Algebraic Techniques for Rectification of Finite Field Circuits

Vikas Rao[1], Haden Ondricek[1], Priyank Kalla[1], and Florian Enescu[2]
[1]Electrical & Computer Engineering, University of Utah
[2]Mathematics & Statistics, Georgia State University

***Abstract*—This paper addresses the rectification of faulty finite field arithmetic circuits by computing patch functions at internal nets using techniques from polynomial algebra. Contemporary approaches that utilize SAT solving and Craig interpolation are infeasible in rectifying arithmetic circuits. Given candidate nets, prior algebra-based techniques can ascertain whether the circuit admits multi-fix rectification at these nets but cannot compute patch functions. We show how the algebraic computing model facilitates the exploration of admissible rectification functions, collectively, for the nets. This model also enables the exploitation of don't care conditions for the synthesis and realization of the patches. Experimental results on large operand width finite field benchmarks, as used in cryptography, substantiate our approach.**

I. Introduction

Debugging and rectification of digital circuits aims to correct a given defective implementation ($Impl$) to match its intended specification ($Spec$). 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. This paper addresses *the computation of multi-fix rectification (MFR) functions for faulty finite field arithmetic circuits* at a given set of targets. Rectification is performed against a given polynomial $Spec$ over finite fields. Such circuits find application in Elliptic Curve Cryptography.

Problem Statement and Objective: We are given the following: i) as the $Spec$, a multivariate polynomial f with coefficients in a finite field of 2^n elements (denoted $\mathbb{F}_{2^n}$), for a given $n \in \mathbb{Z}_{>0}$; ii) a faulty $Impl$ circuit C, with no assumptions on the number or the type of bugs present in C; and iii) a set $W = (w_1, \ldots, w_m)$ of m targets from C, pre-specified or selected using contemporary signal selection heuristics [1], [2], [3]. We assume it has been ascertained that C admits rectification at these m targets, using [4]. The *objective* of our approach is to: i) compute a set of individual rectification functions $U = (u_1, \ldots, u_m)$ for the corresponding targets. Here, each u_i is a polynomial function $u_i : \mathbb{F}_2^{|X_{PI}|} \rightarrow \mathbb{F}_2$, where $\mathbb{F}_2 = \{0, 1\}$, and X_{PI} denotes the set of primary inputs; ii) derive *don't care (DC) conditions* corresponding to the m rectification functions; and iii) synthesize the rectification polynomials into logic sub-circuit patches.

This research is funded in part by the US National Science Foundation grants CCF-1911007 and CCF-1910368.

Prior Work: Contemporary approaches formulate rectification using QBF solving [5], using Craig interpolation or iterative SAT solving [6]. The MFR techniques in [2], [6], [7] iteratively and incrementally compute multiple single-fix functions that partially patch the circuit in each iteration. Recent techniques include more resource awareness in patch generation by reusing existing logic [1], employ improved heuristics for target selection [2], or resolve a combination of such objectives, such as the symbolic sampling approach of [3]. While successful for control-dominated applications, these techniques are computationally infeasible for rectification of arithmetic circuits.

In the context of arithmetic circuits, Symbolic Computer Algebra (SCA) techniques for integer arithmetic [8], [9] and finite field circuits [10], [11] have been considered for rectification. However, these 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, correction must be attempted at multiple targets, i.e. MFR is required. Recently, [4] proposes an SCA-based approach *that decides m-target rectifiability*. Given a set of m-targets, the approach only ascertains whether *there exists* a set of patch functions at those targets. Since it is only a decision procedure, it cannot *compute* rectification functions.

Approach and Contribution: The $Spec$ f and $Impl$ C are modeled in terms of polynomial ideals. The rectification functions are computed as polynomials whose common zeros correspond to the minterms of the functions. These polynomials are translated to Boolean logic by converting the algebraic multiplication and addition operations to the Boolean AND and XOR operations, respectively. Our approach extends the setup of [4] to *compute* individual rectification functions for each target. We present two approaches: one which employs a greedy heuristic to resolve points in the DC conditions, and one which computes a subset of the DC logic of the rectification functions for patch optimization.

