC**	**GFC**	**DFC**
1	**16**	5	100	0.8	6	0.05	0.01	1.7	4.7	0.9	16	0.05	0.44	190	277*	0.9	7	0.07	0.02	6.6	20
2	**32**	5	120	2.8	8	0.14	0.02	2	8	2.8	32	0.16	0.1	114	180*	2.9	13	0.2	0.06	36	85*
3	**64**	3	160	11.2	5	0.61	3.59	5	15	9.6	47	0.53	0.14	7	14	10.6	64	0.8	0.18	10	35
4	**96**	2	240	24.5	5	1.45	0.12	0.4	0.7	21	96	1.35	3.14	87	111*	24.8	96	2.53	0.5	7.7	90
5	**128**	2	370	43.2	5	3.23	0.24	0.8	1.2	35.8	128	2.92	2.06	28	393*	43.2	128	6.42	4.38	49	62*
6	**163**	5	550	69.8	6	6.21	0.4	4.4	7.7	57.5	128	5.38	1.53	131	220*	71.6	22	15.9	1.53	3.2	8
7	**233**	2	750	119	3	12.7	0.66	0.3	1.7	112	233	12.6	2.69	6	40	122	233	19.7	1.44	6.4	32
8	**409**	2	2400	384	2	190	2.22	0.3	9	340	409	136	6.15	2.7	5.2	368	409	224	5.3	7.7	8*
9	**571**	2	5000	827	5	2143	5.96	0.3	26	663	427	1386	49	1.6	30	813	5	2492	13.2	1.2	20

bugs; ii) the number of targets; iii) location of the bugs; iv) location of the targets; v) the number of affected outputs; and vi) size of the patch function being computed in terms of number of gates. Collectively, these factors decide the size of the remainder rem_l, and the number of remainders l. We omit the comparison with the contemporary approaches as they fail (timeout = 3 hrs) to rectify circuits beyond 16-bits, which is the smallest benchmark from our results table.

Table 4 presents the synthesis results post *abc* mapping for GFC and DFC approaches in terms of area (number of gates) and the longest topological delay. The asterisk (*) in the DFC columns denotes the cases where *full simplify* (*sis*) fails to utilize the pairwise intersection don't care network for the on-set function simplification. In these cases, *sis* aborts simplification as the BDD size exceeds 480,000 nodes. For these entries, the patch functions are synthesized using *abc* which ignores the don't care network. The time taken for GFC is less than the time taken for DFC computations. However, synthesis results computed using DFC where *sis* completed simplification successfully are of better quality than the one computed using GFC for most of the cases.

8 CONCLUSION

This paper presents an automated symbolic computer algebra approach to perform MFR of faulty finite field arithmetic circuits at a given set of targets. Our approach reasons about the rectification functions by means of algebraic varieties in finite fields, and computes these functions using Gröbner bases of ideals corresponding to the circuit. We present two MFR approaches, a heuristic which greedily tries to resolve the rectification functions for the targets, and a variety intersection heuristic that explores a subset of don't cares condition for the target functions. Our approach is able to compute rectification functions for circuits with large (NIST-standard) operand widths n. As part of future work, we are working on function computation in terms of internal nets. Further, we are also investigating the extension of this approach to integer arithmetic circuits.

REFERENCES

[1] W. W. Adams and P. Loustaunau. 1994. *An Introduction to Gröbner Bases.* American Mathematical Society.

Table 4. Synthesis results for mapped patch network; I = Benchmark Index, GFC = Greedy function computation, DFC-on = Function computation with don't care optimization of on-set, DFC-off = Function computation with don't care optimization of complement of off-set, A = Area in terms of number of gates , D = Longest delay

	Mastrovito						Montgomery						Point Addition					
	GFC		DFC-on		DFC-off		GFC		DFC-on		DFC-off		GFC		DFC-on		DFC-off	
I	A	D	A	D	A	D	A	D	A	D	A	D	A	D	A	D	A	D
1	19	3	17	3			27788	50	27941*	62*			761	40	265	12		
2	34	5	35	4			19340	65	19384*	57*			8882	69	9063*	60*		
3	1675	29	1577	46			1511	30	560	18			3040	32	3733	41		
4	86	11	21	5			55085	50	55568*	52*			6642	89	6193	70		
5	283	21	103	12			25819	44	26744*	35*			27544	36	27289*	35*		
6	222	17	99	7			27035	68	27409*	61*			66	8	39	7		
7	9	4	9	4			8094	25	4948	28			4345	30	5169	34		
8	16	4	11	4			844	13	4	2			2707	24	2611*	23*		
9	21	7	18	6			299	19	287	19			622	22	210	16		

[2] E. Biham, Y. Carmeli, and A. Shamir. 2008. Bug Attacks. In *Proceedings on Advances in Cryptology*. 221–240.

[3] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-strength Verification Tool. In *Computer Aided Verification (CAV)*. Springer, 24–40.

[4] M. Brickenstein and A. Dreyer. 2009. PolyBoRi: A framework for GrÃűbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation* 44 (2009), 1326–1345.

[5] B. Buchberger. 1965. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* Ph.D. Dissertation. University of Innsbruck.

[6] Ai Quoc Dao, Nian-Ze Lee, Li-Cheng Chen, Mark Po-Hung Lin, Jie-Hong R. Jiang, Alan Mishchenko, and Robert Brayton. 2018. Efficient Computation of ECO Patch Functions. In *Design Automation Conference (DAC)*. ACM, 51:1–51:6.

