Rectification of Arithmetic Circuits with Craig Interpolants in Finite Fields

Utkarsh Gupta[1(✉)], Irina Ilioaea[2], Vikas Rao[1], Arpitha Srinath[1], Priyank Kalla[1], and Florian Enescu[2]

[1] Electrical and Computer Engineering, University of Utah, Salt Lake City, UT, USA
{utkarsh.gupta,vikas.k.rao,arpitha.srinath}@utah.edu, kalla@ece.utah.edu
[2] Mathematics and Statistics, Georgia State University, Atlanta, GA, USA
iilioaea1@student.gsu.edu, fenescu@gsu.edu

Abstract. When formal verification of arithmetic circuits identifies the presence of a bug in the design, the task of rectification needs to be performed to correct the function implemented by the circuit so that it matches the given specification. In our recent work [26], we addressed the problem of rectification of buggy finite field arithmetic circuits. The problems are formulated by means of a set of polynomials (ideals) and solutions are proposed using concepts from computational algebraic geometry. Single-fix rectification is addressed – i.e. the case where any set of bugs can be rectified at a single net (gate output). We determine if single-fix rectification is possible at a particular net, formulated as the Weak Nullstellensatz test and solved using Gröbner bases. Subsequently, we introduce the concept of Craig interpolants in polynomial algebra over finite fields and show that the rectification function can be computed using algebraic interpolants. This article serves as an extension to our previous work, provides a formal definition of Craig interpolants in finite fields using algebraic geometry and proves their existence. We also describe the computation of interpolants using elimination ideals with Gröbner bases and prove that our procedure computes the smallest interpolant. As the Gröbner basis algorithm exhibits high computational complexity, we further propose an efficient approach to compute interpolants. Experiments are conducted over a variety of finite field arithmetic circuits which demonstrate the superiority of our approach against SAT-based approaches.

Keywords: Rectification · Arithmetic circuits · Gröbner bases · Craig interpolants

1 Introduction

The past decade has witnessed extensive investigations into formal verification of arithmetic circuits. Circuits that implement polynomial computations over

This research is funded in part by the US National Science Foundation grants CCF-1619370 and CCF-1320385.

Published by Springer Nature Switzerland AG 2019
N. Bombieri et al. (Eds.): VLSI-SoC 2018, IFIP AICT 561, pp. 79–106, 2019.
https://doi.org/10.1007/978-3-030-23425-6_5

large bit-vector operands are hard to verify automatically using methods such as SAT/SMT-solvers, decision diagrams, etc. Recent techniques have investigated the use of polynomial algebra and algebraic geometry techniques for their verification. These include verification of integer arithmetic circuits [1–3], integer modulo-arithmetic circuits [4], word-level RTL models of polynomial datapaths [5,6], finite field combinational circuits [7–9], and also sequential designs [10]. A common theme among the above approaches is that designs are modeled as sets of polynomials in rings with coefficients from integers $\mathbb{Z}$, finite integer rings $\mathbb{Z}_{2^k}$, finite fields $\mathbb{F}_{2^k}$, and more recently also from the field of fractions $\mathbb{Q}$. Subsequently, the verification checks are formulated using algebraic geometry [11] (e.g., the Nullstellensatz), and *Gröbner basis (GB)* theory and technology [12] are used as decision procedures (ideal membership test) for formal verification.

While these techniques are successful in proving correctness or detecting the presence of bugs, *the task of post-verification debugging, error diagnosis and rectification of arithmetic circuits has not been satisfactorily addressed.* Debugging and rectification of arithmetic circuits is of utmost importance. Arithmetic circuits are mostly custom designed; this raises the potential for errors in the implementation, which have to be eventually rectified. Instead of redesigning the whole circuit, it is desirable to synthesize rectification sub-functions with minimal topological changes to the existing design – a problem often termed as *partial synthesis*. Moreover, the debug, rectification and partial synthesis problem is analogous to that of synthesis for Engineering Change Orders (ECO), where the current circuit implementation should be minimally modified (rectified) to match the ECO-modified specification. The partial synthesis approach also applies here to generate ECO-patches for rectification.

The problem of debug, rectification and ECO synthesis has been addressed for control-dominated applications and random-logic circuits, where the early developments of [13–15] were extended by [16] by formulating as CNF-SAT, and computing rectification functions using *Craig Interpolants* [17] in propositional logic.

Craig Interpolation (CI) is a method in automated reasoning to construct and refine abstractions of functions. It is a logical tool to extract concise explanations for the infeasibility of a set of mutually inconsistent statements. As an alternative to quantifier elimination, CI finds application in verification as well as in partial synthesis – and therefore, in rectification. In propositional logic, they are defined as follows.

Definition 1.1 (Craig Interpolants)**.** Let (A, B) be a pair of CNF formulas (sets of clauses) such that $A \wedge B$ is unsatisfiable. Then there exists a formula I such that: (i) $A \implies I$; (ii) $I \wedge B$ is unsatisfiable; and (iii) I refers only to the common variables of A and B, i.e. $Var(I) \subseteq Var(A) \cap Var(B)$. The formula I is called the **interpolant** of (A, B).

Despite these advancements in automated debugging and rectification of control and random logic circuits, the aforementioned SAT and CI-based approaches are infeasible for rectification of arithmetic circuits.

1.1 Problem Description, Objectives, and Contributions

We address the problem of rectification of buggy finite field arithmetic circuits. Our problem setup is as follows:

- A specification model (*Spec*) is given either as a polynomial description f_{spec} over a finite field, or as a golden model of a finite field arithmetic circuit. The finite field considered is the field of 2^k elements (denoted by $\mathbb{F}_{2^k}$), where k is the operand-width (bit-vector word length). An implementation (*Impl*) circuit C is also given.
- Equivalence checking is performed between the *Spec* and the *Impl* circuit C, and the presence of a bug is detected. No restrictions on the number, type, or locations of the bugs are assumed.
- We assume that error-diagnosis has been performed, and a subset X of the nets of the circuit is identified as *potential rectification locations*, called target nets.

Given the *Spec*, the buggy *Impl* circuit C, the set X of potential rectifiable locations, our objective is to determine whether or not the buggy circuit can be rectified at *one particular net (location)* $x_i \in X$. This is called **single-fix rectification** in literature [16]. If a single-fix rectification does exist at net x_i in the buggy circuit, then our subsequent objective is to derive a polynomial function $U(X_{PI})$ in terms of the set of primary input variables X_{PI}. This polynomial needs to be further translated (synthesized) into a logic sub-circuit such that $x_i = U(X_{PI})$ acts as the rectification function for the buggy *Impl* circuit C so that this modified C matches the specification.

Given the above objective, this article makes the following specific contributions to solve the debug and rectification problem.

1. We formulate the test for single-fix rectifiability at a net x_i using concepts and techniques from algebraic geometry [12].
   - The problem is modeled in polynomial rings of the form $\mathbb{F}_{2^k}[x_1, \ldots, x_n]$, where k corresponds to the operand-width and the variables $x_1, \ldots, x_n$ are the nets of the circuit.
   - The rectification test is formulated using elimination ideals and the Weak Nullstellensatz, and solved using Gröbner basis as a decision procedure.
2. If rectification is feasible at x_i, then we compute a rectification function $x_i = U(X_{PI})$.
   - We show that the rectification function $U(X_{PI})$ can be determined based on the concept of Craig interpolants in algebraic geometry. While Craig interpolation is a well-studied concept in propositional and first-order logic theories, to the best of our knowledge, it has not been investigated in algebraic geometry.
   - We define Craig interpolants in polynomial algebra in finite fields and prove their existence. We also show how to compute such an interpolant using Gröbner bases.

3. The rectification function $U(X_{PI})$ obtained using Craig interpolants is a polynomial in $\mathbb{F}_{2^k}[x_1, \ldots, x_n]$. We subsequently show how a logic circuit can be obtained from this polynomial.
4. We use Gröbner basis not only as a decision procedure for the rectification test, but also as a quantification procedure for computing the rectification function. Computation of Gröbner bases exhibits very high complexity. To make our approach scalable, we further show how to exploit the topological structure of the given circuit to improve this computation.

We demonstrate the application of our techniques to rectify finite field arithmetic circuits with large operand sizes, where conventional SAT-solver based rectification approaches are infeasible.

The paper is organized as follows. The following section reviews previous work in automated diagnosis and rectification, and recent applications of Craig interpolants. Section 3 describes concepts from computer algebra and algebraic geometry. Section 4 describes an equivalence checking framework using the Weak Nullstellensatz over finite fields. Section 5 presents results that ascertain the single-fix rectifiability of the circuit. Section 6 introduces Craig interpolants in finite fields using Gröbner basis methods, and gives a procedure for obtaining the rectification function through algebraic interpolants. Section 7 addresses improvements to the Gröbner basis computation. Section 8 presents our experimental results and Sect. 9 concludes the paper.

2 Review of Previous Work

Automated diagnosis and rectification of digital circuits has been addressed in [13,18]. The paper [14] presents algorithms for synthesizing Engineering Change Order (ECO) patches. The partial equivalence checking problem has been addressed in [15,19] that checks whether a partial implementation can be extended to a complete design so that it becomes equivalent to a given specification. The partial implementation comprises black-boxes for which some functions f_i's need to be computed. The problem is formulated as Quantified Boolean Formula (QBF) solving: does there exist a function f_i, such that for all primary input assignments, the *Impl* circuit is equivalent to the *Spec* circuit. Incremental SAT-solving based approach has been presented in [20] in lieu of solving the QBF problem. This approach has been extended in [21,22] to generate rectification functions when the *Impl* circuit topology is fixed. The use of Craig interpolation as an alternative to quantifier elimination has been presented in [16,23,24] for ECO applications. The single-fix rectification function approach in [23] has been extended in [16] to generate multiple partial-fix functions. Recently, an efficient approach on resource aware ECO patch generation has been presented in [25].

As these approaches are SAT based, they work well for random logic circuits but are not efficient for arithmetic circuits. In contrast, this article presents a word-level formulation for single-fix rectification using algebraic geometry techniques. Computer algebra has been utilized for circuit debugging and rectification in [27–29]. These approaches rely heavily on the structure of the circuit

for debugging, and in general, are incomplete. If the arithmetic circuit contains redundancies, the approach may not identify the buggy gate, nor compute the rectification function. On the other hand, our approach is complete, as it can always compute a single-fix rectification function, if one exists. Although our polynomial algebra based approach is applicable to any circuit in general, it is more efficient and practical for finite field arithmetic circuits.