Paper Organization: The following section covers preliminary background. Section II-1 reviews the polynomial modeling concepts. The rectification check formulation is described in Section III, followed by the rectification function and don't care computations in Section IV. Experimental results are described in Section V, and Section VI concludes the paper.

II. Preliminaries

In order to compute rectification functions U at targets W, we first model the *Impl* and *Spec* as multivariate polynomials in the ring $R = \mathbb{F}_{2^n}[x_1, \ldots, x_d]$. Polynomials in R contain variables from $\{x_1, \ldots, x_d\}$ and coefficients from the finite field $\mathbb{F}_{2^n}$. Section II-1 in the sequel describes how to convert a circuit C and its *Spec* into a set of polynomials $F = \{f_1, \ldots, f_s\}$ contained in R.

We are interested in the set of common zeros of the polynomials in F. This set of points is called the **variety**. Boolean functions also comprise a 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 an on-set minterm of a corresponding Boolean function.

The first two columns of Table I describe the correspondence between operations on Boolean functions and operations on varieties. We utilize this correspondence to construct rectification functions by modeling the on-, off-, and DC-sets as varieties. The variety depends not only on the set of polynomials F, but on the **ideal** generated by F. The ideal generated by F is defined as $J = \langle f_1, \ldots, f_s \rangle = \{h_1 \cdot f_1 + \cdots + h_s \cdot f_s \mid h_1, \ldots, h_s \in R\}$. We denote the variety of J as $V(J)$. Algebraic Geometry analyzes ideals to reason about varieties without explicitly computing the varieties, which is infeasible in practice. In the third column, "$\cdot$" , "$+$" , " : " represent product, sum, and colon operations on ideals, respectively; these ideal operations are implemented in computer algebra tools, which we utilize.

TABLE I: Correspondences between algebraic operations and Boolean operations. Here, $J_1 = \langle f \rangle$, and $J_2 = \langle g \rangle$.

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

The relation $\varphi^{2^n} - \varphi = 0$ holds for every element φ in $\mathbb{F}_{2^n}$. Therefore, any polynomial of the form $x^{2^n} - x$ vanishes at every point in $\mathbb{F}_{2^n}$ and all such polynomials are called vanishing polynomials. We denote the set of all vanishing polynomials as $F_0 = \{x_1^{2^n} - x_1, \ldots, x_d^{2^n} - x_d\}$, and the corresponding ideal generated as $J_0 = \langle F_0 \rangle$. Over circuits, the sum of ideals $J + J_0$ helps us to formulate verification and rectification by restricting the variety $V(J + J_0)$ to a finite set of points in the field $\mathbb{F}_{2^n}$.

Another core operation used in our computations is polynomial reduction. Let $F = \{f_1, \ldots, f_s\}$ be a set of polynomials in R and $f \in R$ be another polynomial. Then $f \xrightarrow{F}_+ r$ denotes the *reduction* of f modulo the set of polynomials F, resulting in a remainder r. The terms in f are iteratively cancelled by the leading terms of the polynomials in F using polynomial division (cf. Algorithm 1.5.1 [12]).

An ideal may have many different bases, or generators. A *Gröbner basis* (GB) is a basis with properties useful in solving many polynomial decision and quantification problems. For example, a polynomial f is a member of ideal J if and only if $f \xrightarrow{GB(J)}_+ 0$. When $f \notin J$, division by $GB(J)$ results in a non-zero remainder r that is unique/canonical. A reduced GB is a canonical representation for an ideal.

In order to systematically process polynomials, their monomials (terms) must be ordered. In this work, we use a reverse topological term order (RTTO), i.e. a lex term order with the circuit variables ordered reverse topologically from POs to PIs. As a consequence, remainders generated by polynomial division using this order comprise only PI variables.