[7] F. Farahmandi and P. Mishra. 2016. Automated Test Generation for Debugging Arithmetic Circuits. In *Design, Automation Test in Europe Conference Exhibition (DATE)*.

[8] F. Farahmandi and P. Mishra. 2017. Automated Debugging of Arithmetic Circuits Using Incremental Gröbner Basis Reduction. In *International Conference on Computer Design (ICCD)*. 193–200.

[9] M. Fujita. 2015. Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design. *Proc. IEEE* (2015).

[10] S. Gao, A. Platzer, and E. Clarke. 2011. Quantifier Elimination over Finite Fields with Gröbner Bases. In *Intl. Conf. Algebraic Informatics*.

[11] A. M. Gharehbaghi and M. Fujita. 2017. A new approach for selecting inputs of logic functions during debug. In *International Symposium on Quality Electronic Design (ISQED)*. 166–173.

[12] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker. 2013. Equivalence Checking of Partial Designs Using Dependency Quantified Boolean Formulae. In *IEEE International Conference on Computer Design (ICCD)*. 396–403.

[13] U. Gupta, I. Ilioaea, V. Rao, A. Srinath, P. Kalla, and F. Enescu. 2018. On the Rectifiability of Arithmetic Circuits using Craig Interpolants in Finite Fields. In *International Conference on Very Large Scale Integration (VLSI-Soc)*. 49–54.

[14] U. Gupta, P. Kalla, I. Ilioaea, and F. Enescu. 2019. Exploring Algebraic Interpolants for Rectification of Finite Field Arithmetic Circuits with Groebner Bases. In *24th IEEE European Test Symposium (ETS)*. 1–6.

[15] Utkarsh Gupta, Priyank Kalla, and Vikas Rao. 2019. Boolean GrÃűbner Basis Reductions on Finite Field Datapath Circuits Using the Unate Cube Set Algebra. *IEEE Trans. CAD (TCAD)* (2019).

[16] Y. Kimura, A. M. Gharehbaghi, and M. Fujita. 2019. Signal Selection Methods for Efficient Multi-Target Correction. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5.

[17] V. N. Kravets, N. Lee, and J. R. Jiang. 2019. Comprehensive Search for ECO Rectification Using Symbolic Sampling. In *Design Automation Conference (DAC)*. 1–6.

[18] C. C. Lin, K. C. Chen, S. C. Chang, and M. Marek-Sadowska. 1995. Logic Synthesis for Engineering Change. In *Proc. Design Automation Conf. (DAC)*. 647–652.

[19] J. Lv, P. Kalla, and F. Enescu. 2013. Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits. In *IEEE Trans. on CAD (TCAD)*. 1409–1420.

[20] K. Magons. 2016. Applications and Benefits of Elliptic Curve Cryptography. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*.

[21] Alireza Mahzoon, Daniel Große, and Rolf Drechsler. 2018. Combining Symbolic Computer Algebra and Boolean Satisfiability for Automatic Debugging and Fixing of Complex Multipliers. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 351–356.

[22] Shin-ichi Minato. 1993. Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Design Automation Conference (DAC)*. 272–277.

[23] Shin-ichi Minato. 1994. Calculation of Unate Cube Set Algebra Using Zero-suppressed BDDs. In *Design Automation Conference (DAC)*. 420–424.

[24] Vikas Rao. 2018. http://eng.utah.edu/~utkarshg/cex.pdf,. (2018).

[25] V. Rao, U. Gupta, A. Srinath, I. Ilioaea, P. Kalla, and F. Enescu. 2018. Post-Verification Debugging and Rectification of Finite Field Arithmetic Circuits using Computer Algebra Techniques. In *Formal Methods in Computer-Aided Design (FMCAD)*. 1–9.

[26] V. Rao, I. Ilioaea, H. Ondricek, P. Kalla, and F. Enescu. 2021. Word-Level Multi-Fix Rectifiability of Finite Field Arithmetic Circuits. In *International Symposium on Quality Electronic Design (ISQED)*. To Appear.

[27] V. Rao, H. Ondricek, P. Kalla, and F. Enescu. 2021. Computing Rectification Functions for Finite Field Arithmetic Circuits using Computer Algebra Techniques. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. To Appear.

[28] C. Scholl and B. Becker. 2001. Checking Equivalence for Partial Implementations. In *Design Automation Conference (DAC)*.

[29] E. Sentovich *et al.* 1992. *SIS: A System for Sequential Circuit Synthesis*. Technical Report UCB/ERL M92/41. ERL, Dept. of EECS, Univ. of California, Berkeley.

[30] A. Smith, A. Veneris, M. F. Ali, and A. Viglas. 2005. Fault Diagnosis and Logic Debugging using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2005).

[31] K. F. Tang, P. K. Huang, C. N. Chou, and C. Y. Huang. 2012. Multi-patch Generation for Multi-error Logic Rectification by Interpolation with Cofactor Reduction. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 1567–1572.

[32] P. Tuyls and L. Batina. 2006. RFID-Tags for Anti-counterfeiting. In *Topics in Cryptology – CT-RSA 2006*. 115–131.

[33] B. H. Wu, C. J. Yang, C. Y. Huang, and J. H. R. Jiang. 2010. A Robust Functional ECO Engine by SAT Proof Minimization and Interpolation Techniques. In *International Conference on Computer-Aided Design (ICCAD)*. 729–734.

[34] H. T. Zhang and J. H. R. Jiang. 2018. Cost-Aware Patch Generation for Multi-Target Function Rectification of Engineering Change Orders. In *Design Automation Conference (DAC)*. 1–6.