The concept of Craig interpolants has been extensively investigated in many first order theories for various applications in synthesis and verification. Given the pair (A, B) of two mutually inconsistent formulas (cf. Definition 1.1) and a proof of their unsatisfiability, a procedure called the *interpolation system* constructs the interpolant in linear time and space in the size of the proof [30]. As the abilities of SAT solvers for proof refutation have improved, interpolants have been exploited as abstractions in various problems that can be formulated as unsatisfiable instances, e.g. model checking [30], logic synthesis [31], etc. Their use as abstractions have also been replicated in other (combinations of) theories [32–35], etc. However, the problem has been insufficiently investigated over polynomial ideals in finite fields from an algebraic geometry perspective. In that regard, the works that come closest to ours are by Gao *et al.* [36] and [37]. While they do not address the interpolation problem per se, they do describe important results of Nullstellensatz, projections of varieties and quantifier elimination over finite fields that we utilize to develop the theory and algorithms for our approach. Moreover, prior to debugging, our approach requires that verification be performed to detect the presence of a bug. For this purpose, we make use of techniques presented in [7,8,38].

We have described the notion of Craig interpolants in finite fields in our work [26]. This article is an extended version of that work where we formally define Craig interpolants in finite fields and prove their existence. Moreover, we describe a procedure for computing an interpolant and prove that the computed interpolant is the smallest. The computation of interpolants uses Gröbner basis based algorithms which have high computational complexity. In contrast to [26], we further propose an efficient approach to compute interpolants based on the given circuit topology.

3 Preliminaries: Notation and Background Results

Let $\mathbb{F}_q$ denote the finite field of q elements where $q = 2^k$, $\overline{\mathbb{F}}_q$ be its algebraic closure, and k is the operand width. The field $\mathbb{F}_{2^k}$ is constructed as $\mathbb{F}_{2^k} \equiv \mathbb{F}_2[x] \pmod{P(x)}$, where $\mathbb{F}_2 = \{0, 1\}$, and $P(x)$ is a primitive polynomial of degree k. Let α be a primitive element of $\mathbb{F}_{2^k}$, so that $P(\alpha) = 0$. Let $R = \mathbb{F}_q[x_1, \ldots, x_n]$ be the polynomial ring in n variables $x_1, \ldots, x_n$, with coefficients from $\mathbb{F}_q$. A monomial is a power product of variables $x_1^{e_1} \cdot x_2^{e_2} \cdots x_n^{e_n}$, where $e_i \in \mathbb{Z}_{\geq 0}, i \in \{1, \ldots, n\}$. A *polynomial* $f \in R$ is written as a finite sum of terms $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$, where $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ are monomials. A monomial order $>$ (or a term order) is imposed on the ring – i.e. a total order and a well-order on all the monomials of R s.t. multiplication with

another monomial preserves the order. Then the monomials of all polynomials $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$ are ordered w.r.t. $>$, such that $X_1 > X_2 > \cdots > X_t$, where $lm(f) = X_1$, $lt(f) = c_1X_1$, and $lc(f) = c_1$ are called the *leading monomial*, *leading term*, and *leading coefficient* of f, respectively. In this work, we employ lexicographic (lex) term orders (see Definition 1.4.3 in [12]).

Polynomial Reduction via Division: Let $f,\ g$ be polynomials. If $lm(f)$ is divisible by $lm(g)$, then we say that f *is reducible to* r modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{lt(f)}{lt(g)} \cdot g$. This operation forms the core operation of polynomial division algorithms and it has the effect of canceling the leading term of f. Similarly, f can be *reduced w.r.t. a set of polynomials* $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r. This reduction is denoted as $f \xrightarrow{F}_+ r$, and the remainder r has the property that no term in r is divisible (i.e. cannot be canceled) by the leading term of any polynomial f_i in F.

We model the given circuit C by a set of multivariate polynomials $f_1, \ldots, f_s \in \mathbb{F}_{2^k}[x_1, \ldots, x_n]$; here $x_1, \ldots, x_n$ denote the nets (signals) of the circuit. Every Boolean logic gate of C is represented by a polynomial in $\mathbb{F}_2$, as $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$. This is shown below. Note that in $\mathbb{F}_{2^k}$, $-1 = +1$.

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

Definition 3.1 (Ideal of polynomials)**.** *Given a set of polynomials* $F = \{f_1, \ldots, f_s\}$ *in* $\mathbb{F}_q[x_1, \ldots, x_n]$*, the ideal* $J \subseteq R$ *generated by* F *is,*

$$J = \langle f_1, \ldots, f_s \rangle = \{\sum_{i=1}^{s} h_i \cdot f_i :\ h_i \in \mathbb{F}_q[x_1, \ldots, x_n]\}.$$

The polynomials $f_1, \ldots, f_s$ *form the basis or the generators of* J*.*

Let $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$ be a point in the affine space, and f a polynomial in R. If $f(\boldsymbol{a}) = 0$, we say that f *vanishes* on $\boldsymbol{a}$. In verification, we have to analyze the *set of all common zeros* of the polynomials of F that lie within the field $\mathbb{F}_q$. In other words, we need to analyze solutions to the system of polynomial equations $f_1 = f_2 = \cdots = f_s = 0$. This zero set is called the *variety*. It depends not just on the given set of polynomials but rather on the ideal generated by them. We denote it by $V(J) = V(f_1, \ldots, f_s)$, and it is defined as follows:

Definition 3.2 (Variety of an ideal)**.** *Given a set of polynomials* $F = \{f_1, \ldots, f_s\}$ *in* $\mathbb{F}_q[x_1, \ldots, x_n]$*, their variety*

$$V(J) = V(f_1, \ldots, f_s) = \{\boldsymbol{a} \in \mathbb{F}_q^n : \forall f \in J, f(\boldsymbol{a}) = 0\}$$

We denote the complement of a variety, $\mathbb{F}_q^n \setminus V(J)$, by $\overline{V(J)}$.

The Weak Nullstellensatz: To ascertain whether $V(J) = \emptyset$, we employ the Weak Nullstellensatz over $\mathbb{F}_q$, for which we use the following notations.

Definition 3.3 (Sum and Product of Ideals)**.** *Given two ideals* $J_1 = \langle f_1, \ldots, f_s \rangle, J_2 = \langle h_1, \ldots, h_r \rangle$*, their sum and product are*

$$J_1 + J_2 = \langle f_1, \ldots, f_s, h_1 \ldots, h_r \rangle$$
$$J_1 \cdot J_2 = \langle f_i \cdot h_j : 1 \leq i \leq s, 1 \leq j \leq r \rangle$$

Ideals and varieties are dual concepts: $V(J_1 + J_2) = V(J_1) \cap V(J_2)$*, and* $V(J_1 \cdot J_2) = V(J_1) \cup V(J_2)$*. Moreover, if* $J_1 \subseteq J_2$ *then* $V(J_1) \supseteq V(J_2)$*.*

For all elements $\alpha \in \mathbb{F}_q, \alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ vanishes everywhere in $\mathbb{F}_q$, and is called the vanishing polynomial of the field. Let $J_0 = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$ be the ideal of all vanishing polynomials in R. Then the variety of ideal J_0 is the entire affine space, i.e. $V(J_0) = \mathbb{F}_q^n$. Moreover, by extending any ideal $J \in R = \mathbb{F}_q[x_1, \ldots, x_n]$ by the ideal of all vanishing polynomials in R, the variety is restricted to points within $\mathbb{F}_q^n$, i.e. $V(J + J_0) \subset \mathbb{F}_q^n$.

Theorem 3.1 (The Weak Nullstellensatz over finite fields (from Theorem 3.3 in [37]))**.** *For a finite field* $\mathbb{F}_q$ *and the ring* $R = \mathbb{F}_q[x_1, \ldots, x_n]$*, let* $J = \langle f_1, \ldots, f_s \rangle \subseteq R$*, and let* $J_0 = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$ *be the ideal of vanishing polynomials. Then* $V(J) = \emptyset \iff 1 \in J + J_0$*.*

To determine whether $V(J) = \emptyset$, we need to test whether or not the unit element 1 is a member of the ideal $J + J_0$. For this *ideal membership test, we need to compute a Gröbner basis of* $J + J_0$*.*

Gröbner Basis of Ideals: An ideal may have many different sets of generators: $J = \langle f_1, \ldots, f_s \rangle = \cdots = \langle g_1, \ldots, g_t \rangle$. Given a non-zero ideal J, a *Gröbner basis* (GB) for J is one such set $G = \{g_1, \ldots, g_t\}$ that possesses important properties that allow to solve many polynomial decision problems.

Definition 3.4 (Gröbner basis [12])**.** *For a monomial ordering* $>$*, a set of non-zero polynomials* $G = \{g_1, g_2, \ldots, g_t\}$ *contained in an ideal* J*, is called a Gröbner basis of* J *iff* $\forall f \in J, f \neq 0$*, there exists* $g_i \in \{g_1, \ldots, g_t\}$ *such that* $lm(g_i)$ *divides* $lm(f)$*; i.e.,* $G = GB(J) \Leftrightarrow \ \forall f \in J : f \neq 0, \ \exists g_i \in G : lm(g_i) \mid lm(f)$*.*

Then $J = \langle G \rangle$ holds and so $G = GB(J)$ forms a basis for J. Buchberger's algorithm [39] is used to compute a Gröbner basis. The algorithm, shown in Algorithm 1, takes as input the set of polynomial $F = \{f_1, \ldots, f_s\}$ and computes their Gröbner basis $G = \{g_1, \ldots, g_t\}$ such that $J = \langle F \rangle = \langle G \rangle$, where the variety $V(\langle F \rangle) = V(\langle G \rangle) = V(J)$. In the algorithm,

$$Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j \tag{2}$$

where $L = LCM(lm(f_i), lm(f_j))$.

Algorithm 1. Buchberger's Algorithm

Require: $F = \{f_1, \ldots, f_s\}$
Ensure: $G = \{g_1, \ldots, g_t\}$
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: $\quad\quad\quad G := G \cup \{h\}$

A GB may contain redundant polynomials, and it can be *reduced* to eliminate these redundant polynomials from the basis. A reduced GB is a canonical representation of the ideal. Moreover, when $1 \in J$, then $G = reduced\_GB(J) = \{1\}$. Therefore, to check if $V(J) = \emptyset$, from Theorem 3.1 we compute a reduced GB G of $J + J_0$ and see if $G = GB(J + J_0) = \{1\}$. If so, the generators of ideal J do not have any common zeros in $\mathbb{F}_q^n$.

Craig Interpolation: The Weak Nullstellensatz is the polynomial analog of SAT checking. For UNSAT problems, the formal logic and verification communities have explored the notion of abstraction of functions by means of Craig interpolants, which has been applied to circuit rectification [16].

Given the pair (A, B) and their refutation proof, a procedure called the *interpolation system* constructs the interpolant in linear time and space in the size of the proof [30]. We introduce the notion of Craig interpolants in polynomial algebra over finite fields, based on the results of the Nullstellensatz. We make use of the following definitions and theorems for describing the results on Craig interpolants in finite fields.