1) Polynomial Modeling of Circuits: 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 field $\mathbb{F}_{2^n}$ is constructed as $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$, where $P_n(x)$ is the given primitive polynomial of degree n with γ as a root, i.e. $P_n(\gamma) = 0$. The function implemented by C is modeled with a system of polynomials over $R = \mathbb{F}_{2^n}[Z, A, X]$, where $X = \{x_1, \ldots, x_d\}$ corresponds to all the bit-level variables (nets) in the circuit. Let $X_{PI} \subset X$ denote the set of all primary input variables from C. Further, $Z = \{z_0, \ldots, z_{n-1}\}$ and $A = \{a_0, \ldots, a_{n-1}\}$ represent the output and input operand words of the circuit, respectively.

The Boolean logic gates in C can be represented as polynomials $\pmod 2$ over $\mathbb{F}_2$, using the mapping $\mathbb{B} \mapsto \mathbb{F}_2$:

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

The bit-level to word-level correspondence is represented:

$$\begin{aligned} f_1 : Z &= z_0 + \gamma \cdot z_1 + \cdots + \gamma^{n-1} \cdot z_{n-1} \\ f_2 : A &= a_0 + \gamma \cdot a_1 + \cdots + \gamma^{n-1} \cdot a_{n-1} \end{aligned} \tag{2}$$

Since $\mathbb{F}_2 \subset \mathbb{F}_{2^n}$, the polynomials in Eqn. (1) can also be interpreted as polynomials over $\mathbb{F}_{2^n}$. Thus, the circuit is represented by a set of polynomials $F = \{f_1, \ldots, f_s\} \in R$. Let $J = \langle F \rangle$ be the ideal generated by this set. Let $F_0 = \{x_i^2 - x_i, Y^{2^n} - Y \mid x_i \in \text{bit-level variables}, Y \in \text{word-level variables}\}$ be the set of all vanishing polynomials, and $J_0 = \langle F_0 \rangle$ the corresponding ideal. Then, ideal $J + J_0$ 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, $f \equiv C \iff f \xrightarrow{GB(J+J_0)}_+ 0$ [13]. In the manuscript, we use the circuit of Fig. 1, borrowed from [4], as a running example to demonstrate our approach for MFR.

Example II.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. Lex order: $\{Z\} > \{A > B\} > \{z_0 > z_1 > z_2\} > \cdots > \{d_1 > d_2 > d_3 > r_0 > d_5 > rr_1\} > \{r_1 > rr_3 > rr_2\} > \{r_2 > r_3 > rr_4\} > \{r_4 > d_4\} > \{a_0 > a_1 > a_2 > b_0 > b_1 > b_2\}$.*

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

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

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

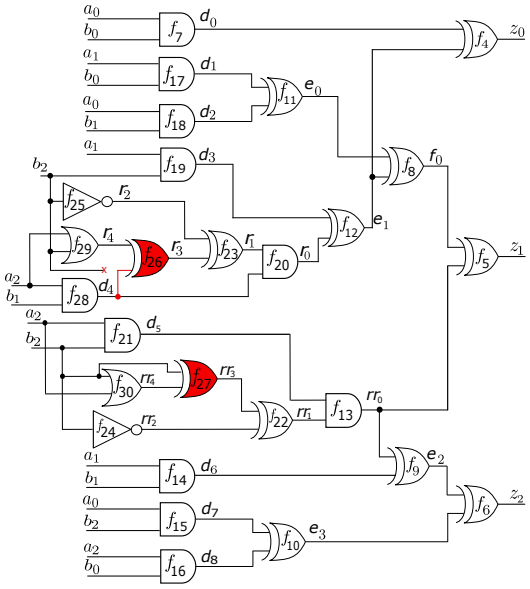

Fig. 1: A 3-bit finite field multiplier *Impl* with bugs introduced at net r_3 (AND gate replaced with an XOR gate and one of the inputs mis-connected to d_4 instead of b_2) and net rr_3 (AND gate replaced with an XOR gate).

