Logic Synthesis from Polynomials with Coefficients in the Field of Rationals

Bhavani Sampathkumar[1], Bailey Martin[1], Ritaja Das[1], Priyank Kalla[1], and Florian Enescu[2]
[1]Electrical & Computer Engineering, University of Utah, Salt Lake City, UT, USA
[2]Mathematics & Statistics, Georgia State University, Atlanta, GA, USA

***Abstract*—This paper introduces a novel concept of performing logic synthesis from multivariate polynomials with coefficients in the field of rationals ($\mathbb{Q}$), where the variables take only Boolean values. Such polynomials are encountered during synthesis and verification of arithmetic circuits using computer algebra and algebraic geometry based techniques. The approach takes as input a polynomial f over $\mathbb{Q}$ with binary variables, and derives a corresponding polynomial $\widetilde{f}$ over the finite field ($\mathbb{F}_2$) of two elements, such that $\widetilde{f}$ has the same variety (zero-set) as f. As $\mathbb{F}_2$ is isomorphic to Boolean algebra, $\widetilde{f}$ can be translated to a Boolean network by mapping the products and sums as AND and XOR operators, respectively. We prove the correctness of our algebraic transformation, and present a recursive algorithm for the same. The translated $\widetilde{f} \in \mathbb{F}_2$ resultingly corresponds to a positive Davio decomposition, and is computed using both explicit and implicit representations. The approach is used to synthesize subfunctions of arithmetic circuits, under the partial synthesis framework. The efficacy of our approach is demonstrated over various integer multiplier architectures, where other contemporary approaches are infeasible.**

I. Introduction

Modern *formal verification* techniques for integer arithmetic circuits model the circuit's functionality using a set of multivariate polynomials, where the variables take only Boolean values, and the coefficients lie in the field of fractions ($\mathbb{Q}$) [1] [2] [3]. Verification is then solved by dividing a specification (*Spec*) polynomial f by a Gröbner basis (GB) [4] of the set of polynomials of the circuit, and checking if the obtained remainder is 0.

Modeling integer arithmetic circuits using polynomials with fractional coefficients has been an important contribution. Since $\mathbb{Q}$ forms a field, while the set of integers $\mathbb{Z}$ does not due to the lack of multiplicative inverses, the decision problems (equivalence check) can be formulated over $\mathbb{Q}$ using the *Nullstellensatz* (Ch. 4 in [5]). By representing the polynomials using a specific term order, and by virtue of $\mathbb{Z} \subset \mathbb{Q}$, it was shown that the GB-based reduction for digital circuits never produces results with fractional coefficients, even though the computations are performed over $\mathbb{Q}$. The soundness and completeness of such an algebraic model for verification was proven in [1]. Its efficacy as compared to SAT and decision diagrams was also demonstrated [2] [3].

Due to its success for verification, the algebraic model has also been explored for *partial synthesis of rectification functions for buggy arithmetic circuits*, both for finite field circuits [6], as well as for integer arithmetic circuits [7]. These techniques also model the circuit by way of a set of polynomials, and perform GB computations to compute rectification functions. Interestingly, when the polynomial model with coefficients in $\mathbb{Q}$ is utilized for partial synthesis of integer arithmetic circuits, *it is observed that the extended Gröbner basis computations may result in polynomials with fractional coefficients* [8]. Subsequently, given a rectification polynomial f with fractional coefficients, it is required to synthesize a corresponding Boolean function $\widetilde{f_{\mathbb{B}}}$ to patch the circuit.

Objective: Given the above context, this paper addresses a novel problem of logic synthesis from polynomials with binary variables and coefficients in $\mathbb{Q}$. Since the polynomial f can evaluate to any value in $\mathbb{Q}$, we have to further ensure that: i) for those variable assignments where f evaluates to 0, $\widetilde{f_{\mathbb{B}}}$ should too; and ii) for those input assignments where $f \neq 0$, we should have $\widetilde{f_{\mathbb{B}}} = 1$. This ensures that $\widetilde{f_{\mathbb{B}}}$ is a Boolean function that has the same zero-set as f.