Definition 3.5. Given an ideal $J \subset R$ and $V(J) \subseteq \mathbb{F}_q^n$, 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 $I_1 \subset I_2$ are ideals then $V(I_1) \supset V(I_2)$, and similarly if $V_1 \subset V_2$ are varieties, then $I(V_1) \supset I(V_2)$.

Definition 3.6. For any ideal $J \subset R$, the **radical** of J is defined as $\sqrt{J} = \{f \in R : \exists m \in \mathbb{N}\ s.t. f^m \in J\}$.

When $J = \sqrt{J}$, then J is called a radical ideal. Over algebraically closed fields, the *Strong Nullstellensatz* establishes the correspondence between radical ideals and varieties. Over finite fields, it has a special form.

Lemma 3.1 (From [36])**.** For an arbitrary ideal $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, and $J_0 = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$, the ideal $J + J_0$ is radical; i.e. $\sqrt{J + J_0} = J + J_0$.

Theorem 3.2 *(The Strong Nullstellensatz over finite fields (Theorem 3.2 in* [36]*))***.** For any ideal $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, $I(V(J)) = J + J_0$.

Definition 3.7. *Given an ideal $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, the l-th elimination ideal J_l is an ideal in R defined as $J_l = J \cap \mathbb{F}_q[x_{l+1}, \ldots, x_n]$.*

Theorem 3.3 *(Elimination Theorem (from Theorem 2.3.4 [12]))***.** Given an ideal $J \subset R$ and its GB G *w.r.t.* the lexicographical (lex) order on the variables where $x_1 > x_2 > \cdots > x_n$, then for every $0 \leq l \leq n$ we denote by G_l the GB of l-th elimination ideal of J and compute it as:

$$G_l = G \cap \mathbb{F}_q[x_{l+1}, \ldots, x_n].$$

J_l is called the l-th elimination ideal as it eliminates the first l variables from J.

Example 3.1 (from [11])**.** *Consider polynomials $f_1 : x^2 - y - z - 1$, $f_2 : x - y^2 - z - 1$, and $f_3 : x - y - z^2 - 1$ and the ideal $J = \langle f_1, f_2, f_3 \rangle \subset \mathbb{C}[x, y, z]$. $GB(J)$ with lex term order $x > y > z$ equals to $\{g_1 : x - y - z^2 - 1, g_2 : y^2 - y - z^2 - z, g_3 : 2yz^2 - z^4 - z^2, g_4 : z^6 - 4z^4 - 4z^3 - z^2\}$. Then, the GB of 2^{nd} elimination ideal of J is $G_2 = GB(J) \cap \mathbb{C}[z] = \{g_4\}$ and GB of 1^{st} elimination ideal is $G_1 = GB(J) \cap \mathbb{C}[y, z] = \{g_2, g_3, g_4\}$.*

Definition 3.8. Given an ideal $J = \langle f_1, \ldots, f_s \rangle \subset R$ and its variety $V(J) \subset \mathbb{F}_q^n$, the l-th projection of $V(J)$ denoted as $Pr_l(V(J))$ is the mapping

$$Pr_l(V(J)) : \mathbb{F}_q^n \to \mathbb{F}_q^{n-l},\ Pr_l(a_1, \ldots, a_n) = (a_{l+1}, \ldots, a_n),$$

for every $\boldsymbol{a} = (a_1, \ldots, a_n) \in V(J)$.

In a general setting, the projection of a variety is a subset of the variety of an elimination ideal: $Pr_l(V(J)) \subseteq V(J_l)$. However, operating over finite fields, when the ideals contain the vanishing polynomials, then the above set inclusion turns into an equality.

Lemma 3.2 (Lemma 3.4 in [36])**.** Given an ideal $J \subset R$ that contains the vanishing polynomials of the field, then $Pr_l(V(J)) = V(J_l)$, i.e. the l-th projection of the variety of ideal J is equal to the variety of its l-th elimination ideal.

4 Algebraic Miter for Equivalence Checking

Given f_{spec} as the *Spec* polynomial and an *Impl* circuit C, we need to construct an *algebraic miter* between f_{spec} and C. For equivalence checking, we need to prove that the miter is infeasible. Figure 1 depicts how a word-level algebraic miter is setup. Suppose that $A = \{a_0, \ldots, a_{k-1}\}$ and $Z = \{z_0 \ldots, z_{k-1}\}$ denote the k-bit primary inputs and outputs of the finite field circuit, respectively. Then $A = \sum_{i=0}^{k-1} a_i \alpha^i, Z = \sum_{i=0}^{k-1} z_i \alpha^i$ correspond to polynomials that relate the word-level and bit-level inputs and outputs of C. Here α is the primitive element of $\mathbb{F}_{2^k}$. Let Z_S be the word-level output for f_{spec}, which computes some polynomial function $\mathcal{F}(A)$ of A, so that $f_{spec} : Z_S + \mathcal{F}(A)$. The word-level outputs Z, Z_S are mitered to check if for all inputs, $Z \neq Z_S$ is infeasible.

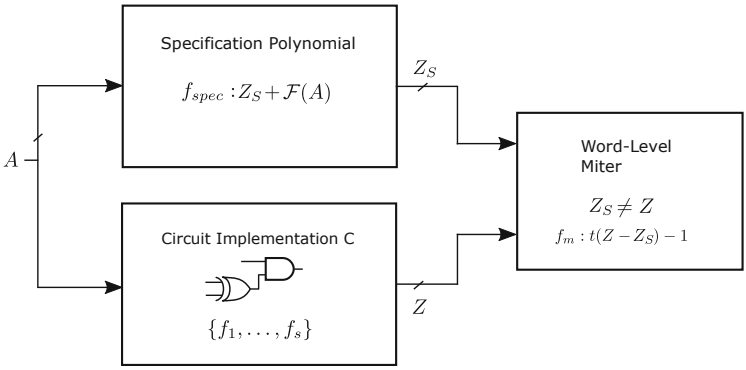

Fig. 1. Word-level miter

The logic gates of C are modeled as the set of polynomials $F = \{f_1, \ldots, f_s\}$ according to Eq. (1). In finite fields, the disequality $Z \neq Z_S$ can be modeled as a single polynomial f_m, called the miter polynomial, where $f_m = t \cdot (Z - Z_S) - 1$, and t is introduced as a free variable. If $Z = Z_S$, then $Z - Z_S = 0$. So $f_m : t \cdot 0 + 1 = 0$ has no solutions (miter is infeasible). Whereas if for some input A, $Z \neq Z_S$, then $Z - Z_S \neq 0$. Let $t^{-1} = (Z - Z_S) \neq 0$. Then $f_m : t \cdot t^{-1} - 1 = 0$ has a solution as (t, t^{-1}) are multiplicative inverses of each other. Thus the miter becomes feasible.

Corresponding to the miter, we construct the ideal $J = \langle f_{spec}, f_1, \ldots, f_s, f_m \rangle$. In our formulation, we need to also include the ideal J_0 corresponding to the vanishing polynomials in variables Z, Z_s, A, t, and x_i; here Z, Z_s, A, t are the word-level variables that take values in $\mathbb{F}_{2^k}$, and x_i corresponds to the bit level (Boolean) variables in the miter. In fact, it was shown in [7] that in J_0 it is sufficient to include vanishing polynomials for only the primary input bits ($x_i \in X_{PI}$). Therefore, $J_0 = \langle x_i^2 - x_i : x_i \in X_{PI} \rangle$.

In this way, equivalence checking using the algebraic model is solved as follows: Construct an ideal $J = \langle f_{spec}, f_1, \ldots, f_s, f_m \rangle$, as described above. Add to it the ideal $J_0 = \langle x_i^2 - x_i : x_i \in X_{PI} \rangle$. Determine if the variety $V(J + J_0) = \emptyset$, i.e. if *reduced* $GB(J + J_0) = \{1\}$? If $V(J + J_0) = \emptyset$, the miter is infeasible, and

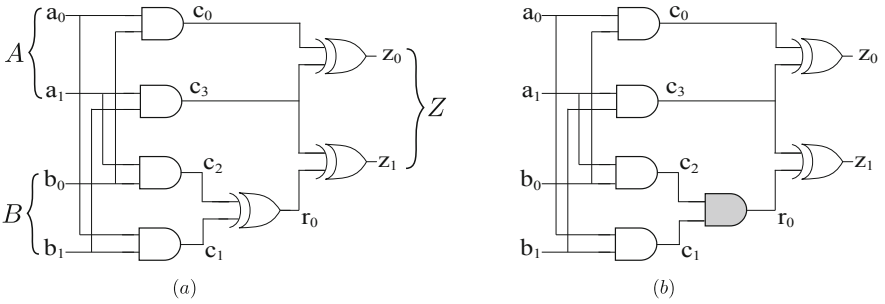

Fig. 2. Correct (a) and buggy (b) 2-bit modulo multiplier circuit implementations

C implements f_{spec}. If $V(J + J_0) \neq \emptyset$, the miter is feasible, and there exists a bug in the design.

Example 4.1. *Consider a modulo multiplier with output Z and inputs A, B. The Spec polynomial is given as $f_{spec} : Z + A \cdot B \pmod{P(X)}$, where $P(X)$ is a primitive polynomial of the field. An implementation of such a multiplier with operand (Z, A, B) bit-width = 2 is shown in Fig. 2(a).*

Now let's say that the designer has introduced a bug, and the XOR gate with output net r_0 has been replaced with an AND gate in the actual implementation in the circuit of Fig. 2(b). The polynomials for the gates of the correct circuit implementation are,

$$f_1 : c_0 + a_0 \cdot b_0, \quad f_2 : c_1 + a_0 \cdot b_1, \quad f_3 : c_2 + a_1 \cdot b_0, \quad f_4 : c_3 + a_1 \cdot b_1,$$
$$f_5 : r_0 + c_1 + c_2, \quad f_6 : z_0 + c_0 + c_3, \quad f_7 : z_1 + r_0 + c_3,$$

whereas for the buggy implementation, the polynomial f_5 is $f_5' : r_0 + c_1 c_2$. The problem is modeled over $\mathbb{F}_4$ and let α be a primitive element of $\mathbb{F}_4$. The word-level polynomials are $f_8 : Z + z_0 + z_1\alpha$, $f_9 : A + a_0 + a_1\alpha$, and $f_{10} : B + b_0 + b_1\alpha$. The specification polynomial is $f_{spec} : Z_s + AB$. We create a miter polynomial against this specification as $f_m : t(Z - Z_s) - 1$.