III. Rectification Check

In [4], the authors model a given circuit as a polynomial ideal, as described above, and present techniques to perform an MFR check at a set of targets $W = (w_1, \ldots, w_m)$. They set up the MFR check by constructing ideals J_l, $1 \leq l \leq 2^m$, for each $\{0,1\}$-value assignment to the targets. Let $W_c = \{(0,0,\ldots,0), \ldots, (1,1,\ldots,1)\}$ denote the set of all assignments to targets W, where $W_c[l]$ represents one set of assignments to targets W. In each ideal J_l, the polynomials representing the target nets are replaced by the corresponding values from $W_c[l]$. Subsequently, the *Spec* polynomial is reduced by each ideal to produce each remainder rem_l. 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. Finally, the rectifiability of the circuit at the targets is determined by checking that the condition $\prod_{l=1}^{2^m} rem_l \xrightarrow{J_0}_+ 0$ holds (Thm. V.1 [4]). This condition implies that every minterm in the input space is contained in the union of varieties of each rem_l.

Example III.1. *Continuing on with the Ex. II.1, we demonstrate the rectification check presented in [4] for $W = (w_1, w_2) = (r_3, rr_3)$.*

Constructing the J_l ideals:

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

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

- $rem_1 = f \xrightarrow{J_1+J_0}_+ (\gamma+1)a_2b_1b_2 + (\gamma^2+\gamma)a_2b_2$
- $rem_2 = f \xrightarrow{J_2+J_0}_+ (\gamma+1)a_2b_1b_2$
- $rem_3 = f \xrightarrow{J_3+J_0}_+ (\gamma+1)a_2b_1b_2 + a_2b_1 + (\gamma^2+\gamma)a_2b_2$
- $rem_4 = f \xrightarrow{J_4+J_0}_+ (\gamma+1)a_2b_1b_2 + a_2b_1$

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

The techniques presented in [4] are limited to proving the *existence* of rectification functions at W. This paper utilizes the remainders described above to characterize and compute rectification functions, as shown in the following section.

IV. 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 rectify the circuit. 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.

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 corresponding to targets w_1 and w_2, respectively. For brevity, let $V_{W_c[l]} = V(rem_l)$, for $1 \leq l \leq 2^m$; in this case, $V_{W_c[1]} = V_{(0,0)} = V(rem_1), V_{W_c[2]} = V_{(0,1)} = V(rem_2)$, and so on.

Recall that $V_{(0,0)}$ comprises the set of points where the *Spec* matches the *Impl* under the assignments $w_1 = w_2 = 0$ to the targets. This implies that at these points, the rectification functions u_1 and u_2 should evaluate to 0. Table II shows the required evaluations of u_1 and u_2 for the points in each variety, following the same reasoning, assuming each $V_{W_c[l]}$ is pairwise disjoint. The on(off)-set of the rectification function for a target corresponds to the union of the varieties where the function evaluates to $1(0)$. In this case, the on- and off-sets of u_1 consists of the set of points in $V_{(1,0)} \cup V_{(1,1)}$ and $V_{(0,0)} \cup V_{(0,1)}$, respectively. Similarly, the on- and off-set of u_2 comprise points in $V_{(0,1)} \cup V_{(1,1)}$ and $V_{(0,0)} \cup V_{(1,0)}$. The functions u_1 and u_2 could be synthesized using these sets.

TABLE II: Rectification 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 each $V_{W_c[l]}$ 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) may 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.