Approach and Contributions: Our approach takes an algebraic geometry view of the problem, and translates the given polynomial f with fractional coefficients to another polynomial $\widetilde{f}$ with coefficients in the finite field $\mathbb{F}_2$ of 2 elements $\{0,1\}$. Moreover, we ensure that both f and $\widetilde{f}$ have the *same zero-set (variety)*, when their variables are restricted to binary Boolean values. Thus, when f evaluates to nonzero values in $\mathbb{Q}$, $\widetilde{f} = 1$, as $\widetilde{f}$ evaluates in $\mathbb{F}_2$. We prove the existence of such a transformation, and our proof motivates a recursive algorithm for this computation. Since Boolean algebra is isomorphic to polynomial algebra over $\mathbb{F}_2$, the polynomial $\widetilde{f}$ can be translated to a AND-XOR Boolean expression $\widetilde{f_{\mathbb{B}}}$ by replacing $(+,\cdot)$ with (XOR $\oplus$, AND $\wedge$), respectively. Such polynomials can be computed for both an on-set function, as well as a don't-care set function, which can be then synthesized with a logic synthesis tool.

While our approach relies upon concepts from algebraic geometry, it turns out that the (de)composition of $\widetilde{f_{\mathbb{B}}}$ resembles a *positive Davio decomposition*, as implemented in functional decision diagrams [9] [10]. We implement our approach to transform f to $\widetilde{f_{\mathbb{B}}}$ as a recursive algorithm, with both an explicit representation of $\widetilde{f}$ as a set of monomial terms, and also with an implicit set representation using the CUDD decision diagrams package [11].

Example I.1. Let us illustrate the problem by means of an example. Let $f = (4/3)a_0a_1b_0b_1 - 2a_0b_0b_1 - (2/7)a_1b_0$ with 4 Boolean variables and fractional coefficients. Then there exists a corresponding polynomial $\widetilde{f} = a_0a_1b_0b_1 + a_0b_0b_1 + a_1b_0$ over $\mathbb{F}_2$ with the same zeros as f. Table I shows that for all variable

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

assignments $\underline{x}$ where $f(\underline{x}) = 0$, we have $\widetilde{f}(\underline{x}) = 0$. Whereas when $f(\underline{x}) \neq 0$, we have $\widetilde{f}(\underline{x}) = 1$. Then, a Boolean function can be generated from $\widetilde{f}$ as $\widetilde{f}_{\mathbb{B}} = (a_0a_1b_0b_1) \oplus (a_0b_0b_1) \oplus (a_1b_0)$.

$\{a_1a_0b_1b_0\}$	f	$\widetilde{f}$	$\{a_1a_0b_1b_0\}$	f	$\widetilde{f}$
0000	0	0	1000	0	0
0001	0	0	1001	-2/7	1
0010	0	0	1010	0	0
0011	0	0	1011	-2/7	1
0100	0	0	1100	0	0
0101	0	0	1101	-2/7	1
0110	0	0	1110	0	0
0111	-2	1	1111	-20/21	1

TABLE I: Evaluation of the polynomials. The shaded rows depict a few cases where $f \neq 0$ implies $\widetilde{f} = 1$.

Paper Organization: The following section, Section II, covers the notation used and the preliminary background. Section III reviews related previous work. Section IV formally states the problem, proves the existence of such a translation of f in $\mathbb{Q}$ to $\widetilde{f}$ in $\mathbb{F}_2$. The algorithm and implementation is described in Section V, whereas Section VI describes the experimental results. Section VII concludes the paper.

II. Notation and Background Concepts

Let $\mathbb{B}$ denote the Boolean domain, $\mathbb{F}_2 = \{0, 1\}$ the finite field of 2 elements, and $\mathbb{Q}$ the field of rational numbers. Let $\{x_1, \dots, x_n\}$ denote a set of variables. Let $\mathbb{K}$ be any field, e.g. $\mathbb{K} = \mathbb{Q}$ or $\mathbb{K} = \mathbb{F}_2$; then $R = \mathbb{K}[x_1, \dots, x_n]$ denotes the ring of polynomials in variables $x_1, \dots, x_n$, with coefficients in $\mathbb{K}$. A polynomial $f \in R$ is written as a sum of terms, $f = c_1X_1 + c_2X_2 + \dots + c_tX_t$, where c_i denote elements in $\mathbb{K}$, X_i denote monomials, and c_iX_i is a term. A monomial is a power-product of the form $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$, with $\alpha_i \in \mathbb{Z}_{\geq 0}$.