To perform equivalence checking of the correct implementation and the specification polynomial, we construct ideal $J = \langle f_{spec}, f_1, \ldots, f_5, \ldots, f_{10}, f_m \rangle$. Computing GB of $J + J_0$ (J_0 is the ideal of vanishing polynomials) results in $\{1\}$, implying the the circuit in Fig. 2(a) is equivalent to the specification. However, computing GB of the ideal $J' + J_0$ where $J' = \langle f_{spec}, f_1, \ldots, f_5', \ldots, f_{10}, f_m \rangle$ results in a set of polynomials $G = \{g_1, \ldots, g_t\} \neq \{1\}$, implying the presence of a bug(s) in the design.

5 Formulating the Rectification Check

Equivalence checking is performed between the *Spec* and *Impl* circuit C, and it reveals the presence of a bug in the design. Post-verification, we assume that error diagnosis has been performed, and a set of nets X has been identified as potential single-fix rectifiable locations. While the nets in X might be target nets for single-fix, the circuit may or may not be rectifiable at any $x_i \in X$. We have to first ascertain that the circuit is indeed single-fix rectifiable at some $x_i \in X$, and subsequently compute a rectification function $U(X_{PI})$, so that $x_i = U(X_{PI})$ rectifies the circuit at that net.

5.1 Single Fix Rectification

Using the Weak Nullstellensatz (Theorem 3.1), we formulate the test for rectifiability of C at a net x_i in the circuit. For this purpose, we state and prove the following result, which is utilized later.

Proposition 5.1. Given two ideals J_1 and J_2 over some finite field such that $\overline{V(J_1)} \cap V(J_2) = \emptyset$, there exists a polynomial U which satisfies $V(J_1) \subseteq V(U) \subseteq \overline{V(J_2)}$.

Proof. Over finite fields $\mathbb{F}_q$, $V(J_1)$ and $V(J_2)$ are finite sets of points. Every finite set of points is a variety of some ideal. Therefore, given $\overline{V(J_1)} \cap V(J_2) = \emptyset$, there exists a set of points (a variety) which contains $V(J_1)$, and which does not intersect with $V(J_2)$. Let this variety be denoted by $V(J_I)$, where J_I is the corresponding ideal. Then $V(J_1) \subseteq V(J_I) \subseteq \overline{V(J_2)}$. In addition, we can construct a polynomial U that vanishes exactly on the points in $V(J_I)$ by means of the Lagrange's interpolation formula. □

We now present the result that ascertains the circuit's rectifiability at a target net. Let the net $x_i \in X$ (*i.e.* i^{th} gate) be the rectification target, and a possible rectification function be $x_i = U(X_{PI})$. Then the i^{th} gate is represented by a polynomial $f_i : x_i + U(X_{PI})$. Consider the ideal J corresponding to the algebraic miter – the polynomials $f_1, \ldots, f_i, \ldots, f_s$ representing the gates of the circuit, the specification polynomial f_{spec}, and the miter polynomial f_m:

$$J = \langle f_{spec}, f_1, \ldots, f_i : x_i + U(X_{PI}), \ldots, f_s, f_m \rangle.$$

The following theorem checks whether the circuit is indeed single-fix rectifiable at the net x_i.

Theorem 5.1. Construct two ideals:

- $J_L = \langle f_{spec}, f_1, \ldots, f_i : x_i + 1, \ldots, f_s, f_m \rangle$ where $f_i : x_i + U(X_{PI})$ in J is replaced with $f_i : x_i + 1$.
- $J_H = \langle f_{spec}, f_1, \ldots, f_i : x_i, \ldots, f_s, f_m \rangle$ where $f_i : x_i + U(X_{PI})$ in J is replaced with $f_i : x_i$.

Compute $E_L = (J_L + J_0) \cap \mathbb{F}_{2^k}[X_{PI}]$ and $E_H = (J_H + J_0) \cap \mathbb{F}_{2^k}[X_{PI}]$ to be the respective elimination ideals, where all the non-primary input variables have been eliminated. Then the circuit can be single-fix rectified at net x_i with the polynomial function $f_i : x_i + U(X_{PI})$ to implement the specification iff $1 \in E_L + E_H$.

Proof. We will first prove the *if* case of the theorem. Assume $1 \in E_L + E_H$, or equivalently $V_{X_{PI}}(E_L) \cap V_{X_{PI}}(E_H) = \emptyset$. The subscript X_{PI} in $V_{X_{PI}}$ denotes that the variety is being considered over X_{PI} variables, as the non-primary inputs have been eliminated from E_L and E_H. Using Proposition 5.1, we can find a polynomial $U(X_{PI})$ such that,

$$V_{X_{PI}}(E_L) \subseteq V_{X_{PI}}(U(X_{PI})) \subseteq \overline{V_{X_{PI}}(E_H)}. \tag{3}$$

Note, however, that since $V_{X_{PI}}(E_L), V_{X_{PI}}(E_H)$ are considered over only primary input bits, they contain points from $\mathbb{F}_2^{|X_{PI}|}$. Therefore, there exists a polynomial $U(X_{PI})$ as in Eq. (3) with coefficients only in $\mathbb{F}_2$.

Let us consider a point $\boldsymbol{p}$ in $V(J)$. Point $\boldsymbol{p}$ is an assignment to every variable in J such that all the generators of J are satisfied. We denote by $\boldsymbol{a}$, the projection of $\boldsymbol{p}$ on the primary inputs (i.e. the primary input assignments under $\boldsymbol{p}$). There are only two possibilities for $U(X_{PI})$,

1. $U(\boldsymbol{a}) = 1$, or in other words $\boldsymbol{a} \notin V_{X_{PI}}(U(X_{PI}))$. It also implies that the value of x_i under $\boldsymbol{p}$ must be 1 because $x_i + U(X_{PI}) = 0$ needs to be satisfied. Since the generator f_i of J_L also forces x_i to be 1 and all other generators are exactly the same as those of J, $\boldsymbol{p}$ is also a point in $V(J_L)$. Moreover, E_L is the elimination ideal of J_L, and therefore, $\boldsymbol{a} \in V_{X_{PI}}(E_L)$. But this a contradiction to our assumption that $V_{X_{PI}}(E_L) \subseteq V_{X_{PI}}(U(X_{PI}))$ and such a point $\boldsymbol{a}$ (and $\boldsymbol{p}$) does not exist.
2. $U(\boldsymbol{a}) = 0$, or in other words $\boldsymbol{a} \in V_{X_{PI}}(U(X_{PI}))$. Using similar argument as the previous case, we can show that $\boldsymbol{a} \in V_{X_{PI}}(E_H)$. This is again a contradiction to our assumption $V_{X_{PI}}(U(X_{PI})) \subseteq \overline{V_{X_{PI}}(E_H)}$.

In conclusion, there exists no point in $V(J)$ (or the miter is infeasible) when $U(X_{PI})$ satisfies Eq. 3, and therefore, circuit can be rectified at x_i.

Now we will prove the *only if* direction of the proof. We show that if $1 \notin E_L + E_H$, then there exists no polynomial $U(X_{PI})$ that can rectify the circuit. If $1 \notin E_L + E_H$, then E_L and E_H have a common zero. Let $\boldsymbol{a}$ be a point in $V_{X_{PI}}(E_L)$ and $V_{X_{PI}}(E_H)$. This point can be extended to some points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ in $V(J_L)$ and $V(J_H)$, respectively. Notice that in point $\boldsymbol{p'}$ the value of x_i will be 1, and in $\boldsymbol{p''}$ x_i will be 0. Any polynomial $U(X_{PI})$ will either evaluate to 0 or 1 for the assignment $\boldsymbol{a}$ to the primary inputs. If it evaluates to 1, then we can say that $\boldsymbol{p'}$ is in $V(J)$ as f_i in J forces $x_i = 1$ and all other generators of J and J_L are same. This implies that $f_m(\boldsymbol{p'}) = 0$ (f_m: miter polynomial is feasible) and this choice of $U(X_{PI})$ will not rectify the circuit. If $U(X_{PI})$ evaluates to 0, then $\boldsymbol{p''}$ is a point in $V(J)$.

Therefore, no choice of $U(X_{PI})$ can rectify the circuit if $1 \notin E_L + E_H$. □

Example 5.1. *Consider the buggy modulo multiplier circuit of Fig. 2(b) (reproduced in Fig. 3), where the gate output r_0 should have been the output of an XOR gate, but an AND gate is incorrectly implemented. We apply Theorem 5.1 to check for single-fix rectifiability at r_0. The polynomials for the gates of the correct circuit implementation are,*

$$f_1 : c_0 + a_0 \cdot b_0, \quad f_2 : c_1 + a_0 \cdot b_1, \quad f_3 : c_2 + a_1 \cdot b_0, \quad f_4 : c_3 + a_1 \cdot b_1,$$
$$f_5 : r_0 + c_1 + c_2, \quad f_6 : z_0 + c_0 + c_3, \quad f_7 : z_1 + r_0 + c_3$$

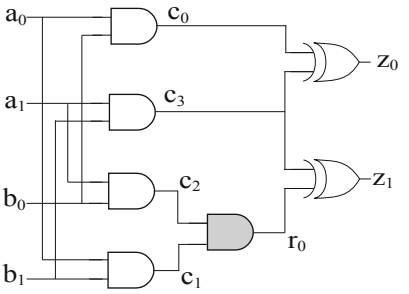

Fig. 3. A buggy 2-bit modulo multiplier circuit

The problem is modeled over $\mathbb{F}_4$ *and let* α *be a primitive element of* $\mathbb{F}_4$*. The word-level polynomials are* $f_8 : Z + z_0 + z_1\alpha$*,* $f_9 : A + a_0 + a_1\alpha$*, and* $f_{10} : B + b_0 + b_1\alpha$*. The specification polynomial is* $f_{spec} : Z_s + AB$*. We create a miter polynomial against this specification as* $f_m : t(Z - Z_s) - 1$*.*

The ideals J_L *and* J_H *are constructed as:*

$$J_L = \langle f_{spec}, f_1, \ldots, f_4, r_0 + 1, f_6, \ldots, f_{10}, f_m \rangle$$
$$J_H = \langle f_{spec}, f_1, \ldots, f_4, r_0, f_6, \ldots, f_{10}, f_m \rangle$$

The ideal J_0 *is:*

$$J_0 = \langle b_1^2 - b_1, b_0^2 - b_0, a_1^2 - a_1, a_0^2 - a_0 \rangle,$$

and the corresponding ideals E_L *and* E_H *are computed to be:*

$$E_L = \langle a_0b_1 + a_1b_0, a_1b_0b_1 + a_1b_0, a_0a_1b_0 + a_1b_0 \rangle$$
$$E_H = \langle b_0b_1 + b_0 + b_1 + 1, a_1b_1 + a_1 + b_1 + 1, a_0b_1 + a_1b_0 + 1,$$
$$a_0b_0 + a_0 + b_0 + 1, a_0a_1 + a_0 + a_1 + 1 \rangle$$

Computing a Gröbner basis G *of* $E_L + E_H$ *results in* $G = \{1\}$*. Therefore, we can rectify this circuit at* r_0*.*

On the other hand, if we apply the rectification theorem at net c_2*, the respective ideals* E_L *and* E_H *are as follows,*

$$E_L = \langle a_0^2 + a_0, a_1^2 + a_1, b_0^2 + b_0, b_1^2 + b_1, a_1b_0 + b_0, a_0b_1b_0 + a_0b_1 + a_0b_0 + a_0, a_0a_1 + a_0 \rangle$$
$$E_H = \langle a_0^2 + a_0, b_0^2 + b_0, b_1^2 + b_1, b_1b_0 + b_1 + b_0 + 1, a_1 + 1, a_0b_0 + a_0 + b_0 + 1 \rangle$$

When we compute $G = GB(E_L + E_H)$*, we obtain* $G \neq \{1\}$ *indicating that single-fix rectification is not possible at net* c_2*, for the given bug.*

6 Craig Interpolants in Finite Fields

Once it is ascertained that a net x_i admits single-fix rectification, the subsequent task is to compute a rectification polynomial function $x_i = U(X_{PI})$ in terms of the primary inputs of the circuit. In this section, we describe how such a rectification polynomial function can be computed. For this purpose, we introduce the concept of Craig interpolants using algebraic geometry in finite fields.