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

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):

$$S_l = \begin{cases} V_{W_c[1]}, & \text{if } l = 1 \\ V_{W_c[l]} \setminus (\bigcup_{j=1}^{l-1} V_{W_c[j]}), & 2 \leq l \leq 2^m \end{cases} \tag{3}$$

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

$$\begin{aligned} V(u_{i_{on}}) &= \bigcup S_l, \ \forall l \mid W_c[l][i] = 1 \\ V(u_{i_{off}}) &= \bigcup S_l, \ \forall l \mid W_c[l][i] = 0 \end{aligned} \tag{4}$$

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 two sets of assignments to the targets $W_c[j]$ 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 $1 \leq j, k \leq 2^m$.

$$DC_{pair} = \{V_{W_c[j]} \cap V_{W_c[k]} \mid d(W_c[j], W_c[k]) = 1\} \tag{5}$$

Since the Hamming distance $d = 1$ between the assignments $W_c[j]$ 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 below.

3) 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 outside of 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.

$$S_l^d = \begin{cases} V_{W_c[1]} \setminus DC_{pair}, & \text{if } l = 1 \\ V_{W_c[l]} \setminus ((\bigcup_{j=1}^{l-1} V_{W_c[j]}) \cup DC_{pair}), & 2 \leq l \leq 2^m \end{cases} \tag{6}$$

Points in these composite sets are assigned to the on- and off-set for each rectification function in the same way as Eqn. (4), substituting S_l with S_l^d. Next, we place the points in DC_{pair} in the on-, off-, or DC sets for each rectification function, by imposing an order on the intersections and resolving them as explained in the following example.

Given 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 impose the order $V_{(0,0)} > V_{(0,1)} > V_{(1,0)} > V_{(1,1)}$. 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.

4) Synthesizing Rectification Functions: The above techniques show how to construct a rectification function by reasoning about the varieties of each rem_l. However, algebraically, we compute these functions using their corresponding ideals. Specifically, we show how the remainders computed in Sec.III 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 $\{0,1\}$-assignments to bit-level X_{PI}. In [10], it was shown that the reduced Gröbner bases of such ideals ($\langle rem_l, J_0 \rangle$) only contain coefficients from $\mathbb{F}_2$. Consequently, the rectification function operations are restricted to algebraic computations in $\mathbb{F}_2[X_{PI}]$, i.e. in Boolean rings.

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

1) Compute a reduced Gröbner basis for each $\langle rem_l, J_0 \rangle$.
2) Impose the order on the remainders: $rem_1 > \cdots > rem_l$.
3) Greedy approach: Compute composite sets from Eqn. (3).
   - Assign points from composite sets to the on- or off-sets of the rectification functions (sec. IV-1).
4) DC-based approach: Compute DC_{pair} using Eqn. (5), and then compute the composite sets in Eqn. (6)
   - Assign the points in the composite sets and DC_{pair} to the DC-, on- or off-sets of the rectification functions (Sec. IV-2).
5) Perform the union, intersection, and set difference of varieties using the respective ideal operations (Table I).
6) Translate the ideals representing $u_{i_{DC}}$ and $u_{i_{on}}$ into Boolean functions by interpreting the algebraic product and sum as Boolean AND and XOR gates, respectively.
7) Optimize the on-set $u_{i_{on}}$ w.r.t. to the DC-set $u_{i_{DC}}$ for the DC-based approach.

Example IV.1. *Continuing with Ex. III.1, consider rem_3:*

- $rem_3 = (\gamma + 1)a_2b_1b_2 + a_2b_1 + (\gamma^2 + \gamma)a_2b_2$.
- $redGB(\langle rem_3, J_0 \rangle) = \{a_2b_1, a_2b_2\}$
  - *Repeat for rem_1, rem_2, and rem_4.*
  - *Note, $V(rem_3) = V(redGB(\langle rem_3, J_0 \rangle))$*
- *Impose the order $rem_1 > rem_2 > rem_3 > rem_4$.*
- *The rectification polynomials for the targets (r_3, rr_3) computed using steps 3, 5, and 6 from the procedure:*

$$u_{1_{on}} = a_2b_1b_2; \qquad u_{2_{on}} = a_2b_2;$$
$$r_3 = u_1 = (a_2 \wedge b_1 \wedge b_2); \qquad rr_3 = u_2 = (a_2 \wedge b_2);$$

- *The rectification polynomials for the targets (r_3, rr_3) computed using steps 4-7 from the procedure:*

$$u_{1_{on}} = a_2b_1b_2; \qquad u_{2_{on}} = a_2b_2;$$
$$u_{1_{dc}} = a_2b_1b_2 + a_2b_2; \qquad u_{2_{dc}} = a_2b_2 + 1;$$
$$r_3 = u_1 = a_2 \wedge b_2; \qquad rr_3 = u_2 = 1;$$

V. Experimental Results

The benchmark suite includes a Mastrovito multiplier and a circuit that performs Point Addition over NIST standard Elliptic curves. These benchmarks are taken from [13] and are synthesized using the *abc* tool with a gate library. We introduce gate and wiring faults in the netlists such that bugs propagate to multiple POs. The faults are placed at various topological levels; some faults are placed closer to PIs, some are placed in the middle of the circuit, and some near POs. In our experiments, we choose between two to ten targets m, based on the number of bugs introduced in the benchmark.

Our custom software is implemented using the programming language Python as a wrapper around libraries from PolyBori [14], *abc*, and *sis*. The wrapper processes the netlists of the *Impl* and *Spec* and uses Polybori's ZDD datastructure to model and efficiently perform the polynomial divisons $f \xrightarrow{J_l + J_0}_+ rem_l$. The software utilizes the remainders rem_l to perform an MFR check at the targets. It then computes the rectification functions by performing the algebraic ideal operations. Finally, the software uses *sis* [15] and *abc* [16] 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.IV-2. We use *abc* to perform *structural hashing, functional reduction, balancing, refactoring, rewriting, etc.* Finally, we *map* the computed functions using a library of AND-XOR gates and extract the synthesis results for *Area* and *Delay*. The experiments are performed on a 3.5GHz Intel(R) Core™ i7-4770K Quad-Core CPU with 32 GB RAM.

Table III presents the datapath size of the faulty benchmarks, the number of given targets, the total number of outputs affected by the bugs, the execution time for different stages of the approach, and the area and delay of the resulting patch sub-circuits after *abc* and *sis* optimization. The execution time TGC includes the total time required to generate the patch sub-circuit from the remainders and to perform re-synthesis (using *abc*) of the original faulty circuit after integrating the patch at the given targets. During TDC, we perform patch generation, attempt logic optimization (with *sis*) using the computed don't cares, and then perform re-synthesis (using *abc*) of the patch integrated benchmark.

Columns PGC and PDC denote the post-optimization synthesis results for the greedy approach and the DC-based approach, respectively. The synthesis results computed in PDC where *sis* completed simplification successfully contain a smaller area and delay than the ones computed in PGC. The asterisk (*) in the PDC column denotes the cases where *full simplify* fails to utilize the don't care network for function simplification. In these cases, *sis* aborts simplification because the BDD size exceeds 480,000 nodes. For these entries, the patch functions are synthesized using *abc* which ignores the provided don't care network.

The execution time for function computation (TGC and TDC) and the resulting patch area and delay depend on factors such as: i) the number of bugs; ii) the number of targets; iii) the location of the bugs; iv) the location of the targets; v) the number of affected outputs; and vi) the size of