We describe the setup for Craig interpolation in the ring $R = \mathbb{F}_q[x_1, \ldots, x_n]$. Partition the variables $\{x_1, \ldots, x_n\}$ into disjoint subsets A, B, C. We are given two ideals $J_A \subset \mathbb{F}_q[A, C]$, $J_B \subset \mathbb{F}_q[B, C]$ such that the C-variables are common to the generators of both J_A, J_B. *From here on, we will assume that all ideals include the corresponding vanishing polynomials.* For example, generators of J_A include $\boldsymbol{A^q - A, C^q - C}$, where $\boldsymbol{A^q - A} = \{x_i^q - x_i : x_i \in A\}$, and so on. Then these ideals become radicals and we can apply Lemmas 3.1 and 3.2. We use $V_{A,C}(J_A)$ to denote the variety of J_A over the $\mathbb{F}_q$-space spanned by A and C variables, i.e. $V_{A,C}(J_A) \subset \mathbb{F}_q^A \times \mathbb{F}_q^C$. Similarly, $V_{B,C}(J_B) \subset \mathbb{F}_q^B \times \mathbb{F}_q^C$.

Now let $J = J_A + J_B \subseteq \mathbb{F}_q[A, B, C]$, and suppose that it is found by application of the Weak Nullstellensatz (Theorem 3.1) that $V_{A,B,C}(J) = \emptyset$. When we compare the varieties of J_A and J_B, then we can consider the varieties in $\mathbb{F}_q^A \times \mathbb{F}_q^B \times \mathbb{F}_q^C$, as $V_{A,B,C}(J_A) = V_{A,C}(J_A) \times \mathbb{F}_q^B \subset \mathbb{F}_q^A \times \mathbb{F}_q^B \times \mathbb{F}_q^C$. With this setup, we define the interpolants as follows.

Definition 6.1 (*Interpolants in finite fields*)**.** Given two ideals $J_A \subset \mathbb{F}_q[A, C]$ and $J_B \subset \mathbb{F}_q[B, C]$ where A, B, C denote the three disjoint sets of variables such that $V_{A,B,C}(J_A) \cap V_{A,B,C}(J_B) = \emptyset$. Then there exists an ideal J_I satisfying the following properties:

1. $V_{A,B,C}(J_I) \supseteq V_{A,B,C}(J_A)$
2. $V_{A,B,C}(J_I) \cap V_{A,B,C}(J_B) = \emptyset$
3. Generators of J_I contain only the C-variables; or $J_I \subseteq \mathbb{F}_q[C]$.

We call $V_{A,B,C}(J_I)$ the **interpolant** in finite fields of the pair $(V_{A,B,C}(J_A), V_{A,B,C}(J_B))$, and the corresponding ideal J_I the **ideal-interpolant**.

As the generators of J_I contain only the C-variables, the interpolant $V_{A,B,C}(J_I)$ is of the form $V_{A,B,C}(J_I) = \mathbb{F}_q^A \times \mathbb{F}_q^B \times V_C(J_I)$. Therefore, the subscripts A, B for the interpolant $V_{A,B,C}(J_I)$ may be dropped for the ease of readability.

Example 6.1 Consider the ring $R = \mathbb{F}_2[a, b, c, d, e]$, partition the variables as $A = \{a\}, B = \{e\}, C = \{b, c, d\}$. Let ideals

$$J_A = \langle ab, bd, bc + c, cd, bd + b + d + 1 \rangle + J_{0,A,C}$$
$$J_B = \langle b, d, ec + e + c + 1, ec \rangle + J_{0,B,C}$$

where $J_{0,A,C}$ and $J_{0,B,C}$ are the corresponding ideals of vanishing polynomials. Then,

$$V_{A,B,C}(J_A) = \mathbb{F}_q^B \times V_{A,C}(J_A) =$$
$$(abcde) : \{01000, 00010, 01100, 10010, 01001, 00011, 01101, 10011\}$$
$$V_{A,B,C}(J_B) = \mathbb{F}_q^A \times V_{B,C}(J_B) =$$
$$(abcde) : \{00001, 00100, 10001, 10100\}$$

Ideals J_A, J_B have no common zeros as $V_{A,B,C}(J_A) \cap V_{A,B,C}(J_B) = \emptyset$. The pair (J_A, J_B) admits a total of 8 interpolants:

1. $V(J_S) = (bcd) : \{001, 100, 110\}$
   $J_S = \langle cd, b + d + 1\rangle$
2. $V_C(J_1) = (bcd) : \{001, 100, 110, 101\}$
   $J_1 = \langle cd, bd + b + d + 1, bc + cd + c\rangle$
3. $V_C(J_2) = (bcd) : \{001, 100, 110, 011\}$
   $J_2 = \langle b + d + 1\rangle$
4. $V_C(J_3) = (bcd) : \{001, 100, 110, 111\}$
   $J_3 = \langle b + cd + d + 1\rangle$
5. $V_C(J_4) = (bcd) : \{001, 100, 110, 011, 111\}$
   $J_4 = \langle bd + b + d + 1, bc + b + cd + c + d + 1\rangle$
6. $V_C(J_5) = (bcd) : \{001, 100, 110, 101, 111\}$
   $J_5 = \langle bc + c, bd + b + d + 1\rangle$
7. $V_C(J_6) = (bcd) : \{001, 100, 110, 101, 011\}$
   $J_6 = \langle bd + b + d + 1, bc + cd + c\rangle$
8. $V_C(J_L) = (bcd) :$
   $\{001, 011, 100, 101, 110, 111\}$
   $J_L = \langle bd + b + d + 1\rangle$.

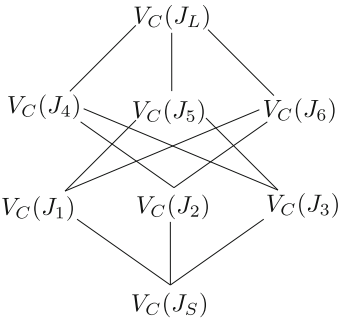

Fig. 4. The Interpolant lattice for Example 6.1

It is easy to check that all $V(J_I)$ satisfy the 3 conditions of Definition 6.1. Note also that $V(J_S)$ is the smallest interpolant, contained in every other interpolant. Likewise, $V(J_L)$ contains all other interpolants and it is the largest. The other containment relationships are shown in the corresponding interpolant lattice in Fig. 4; $V_C(J_1) \subset V_C(J_5), V_C(J_1) \subset V_C(J_6)$, etc.

Theorem 6.1 (Existence of Craig Interpolants)**.** An ideal-interpolant J_I, and correspondingly the interpolant $V_{A,B,C}(J_I)$, as given in Definition 6.1, always exists.

Proof. Consider the elimination ideal $J_I = J_A \cap \mathbb{F}_q[C]$. We show J_I satisfies the three conditions for the interpolant.

Condition 1: $V_{A,B,C}(J_I) \supseteq V_{A,B,C}(J_A)$. This condition is trivially satisfied due to construction of elimination ideals. As $J_I \subseteq J_A$, $V_{A,B,C}(J_I) \supseteq V_{A,B,C}(J_A)$.
Condition 2: $V_{A,B,C}(J_I) \cap V_{A,B,C}(J_B) = \emptyset$. This condition can be equivalently stated as $V_{B,C}(J_I) \cap V_{B,C}(J_B) = \emptyset$ as neither J_I nor J_B contain any variables from the set A. We prove this condition by contradiction. Let's assume that

there exists a common point $(\mathbf{b}, \mathbf{c})$ in $V_{B,C}(J_I)$ and $V_{B,C}(J_B)$. We know that the projection of the variety $Pr_A(V_{A,C}(J_A))$ is equal to the variety of the elimination ideal $V_C(J_I)$, where $J_I = J_A \cap \mathbb{F}_q[C]$, due to Lemma 3.2. Therefore, the point $(\mathbf{c})$ in the variety of J_I can be extended to a point $(\mathbf{a}, \mathbf{c})$ in the variety of J_A. This implies that the ideals J_A and J_B vanish at $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. This is a contradiction to our initial assumption that the intersection of the varieties of J_A and J_B is empty. Thus J_I, J_B have no common zeros.
Condition 3: The generators of J_I contain only the C-variables. This condition is trivially satisfied as J_I is the elimination ideal obtained by eliminating A-variables in J_A. □

The above theorem not only proves the existence of an interpolant, but also gives a procedure to construct its ideal: $J_I = J_A \cap \mathbb{F}_q[C]$. In other words, compute a reduced Gröbner basis G of J_A w.r.t. the elimination order $A > B > C$ and take $G_I = G \cap \mathbb{F}_q[C]$. Then G_I gives the generators for the ideal-interpolant J_I.

Example 6.2. *The elimination ideal J_I computed for J_A from Example 6.1 is $J_I = J_S = \langle cd, b + d + 1 \rangle$ with variety $V_C(J_I) = (bcd) : \{001, 100, 110\}$. This variety over the variable set A and C is $V_{A,C}(J_I) = (abcd) : \{0001, 0100, 0110, 1001, 1100, 1110\}$, and it contains $V_{A,C}(J_A)$. Moreover, $V_{A,B,C}(J_I)$ also has an empty intersection with $V_{A,B,C}(J_B)$.*

Theorem 6.2 (Smallest interpolant)**.** The interpolant $V_{A,B,C}(J_S)$ corresponding to the ideal $J_S = J_A \cap \mathbb{F}_q[C]$ is the smallest interpolant.