TABLE III: Time in seconds; n = Datapath size, m = Number of targets, FO = Number of faulty outputs, TPS = Execution time of PolyBori setup (ring declaration/poly collection/spec collection), TRC = Execution time for verification, multi-fix setup, and rectifiability check, TGC = Execution time for function computation using the greedy approach, TDC = Execution time for function computation with DC-based approach, PGC = Synthesized patch sub-circuit using the greedy approach, PDC = Patch sub-circuit using the DC-based approach, A = Area in terms of number of gates , D = Longest delay

n	Mastrovito multiplier										Point Addition circuit									
	m	FO	TPS	TRC	TGC	TDC	PGC		PDC		m	FO	TPS	TRC	TGC	TDC	PGC		PDC	
							A	D	A	D							A	D	A	D
16	**5**	10	0.1	0.1	7	10	19	3	17	3	**5**	7	0.07	0.02	14	41	529	25	217	17
32	**10**	21	0.1	0.3	†1620	†1810	10	1	10*	1*	**5**	13	0.2	0.06	76	675	1314	33	890	29
64	**5**	10	0.6	9	106	280	1675	29	1577*	46*	**3**	64	0.8	0.1	310	315	2288	30	2250*	29*
96	**7**	15	1.5	0.2	50	110	1980	51	3435*	39*	**5**	14	2.4	0.3	390	431	23	6	20	4
128	**4**	7	3.1	0.2	80	94	334	18	211	13	**2**	128	6.42	4.38	485	491	5930	33	6340*	31*
163	**5**	6	6.4	0.4	72	117	122	9	95	7	**5**	22	15.9	1.53	152	177	50	10	24	4
233	**8**	12	13	0.6	151	223	19	4	17	3	**2**	233	19.7	1.44	245	246	4980	30	3150*	27*
409	**6**	7	190	2	403	414	26	4	24	4	**2**	409	224	5.3	756	762	2226	21	2000*	21*
571	**4**	8	2143	6	957	979	29	4	27	4	**2**	5	2492	13.2	1.2	20	622	22	210	16

the remainder rem_l. The cells marked † in Table III show anomalous data where the area of the pre-optimized patch circuit was approximately 20,000 gates, in part due to the large number of targets selected. At the cost of significant execution time, *abc* was able to optimize the patch to 10 gates.

We compare our results to the SAT-based procedures [17] and [18] which we implemented using *abc*. Table IV presents the execution time for the rectification of faulty Mastrovito multipliers for these approaches. As shown, the SAT-based procedures time out while rectifying benchmarks at a single target beyond ten-bit operand words.

TABLE IV: Rectification of Mastrovito Multipliers using SAT-based techniques; n = Operand width, m = Number of targets, #G = number of gates in the miter model, Time-Out (TO) = 6000s

n	m	#G	[17]	[18]
8	1	380	158	24
9	1	510	4507	37
10	1	750	TO	215
12	1	1000	TO	TO

Table V compares the area and delay between the original benchmarks and the patch-integrated benchmarks. As shown, the area is comparable to the original faulty benchmark after resynthesis (using *abc*). The delay, however, is significantly longer in some cases, due to a long chain of logic introduced by our model during the rectification function computations.

VI. 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 DC conditions for the target functions. Our approach is able to compute rectification functions for circuits with large (NIST-standard) operand widths n. Improving the model to reduce the delay of the patch circuits, and computing functions in terms of internal nets comprise part of our future work. We are also investigating the extension of this approach to integer arithmetic circuits.

TABLE V: Comparison of Area and Delay between original benchmark and patch-integrated rectified benchmarks. n = Datapath size, ORIG = Original faulty benchmark, Greedy = Benchmark rectified using patch from our greedy approach, DC-based = Benchmark rectified using patch from our DC-based approach, A = Area in terms of number of gates $\times 10^3$, D = Longest delay

n	Mastrovito multiplier						Point Addition circuit					
	ORIG		Greedy		DC-based		ORIG		Greedy		DC-based	
	A	D	A	D	A	D	A	D	A	D	A	D
16	.81	10	.81	10	.81	10	0.9	16	1.27	32	0.91	22
32	2.9	12	2.8	12	2.8	12	2.9	14	3.3	21	3.03	21
64	11.2	18	11.5	30	11.3	33	10.7	17	10.8	28	10.9	30
96	24.5	16	24.5	58	25.1	45	24.8	18	24.8	18	24.8	17
128	43.2	12	43.2	21	41.5	17	43.2	16	43.4	31	43.5	29
163	69.8	12	68.1	13	68.1	13	71.6	16	71.6	16	71.6	16
233	119	12	118.5	13	118.5	13	122	14	122.8	23	122.2	19
409	384	12	384	12	384	12	368	14	368.1	14	368	14
571	827	15	825	15	825	15	813	17	813	18	813	17

References

[1] A. Q. Dao, N.-Z. Lee, L.-C. Chen, M. P.-H. Lin, J.-H. R. Jiang, A. Mishchenko, and R. Brayton, "Efficient Computation of ECO Patch Functions," in *DAC*, 2018, pp. 51:1–51:6.

[2] Y. Kimura, A. M. Gharehbaghi, and M. Fujita, "Signal Selection Methods for Efficient Multi-Target Correction," in *ISCAS*, 2019, pp. 1–5.

[3] V. N. Kravets, N. Lee, and J. R. Jiang, "Comprehensive Search for ECO Rectification Using Symbolic Sampling," in *DAC*, 2019, pp. 1–6.

[4] V. Rao, I. Ilioaea, H. Ondricek, P. Kalla, and F. Enescu, "Word-level multi-fix rectifiability of finite field arithmetic circuits," in *ISQED*, 2021.

[5] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence Checking of Partial Designs Using Dependency Quantified Boolean Formulae," in *ICCD*, 2013.

[6] K. F. Tang, P. K. Huang, C. N. Chou, and C. Y. Huang, "Multi-patch Generation for Multi-error Logic Rectification by Interpolation with Cofactor Reduction," in *DATE*, 2012, pp. 1567–1572.

[7] H. T. Zhang and J. H. R. Jiang, "Cost-aware Patch Generation for Multi-target Function Rectification of Engineering Change Orders," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[8] F. Farahmandi and P. Mishra, "Automated Debugging of Arithmetic Circuits Using Incremental Gröbner Basis Reduction," in *ICCD*, 2017.

[9] A. Mahzoon, D. Große, and R. Drechsler, "Combining Symbolic Computer Algebra and Boolean Satisfiability for Automatic Debugging and Fixing of Complex Multipliers," in *ISVLSI*, 2018, pp. 351–356.
[10] U. Gupta, I. Ilioaea, V. Rao, A. Srinath, P. Kalla, and F. Enescu, "On the Rectifiability of Arithmetic Circuits using Craig Interpolants in Finite Fields," in *VLSI-SoC*, Oct 2018, pp. 49–54.
[11] V. Rao, U. Gupta, A. Srinath, I. Ilioaea, P. Kalla, and F. Enescu, "Post-Verification Debugging and Rectification of Finite Field Arithmetic Circuits using Computer Algebra Techniques," in *FMCAD*, 2018.
[12] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*. American Mathematical Society, 1994.
[13] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits," in *IEEE Trans. on CAD*, vol. 32, no. 9, 2013, pp. 1409–1420.
[14] M. Brickenstein and A. Dreyer, "Polybori: A framework for gröbner-basis computations with boolean polynomials," *J. Symb. Comput.*, 2009.
[15] E. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis," ERL, Dept. of EECS, Univ. of California, Berkeley., Tech. Rep. UCB/ERL M92/41, 1992.
[16] R. Brayton and A. Mishchenko, "Abc: An Academic Industrial-strength Verification Tool," in *Computer Aided Verification*, 2010.
[17] K. F. Tang, C. A. Wu, P. K. Huang, and C. Y. Huang, "Interpolation-Based Incremental ECO Synthesis for Multi-Error Logic Rectification," in *Proc. Design Automation Conf. (DAC)*, 2011, pp. 146–151.
[18] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design," *Proceedings of the IEEE*, 2015.