Proof. Let $J_I \subseteq \mathbb{F}_q[C]$ be any another ideal-interpolant $\neq J_S$. We show that $V_C(J_S) \subseteq V_C(J_I)$. For $V_C(J_I)$ to be an interpolant it must satisfy

$$V_{A,B,C}(J_A) \subseteq V_{A,B,C}(J_I)$$

which, due to Theorem 3.2, is equivalent to

$$I(V_{A,B,C}(J_A)) \supseteq I(V_{A,B,C}(J_I)) \implies J_A \supseteq J_I$$

As the generators of J_I only contain polynomials in C-variables, this relation also holds for the following

$$J_A \cap \mathbb{F}_q[C] \supseteq J_I \implies J_S \supseteq J_I \implies V_C(J_S) \subseteq V_C(J_I).$$ □

6.1 Computing a Rectification Function from Craig Interpolants

Back to our formulation of single-fix rectification, from Theorem 5.1 we have $1 \in E_L + E_H$ or $V(E_L) \cap V(E_H) = \emptyset$. Therefore, we can consider the pair (E_L, E_H) for Craig interpolation. In other words, based on the notation from Definition 6.1, $J_A = E_L$ and $J_B = E_H$. Moreover, E_L and E_H are elimination ideals containing only X_{PI} variables. As a result, the partitioned set of variables for Craig interpolation A, B, and C all correspond to primary inputs. Furthermore, we want to compute an ideal J_I in X_{PI} such that

$V_{X_{PI}}(E_L) \subseteq V_{X_{PI}}(J_I)$ and $V_{X_{PI}}(J_I) \cap V_{X_{PI}}(E_H) = \emptyset$. The *smallest ideal-interpolant* $J_I = E_L \cap \mathbb{F}_{2^k}[X_{PI}] = E_L$ itself. Therefore, we use E_L to compute the correction function $U(X_{PI})$.

Obtaining $U(X_{PI})$ from E_L: In finite fields, given an ideal J, it always possible to find a polynomial U such that $V(U) = V(J)$. The reason is that every ideal in a finite field has a finite variety, and a polynomial with those points as its roots can always be constructed using the Lagrangian interpolation formula. We construct the rectification polynomial U from the ideal-interpolant E_L as shown below, such that $V(E_L) = V(U)$.

Let the generators of E_L be denoted by $g_1, \ldots, g_t$. We can compute U as,

$$U = (1 + g_1)(1 + g_2) \cdots (1 + g_t) + 1 \tag{4}$$

It is easy to assert that $V(U) = V(E_L)$. Consider a point $\boldsymbol{a}$ in $V(E_L)$. As all of $g_1, \ldots, g_t$ vanish $(= 0)$ at $\boldsymbol{a}$,

$$\begin{aligned} U(\boldsymbol{a}) &= (1 + g_1(\boldsymbol{a}))(1 + g_2(\boldsymbol{a})) \cdots (1 + g_t(\boldsymbol{a})) + 1 \\ &= (1 + 0)(1 + 0) \cdots (1 + 0) + 1 = 0 \end{aligned}$$

Conversely, for a point $\boldsymbol{a'} \notin V(E_L)$, at least one of $g_1, \ldots, g_t$ will evaluate to 1. Without loss of generality, if g_1 evaluates to 1 at $\boldsymbol{a'}$, then $U = (1 + 1)(1 + 0) \cdots (1 + 0) + 1 \neq 0$.

Using Eq. (4), a recursive procedure is derived to compute U, and it is depicted in Algorithm 2. At every recursive step, we also reduce the intermediate results by $\pmod{J_0}$ (line 7) so as to avoid terms of high degree. In this fashion, from the ideal-interpolant E_L, we compute the single-fix rectification polynomial function $U(X_{PI})$, and synthesize a sub-circuit at net x_i such that $x_i = U(X_{PI})$ rectifies the circuit.

Algorithm 2. Compute U from J such that $V(U) = V(J)$

```
1: U = compute_U(J, J_0) + 1
2: procedure compute_U(J, J_0)   /*J = ⟨g_1, ..., g_t⟩*/
3:     if size(J) = 1 then
4:         return (1 + J[1])
5:     subsetJ = {J[1], J[2], ..., J[size(J) − 1]}
6:     poly S_1 = compute_U(subsetJ, J_0)
7:     Perform S_1 · J[size(J)] −J_0→+ S_2
8:     return S_1 + S_2
```

Example 6.3. *Example 5.1 showed that the buggy circuit of Fig. 3 can be rectified at net r_0. This rectification check required the computation of the (Gröbner basis of) ideal E_L. Using Algorithm 2, we compute $U(X_{PI})$ from E_L to be $a_0b_1 + a_1b_0$, and the rectification polynomial as $r_0 + a_0b_1 + a_1b_0$. This can be synthesized into a sub-circuit as $r_0 = (a_0 \wedge b_1) \oplus (a_1 \wedge b_0)$, by replacing the modulo 2 product and sum in the polynomial with the Boolean AND and XOR operators, respectively.*

7 Efficient Gröbner Basis Computations for E_L and E_H

The proposed rectification approach requires the computation of (generators of) elimination ideals E_L and E_H. This is achieved by computing a Gröbner basis each for $GB(J_L + J_0) \cap \mathbb{F}_{2^k}[X_{PI}]$ and $GB(J_H + J_0) \cap \mathbb{F}_{2^k}[X_{PI}]$, respectively. The rectification polynomial function $x_i = U(X_{PI})$ is subsequently derived from the generators of E_L. As the generators of J_L and J_H comprise polynomials derived from the entire circuit, these GB-computations become infeasible for larger circuits due to its high complexity. In [37], it was shown that the time and space complexity of computing $GB(J + J_0)$ over $\mathbb{F}_q[x_1, \ldots, x_n]$ is bounded by $q^{O(n)}$. In the context of our work, as $q = 2^k$ where k is the operand-width, and n the number of variables (nets) in the miter, we have to overcome this complexity to make our approach practical for large circuits.

Prior work [8] has shown that the GB-computation can be significantly improved when the polynomials are derived from circuits. By analyzing the topology of the given circuit, a specialized term order can be derived that can significantly reduce the number of *Spoly* computations in the GB-algorithm. We present a similar approach to improve the GB-computation for ideals E_L, E_H.

Lemma 7.1 (Product Criterion [40])**.** For two polynomials f_i, f_j in any polynomial ring R, if the equality $lm(f_i) \cdot lm(f_j) = LCM(lm(f_i), lm(f_j))$ holds, i.e. if $lm(f_i)$ and $lm(f_j)$ are relatively prime, then $Spoly(f_i, f_j) \xrightarrow{G}_+ 0$.

Buchberger's algorithm therefore does not pair those polynomials f_i, f_j (Algorithm 1, line 4) whose leading monomials are relatively prime, as they do not produce any new information in the basis. Moreover, based on the above criterion, when the leading monomials of *all polynomials in the basis* $F = \{f_1, \ldots, f_s\}$ are relatively prime, then all $Spoly(f_i, f_j) \xrightarrow{G}_+ 0$. As no new polynomials are generated in Buchberger's algorithm, F already constitutes a Gröbner basis ($F = GB(J)$). For a combinational circuit C, a specialized term order $>$ can always be derived by analyzing the circuit topology which ensures such a property [4,7]:

Proposition 7.1 (From [7])**.** Let C be an arbitrary combinational circuit. Let $\{x_1, \ldots, x_n\}$ 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 $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a *lex* term order $>$ to represent each gate as a polynomial f_i, s.t. $f_i = x_i + tail(f_i)$. Then the set of all polynomials $\{f_1, \ldots, f_s\}$ forms a Gröbner basis G, as $lt(f_i) = x_i$ and $lt(f_j) = x_j$ for $i \neq j$ are relatively prime. This term order $>$ is called the **Reverse Topological Term Order (RTTO)**.

RTTO ensures that the set of all polynomials $\{f_1, \ldots, f_s\}$ of the given circuit C have relatively prime leading terms. However, the model of the algebraic miter (Fig. 1, with the *Spec* and the miter polynomial, in addition to the given circuit) is such that under RTTO $>$, not all polynomials have relatively prime leading

terms. However, we show that imposition of RTTO on the miter still significantly reduces the amount of computation required for Gröbner bases. We demonstrate the technique on the GB computation for the ideal $J_L + J_0$ (analogously also for $J_H + J_0$), corresponding to the miter, as per Theorem 5.1.

Given the word-level miter of Fig. 1, impose a lexicographic (*lex*) monomial order on the ring R, with the following variable order:

$$t > Z > Z_S > A > \textit{nets of } C \textit{ in RTTO order} > \textit{Primary input variables} \quad (5)$$

Here t is the free variable used in the miter polynomial, and Z, Z_s are the word-level outputs of *Impl* and *Spec*, respectively, and A is the word-level input. Corresponding to the circuit in Fig. 3 (Example 5.1), we use a *lex* term order with variable order:

$$t > Z > Z_S > A > B > z_1 > z_0 > r_0 > c_0 > c_1 > c_2 > c_3 > b_1 > b_0 > a_1 > a_0 \quad (6)$$

The polynomials $\{f_1, \ldots, f_{10}, f_{spec}, f_m\}$ in Example 5.1 are already written according to the term order of Eq. (6). Note also that the leading terms of the generators of the ideal J_L are the same as the leading terms of polynomials in $\{f_1, \ldots, f_{10}, f_{spec}, f_m\}$. From among these, the only pair of polynomials that *do not have relatively prime leading* terms are f_8 and f_m. This condition also holds when considering the ideal $J_L + J_0$ (instead of only J_L) as J_0 is composed of only bit-level primary input variables.

In general, modeling an algebraic miter with RTTO $>$ will ensure that we have *exactly one pair of polynomials with leading monomials that are not relatively prime*. This pair includes: (i) the miter polynomial $f_m : tZ - tZ_s - 1$, with $lm(f_m) = tZ$; and (ii) the polynomial (hereafter denoted by f_o) that relates the word-level and bit-level variables of the circuit, $f_o : Z + z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1}$, with $lm(f_o) = Z$. Therefore, in the first iteration of Algorithm 1 for computing $GB(J_L + J_0)$, the only critical pair to compute is $Spoly(f_m, f_o)$, as all other pairs reduce to 0, due to Lemma 7.1. Moreover, computing $Spoly(f_m, f_o)$ results in $Spoly(f_m, f_o) = t(Z_S + z_0 + \cdots + z_{k-1}\alpha^{k-1}) + 1$. Once again, RTTO $>$ ensures the following:

Lemma 7.2. $Spoly(f_m, f_o) \xrightarrow{J_L + J_0}_+ h = t \cdot r + 1$, where r is a polynomial in bit-level primary input variables.

Proof. Consider the polynomial reduction of $Spoly(f_m, f_o) \xrightarrow{J_L + J_0}_+ h$:

$$t(Z_S + z_0 + \cdots + z_{k-1}\alpha^{k-1}) + 1 \xrightarrow{f_{spec}}_+$$

where $f_{spec} = Z_S + \mathcal{F}(A)$. The remainder for this reduction will be

$$t(\mathcal{F}(A) + z_0 + \cdots + z_{k-1}\alpha^{k-1}) + 1,$$

where $\mathcal{F}(A)$ is the polynomial specification in word-level input variable(s). This remainder is then reduced by the polynomial relating the word-level and bit-level primary input variables, i.e. by $A + a_0 + \cdots + a_{k-1}\alpha^{k-1}$. The subsequent remainder is

$$\begin{aligned} t(\mathcal{F}(A) + z_0 + \cdots + z_{k-1}\alpha^{k-1}) + 1 \xrightarrow{A+a_0+\cdots+a_{k-1}\alpha^{k-1}}_{+} \\ t(z_0 + \cdots + z_{k-1}\alpha^{k-1} + \mathcal{G}(a_0, \cdots, a_{k-1})) + 1, \end{aligned} \tag{7}$$

where the word-level specification polynomial $\mathcal{F}(A)$ gets reduced to a polynomial expression $\mathcal{G}(a_0, \ldots, a_{k-1})$ in primary input bits. Due to RTTO $>$, subsequent divisions of the above remainder in Eq. (7) by $\{f_1, \ldots, f_s\}$ will successively cancel the terms in variables $z_i, i = 0, \ldots, k-1$, and express them in terms of the primary input bits. Since primary input bits are last in RTTO $>$, they never appear as leading terms in any of the polynomials in J_L; so the terms in primary input bits cannot be canceled. As a result, after complete reduction of $Spoly(f_m, f_o)$ by $J_L + J_0$, the remainder will be a polynomial expression of the form $Spoly(f_m, f_o) \xrightarrow{J_L+J_0}_{+} h = t \cdot r + 1$, where r is a polynomial only in bit-level primary input variables. □

Coming back to the computation $GB(J_L + J_0)$, the polynomial h is now added to the current basis, i.e. $G = \{J_L + J_0\} \cup \{h\}$ in Buchberger's algorithm (Line 7 in Algorithm 1). This polynomial h now needs to be paired with other polynomials in the basis. There are only two sets of possibilities for subsequent critical pairings: (i) the pair $Spoly(f_m, h)$; and (ii) to pair h with corresponding vanishing polynomials from the ideal J_0. For all other polynomials $f_i \in \{f_1, \ldots, f_s\}$, $lm(h)$ and $lm(f_i)$ have relatively prime leading terms, so $Spoly(h, f_i)_{i=1,\ldots,s} \xrightarrow{J_L+J_0}_{+} 0$; so the pairs (h, f_i) need not be considered in $GB(J_L + J_0)$. We now show that $Spoly(f_m, h) \xrightarrow{G=\{J_L+J_0\}\cup\{h\}}_{+} 0$, so the pair (f_m, h) also need not be considered.

From Lemma 7.2 and its proof, we have that $h = t \cdot r + 1$ and $Z + Z_S \xrightarrow{G=\{J_L+J_0\}}_{+} = r$, with r composed of primary input bits. Let $r = e + r'$, where $e = lt(r)$ is the leading term and $r' = r - e$ is $tail(r)$, both expressed in primary input bits. With this notation, $h = te + tr' + 1$ and $lt(h) = te$. The LCM L of leading monomials of f_m and h is $L = LCM(lm(f_m), lm(h)) = LCM(tZ, te) = tZe$. Consider the computation $Spoly(f_m, h)$:

$$\begin{aligned} Spoly(f_m, h) &= \frac{L}{lt(f_m)} \cdot f_m - \frac{L}{lt(h)} \cdot h \\ &= ef_m + Zh = e(tZ + tZ_S + 1) + Z(te + tr' + 1) \\ &= tr'Z + teZ_S + Z + e \end{aligned} \tag{8}$$

Next consider the reduction of $Spoly(f_m, h)$ by $\{J_L + J_0\} \cup \{h\}$, where h itself is used in the division. The reduction $Spoly(f_m, h) \xrightarrow{h}_+$ is computed as,

$$\begin{aligned} tr'Z + teZ_S + Z + e &\xrightarrow{h}_+ tr'Z + (tr' + 1)Z_S + Z + e \\ &= tr'(Z + Z_S) + Z + Z_S + e \\ &= (tr' + 1)(Z + Z_S) + e \end{aligned} \tag{9}$$

Reducing the intermediate remainder of Eq. (9) by the polynomials in $J_L + J_0$ results in $(tr'+1)(r)+e$. This reduction process is similar to the one in the proof of Lemma 7.2. Now consider the polynomial $(tr' + 1)(r) + e$

$$\begin{aligned} (tr' + 1)(r) + e &= (tr' + 1)(e + r') + e \\ &= ter' + tr'^2 + e + r' + e \\ &= ter' + tr'^2 + r' \end{aligned} \tag{10}$$

The polynomial in Eq. (10) can be further reduced by h which results in 0 implying that $Spoly(f_m, h) \xrightarrow{\{J_L+J_0\}\cup\{h\}}_+ 0$.

$$\begin{aligned} ter' + tr'^2 + r' &\xrightarrow{h}_+ (tr' + 1)r' + tr'^2 + r' \\ &= tr'^2 + r' + tr'^2 + r' = 0 \end{aligned}$$

In summary, we have shown that to compute E_L as $GB(J_L + J_0) \cap \mathbb{F}_{2^k}[X_{PI}]$, we only need to compute $Spoly(f_m, f_o) \xrightarrow{J_L+J_0}_+ h$, and pair h with polynomials of J_0, as all other $Spoly(h, f_i)$ reduce to 0. This gives us the following procedure to compute the Gröbner basis of E_L (respectively E_H):

1. Compute $Spoly(f_o, f_m) \xrightarrow{J_L+J_0}_+ h$, where (f_m, f_o) is the only pair of polynomials in $J_L + J_0$ that do not have relatively prime leading monomials.
2. Use Buchberger's algorithm to compute GB of the set of vanishing polynomials and h, i.e. compute $G = GB(J_0 = \{x_i^2 - x_i : x_i \in X_{PI}\},\ h)$.
3. From G, collect the polynomials *not containing* t; i.e $E_L = G \cap \mathbb{F}_{2^k}[X_{PI}]$. These polynomials generate the ideal E_L.

The same technique is also used to compute E_H by replacing J_L with J_H in the above procedure. In our approach, we use the above procedures to compute E_L, E_H for Theorem 5.1 and then compute $U(X_{PI})$ from E_L using Algorithm 2.

8 Experimental Results

We have performed rectification experiments on finite field arithmetic circuits that are used in cryptography, where the implementation is different from the specification due to exactly one gate. This is to ensure that single-fix rectification is feasible for such bugs, so that a rectification function can be computed. We have implemented the procedures described in the previous sections—i.e.

the concepts of Theorem 5.1, Sect. 7 and Algorithm 2—using the SINGULAR symbolic algebra computation system [ver. 4-1-0] [41]. Given a *Spec*, a buggy *Impl* circuit C, and the set X of rectification targets, our approach checks for each net $x_i \in X$ if single-fix rectification is feasible, and if so, computes a rectification function $x_i = U(X_{PI})$. The experiments were conducted on a desktop computer with a 3.5 GHz Intel Core^{TM} i7-4770K Quad-core CPU, 16 GB RAM, running 64-bit Linux OS.

Experiments are performed with three different types of finite field circuit benchmarks. Two of these are the Mastrovito and the Montgomery multiplier circuit architectures used for modular multiplication. Mastrovito multipliers compute $Z = A \times B \pmod{P(x)}$ where $P(x)$ is a given primitive polynomial for the datapath size k. Montgomery multipliers are instead preferred for exponentiation operations (often required in cryptosystems). The last set of benchmarks are circuits implementing point addition over elliptic curves used for encryption, decryption and authentication in elliptic curve cryptography (ECC).

Table 1. Mastrovito multiplier rectification against Montgomery multiplier specification. Time in seconds; Time-out = 5400 s; k: Operand width

k	# of Gates		SAT	Theorem 5.1	Algorithm 2	Mem
	Mas	Mont				
4	48	96	0.09	0.03	0.001	8.16 MB
8	292	319	158.34	0.41	0.006	20.36 MB
9	237	396	4,507	0.47	0.001	18.95 MB
10	285	480	TO	0.84	0.001	28.2 MB
16	1,836	1,152	TO	73.63	0.024	0.32 GB
32	5,482	4,352	TO	3621	0.043	2.4 GB

First we present the results for the case where the reference *Spec* is given as a Montgomery multiplier, and the buggy implementation is given as a Mastrovito multiplier, which is to be rectified. Theorem 5.1, along with efficient GB-computation of the ideals E_L, E_H, is applied at a net $x_i \in X$, such that the circuit is rectifiable at x_i. Table 1 compares the execution time for the SAT-based approach of [16] against ours (Theorem 5.1) for checking whether a buggy Mastrovito multiplier can be rectified at a certain location in the circuit against a Montgomery multiplier specification. The SAT procedure is implemented using the *abc* tool [42]. We execute the command *inter* on the ON set and OFF set as described in [16]. The SAT-based procedure is unable to perform the necessary unsatisfiability check for circuits beyond 9-bit operand word-lengths, whereas our approach easily scales to 32-bit circuits. Using our approach, the polynomial $U(X_{PI})$ needed for rectification is computed from E_L and the time is reported in Table 1 in the Algorithm 2 column. The last column in the table shows the memory usage of our approach.

We also perform the rectification when the *Spec* is given as a polynomial expression instead of a circuit. Table 2 shows the results for checking whether the incorrect Mastrovito implementation can be single-fix rectified against the word-level specification polynomial $f_{spec} : Z_S + A \cdot B$.

Table 2. Mastrovito multiplier rectification against polynomial specification $Z_S = AB$. Time in seconds; Time-out = 5400 s; k: Operand width

k	# of Gates	Theorem 5.1	Algorithm 2	Mem
4	48	0.01	0.001	7.24 MB
8	292	0.08	0.006	14.95 MB
16	1,836	4.83	0.038	0.2 GB
32	5,482	100.52	0.015	1.42 GB
64	21,813	4,989	0.117	12.25 GB

Point addition is an important operation required for the task of encryption, decryption and authentication in ECC. Modern approaches represent the points in projective coordinate systems, *e.g.*, the López-Dahab (LD) projective coordinate [43], due to which the operations can be implemented as polynomials in the field.

Table 3. Point Addition circuit rectification against polynomial specification $D = B^2 \cdot (C + aZ_1^2)$. Time in seconds; Time-out = 5400 s; k: Operand width

k	# of Gates	Theorem 5.1	Algorithm 2	Mem
8	243	0.05	0.022	9.73 MB
16	1,277	3.48	0.019	88.78 MB
32	3,918	86.75	0.028	0.47 GB
64	1,5305	4,923	0.053	7.13 GB

Example 8.1. *Consider point addition in López-Dahab (LD) projective coordinate. Given an elliptic curve:* $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ *over* $\mathbb{F}_{2^k}$*, where* X, Y, Z *are* k*-bit vectors that are elements in* $\mathbb{F}_{2^k}$ *and similarly,* a, b *are constants from the field. We represent point addition over the elliptic curve as* $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1)$*. Then* X_3, Y_3, Z_3 *can be computed as follows:*

$$A = Y_2 \cdot Z_1^2 + Y_1 \qquad B = X_2 \cdot Z_1 + X_1$$
$$C = Z_1 \cdot B \qquad D = B^2 \cdot (C + aZ_1^2)$$
$$Z_3 = C^2 \qquad E = A \cdot C$$
$$X_3 = A^2 + D + E \qquad F = X_3 + X_2 \cdot Z_3$$
$$G = X_3 + Y_2 \cdot Z_3 \qquad Y_3 = E \cdot F + Z_3 \cdot G$$

Each of the polynomials in the above design are implemented as (gate-level) logic blocks and are interconnected to obtain final outputs X_3, Y_3 and Z_3. Table 3 shows the results for the block that computes $D = B^2 \cdot (C + aZ_1^2)$. Our approach can rectify up to 64-bit circuits.

Limitations of Our Approach: We also performed experiments where we apply Theorem 5.1 at a gate output which *cannot* rectify the circuit. We used the Montgomery multiplier as the specification and a Mastrovito multiplier as the implementation. For 4- and 8-bit word-lengths, the execution time of our approach was comparable to that of the SAT-based approach, and was ∼0.1 s. For the 16-bit multipliers, the SAT-based approach completed in 0.11 s. On the other hand, application of Theorem 5.1 resulted in a memory explosion and consumed ∼30 GB of memory within 5–6 min. This is due to the fact that when $1 \notin E_L + E_H$, then $GB(E_L + E_H)$ is not equal to $\{1\}$ and the Gröbner basis algorithm produces a very large output. To improve our approach we are working on term ordering heuristics so that our approach can perform efficiently in both cases. We also wish to employ other data-structures better suited to circuits, as SINGULAR's data structure is not very memory efficient. SINGULAR also has an upper limit on the number of variables (32,768) that can be accommodated in the system, limiting application to larger circuits.

9 Conclusion

This paper considers single-fix rectification of arithmetic circuits. The approach is applied after formal verification detects the presence of a bug in the design. We assume that post-verification debugging has been performed a set (X) of nets is provided as rectification targets. The paper presents necessary and sufficient conditions that ascertains whether a buggy circuit can be single-fix rectified at a net $x_i \in X$. When single-fix rectification is feasible, we compute a rectification polynomial function $x_i = U(X_{PI})$, which can be synthesized into a circuit. For this purpose, the paper introduces the notion of Craig interpolants in algebraic geometry in finite fields, proves their existence, and gives an effective procedure for their computation. Furthermore, we show how the rectification polynomial can be computed from algebraic interpolants. Experiments are performed over various finite field arithmetic circuits that show the efficiency of our approach as against SAT-based approaches. Limitations of our approach are also analyzed. We are currently investigating the extension of our approach to multi-fix rectification.

References

1. Ritirc, D., Biere, A., Kauers, M.: Column-wise verification of multipliers using computer algebra. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 23–30 (2017)
2. Ciesielski, M., Yu, C., Brown, W., Liu, D., Rossi, A.: Verification of gate-level arithmetic circuits by function extraction. In: 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2015)

3. Sayed-Ahmed, A., Große, D., Kühne, U., Soeken, M., Drechsler, R.: Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1048–1053 (2016)
4. Wienand, O., Wedler, M., Stoffel, D., Kunz, W., Greuel, G.-M.: An algebraic approach for proving data correctness in arithmetic data paths. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 473–486. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_45
5. Shekhar, N., Kalla, P., Enescu, F.: Equivalence verification of polynomial datapaths using ideal membership testing. IEEE Trans. CAD **26**(7), 1320–1330 (2007)
6. Tew, N., Kalla, P., Shekhar, N., Gopalakrishnan, S.: Verification of arithmetic datapaths using polynomial function models and congruence solving. In: Proceedings of International Conference on Computer-Aided Design (ICCAD), pp. 122–128 (2008)
7. Lv, J., Kalla, P., Enescu, F.: Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits. IEEE Trans. CAD **32**(9), 1409–1420 (2013)
8. Pruss, T., Kalla, P., Enescu, F.: Efficient symbolic computation for word-level abstraction from combinational circuits for verification over finite fields. IEEE Trans. CAD **35**(7), 1206–1218 (2016)
9. Lvov, A., Lastras-Montano, L., Trager, B., Paruthi, V., Shadowen, R., El-Zein, A.: Verification of Galois field based circuits by formal reasoning based on computational algebraic geometry. Form. Methods Syst. Des. **45**(2), 189–212 (2014)
10. Sun, X., Kalla, P., Pruss, T., Enescu, F.: Formal verification of sequential Galois field arithmetic circuits using algebraic geometry. In: Proceedings of Design, Automation and Test in Europe (2015)
11. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, New York (2007). https://doi.org/10.1007/978-0-387-35651-8
12. Adams, W.W., Loustaunau, P.: An Introduction to Gröbner Bases. American Mathematical Society, Providence (1994)
13. Madre, J.C., Coudert, O., Billon, J.P.: Automating the diagnosis and the rectification of design errors with PRIAM. In: Kuehlmann, A. (ed.) The Best of ICCAD. Springer, Boston (2003). https://doi.org/10.1007/978-1-4615-0292-0_2
14. Lin, C.C., Chen, K.C., Chang, S.C., Marek-Sadowska, M.: Logic synthesis for engineering change. In: Proceedings of Design Automation Conference (DAC), pp. 647–652 (1995)
15. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: Equivalence Checking of Digital Circuits. Springer, Boston (2004)
16. Tang, K.F., Wu, C.A., Huang, P.K., Huang, C.Y.: Interpolation-based incremental ECO synthesis for multi-error logic rectification. In: Proceedings of Design Automation Conference (DAC), pp. 146–151 (2011)
17. Craig, W.: Linear reasoning: a new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957)
18. Liaw, H.T., Tsaih, J.H., Lin, C.S.: Efficient automatic diagnosis of digital circuits. In: Proceedings of ICCAD, pp. 464–467 (1990)
19. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulae. In: IEEE International Conference on Computer Design (ICCD) (2013)

20. Jo, S., Matsumoto, T., Fujita, M.: SAT-based automatic rectification and debugging of combinational circuits with LUT insertions. In: IEEE 21st Asian Test Symposium (2012)
21. Fujita, M., Mishchenko, A.: Logic synthesis and verification on fixed topology. In: 22nd International Conference on Very Large Scale Integration (VLSI-SoC) (2014)
22. Fujita, M.: Toward unification of synthesis and verification in topologically constrained logic design. Proc. IEEE **103**, 2052–2060 (2015)
23. Wu, B.H., Yang, C.J., Huang, C.Y., Jiang, J.H.R.: A robust functional ECO engine by SAT proof minimization and interpolation techniques. In: International Conference on Computer Aided Design, pp. 729–734 (2010)
24. Ling, A.C., Brown, S.D., Safarpour, S., Zhu, J.: Toward automated ECOs in FPGAs. IEEE Trans. CAD **30**(1), 18–30 (2011)
25. Dao, A.Q., et al.: Efficient computation of ECO patch functions. In: 55th Design Automation Conference (DAC), pp. 51:1–51:6, June 2018
26. Gupta, U., Ilioaea, I., Rao, V., Srinath, A., Kalla, P., Enescu, F.: On the rectifiability of arithmetic circuits using Craig interpolants in finite fields. In: International Conference on Very Large Scale Integration (VLSI-SoC), pp. 49–54 (2018)
27. Ghandali, S., Yu, C., Liu, D., Brown, W., Ciesielski, M.: Logic debugging of arithmetic circuits. In: IEEE Computer Society Annual Symposium on VLSI (2015)
28. Farahmandi, F., Mishra, P.: Automated debugging of arithmetic circuits using incremental Gröbner basis reduction. In: IEEE International Conference on Computer Design (ICCD) (2017)
29. Farahmandi, F., Mishra, P.: Automated test generation for debugging arithmetic circuits. In: Proceedings of the 2016 Conference on Design, Automation & Test in Europe, DATE (2016)
30. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_1
31. Lee, R.-R., Jiang, J.-H.R., Hung, W.-L.: Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In: Proceedings of Design Automation Conference (DAC), pp. 636–641 (2008)
32. McMillan, K.: An interpolating theorem prover, theoretical computer science. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004), vol. 345, no. 1, pp. 101–121 (2005)
33. Kapur, D., Majumdar, R., Zarba, G.: Interpolation for data-structures. In: Proceedings of ACM SIGSOFT International Symposium on Foundation of Software Engineering, pp. 105–116 (2006)
34. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo theories. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 397–412. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_30
35. Griggio, A.: Effective word-level interpolation for software verification. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 28–36 (2011)
36. Gao, S., Platzer, A., Clarke, E.: Quantifier elimination over finite fields with Gröbner bases. In: Algebraic Informatics: 4th International Conference, CAI, pp. 140–157 (2011)
37. Gao, S.: Counting zeros over finite fields with Gröbner bases. Master's thesis, Carnegie Mellon University (2009)
38. Lv, J.: Scalable formal verification of finite field arithmetic circuits using computer algebra techniques. Ph.D. dissertation, Univ. of Utah, August 2012

39. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. dissertation, University of Innsbruck (1965)
40. Buchberger, B.: A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. LNCS, vol. 72, pp. 3–21. Springer, Heidelberg (1979). https://doi.org/10.1007/3-540-09519-5_52
41. Decker, W., Greuel, G.-M., Pfister, G., Schönemann, H.: Singular 4-1-0 – a computer algebra system for polynomial computations (2016). http://www.singular.uni-kl.de
42. Brayton, R., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 24–40. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_5
43. López, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48892-8_16