In our work, the variables $\{x_1, \dots, x_n\}$ correspond to nets in a circuit, and the gates of the circuit are modeled using a set of polynomials $F = \{f_1, \dots, f_s\}$. Given a set of polynomials $F = \{f_1, \dots, f_s\}$ from R, the *ideal* generated by F is $J = \langle F \rangle = \langle f_1, \dots, f_s \rangle = \{h_1 \cdot f_1 + \dots + h_s \cdot f_s : h_1, \dots, h_s \in R\}$. The polynomials $f_1, \dots, f_s$ form the basis of ideal J.

Let $\underline{x} = (\underline{x_1}, \dots, \underline{x_n}) \in \mathbb{K}^n$ be a point in the affine space, and f a polynomial in R. If $f(\underline{x}) = 0$, we say that f *vanishes* on $\underline{x}$. We have to analyze the *set of all common zeros* of the polynomials of F that lie within the field $\mathbb{K}$. 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 $V_{\mathbb{K}}(J)$, where: $V_{\mathbb{K}}(J) = V_{\mathbb{K}}(f_1, \dots, f_s) = \{\underline{x} \in \mathbb{K}^n : \forall f \in J, f(\underline{x}) = 0\}$.

An ideal may have many different sets of generators that have the same variety; i.e. it is possible to have $J = \langle f_1, \dots, f_s \rangle = \langle g_1, \dots, g_t \rangle = \dots = \langle h_1, \dots, h_r \rangle$, such that $V_{\mathbb{K}}(f_1, \dots, f_s) = V_{\mathbb{K}}(g_1, \dots, g_t) = \dots = V_{\mathbb{K}}(h_1 \dots, h_r)$. A Gröbner basis (GB) of an ideal is one such generating set $G = \{g_1, \dots, g_t\}$, that is a canonical representation of the ideal, and helps in solving many polynomial decision and quantification problems. A Gröbner basis can be computed using the Buchberger's algorithm (Alg. 1.7.1 in [4]). It takes as input a set of polynomials $\{f_1, \dots, f_s\}$ and computes its GB $G = \{g_1, g_2, \cdots, g_t\}$.

A. The ideal of Vanishing Polynomials, Idempotency and Quotient Rings

In this work, we consider $f \in R = \mathbb{Q}[x_1, \dots, x_n]$, where f is derived from a circuit, so $x_1, \dots, x_n$ take only Boolean values 0 or 1. Equivalently, each variable x_i is idempotent in that $x_i^2 = x_i$ or $x_i^2 - x_i = 0$. We call the polynomial $(x_i^2 - x_i)$ a *Boolean vanishing polynomial (BVP)* as it vanishes on $x_i = 0, 1$. Let $F_0 = \{x_1^2 - x_1, \dots, x_n^2 - x_n\}$ be the set of all BVPs in $\mathbb{Z}[x_1, \dots, x_n]$, and $J_0 = \langle F_0 \rangle$. Note that there are natural maps from $\mathbb{Z}[x_1, \dots, x_n]$ to $R = \mathbb{Q}[x_1, \dots, x_n]$ and to $S = \mathbb{F}_2[x_1, \dots, x_n]$, which lead to the ideals F_0R in R and F_0S in S, generated by F_0 in R and, respectively, S. In other words, the same set of BVPs $\{x_i^2 - x_i : i = 1, \dots, n\}$ denotes different ideals in different rings: F_0R in R, and F_0S in S. We will use J_0 to denote F_0R.

Imposition of this Boolean idempotency for each variable in f requires that f is always reduced $\pmod{J_0}$, i.e. f is divided by BVPs for all variables and the resulting remainder r is taken as the result: denoted $f \pmod{J_0} = f \xrightarrow{J_0 = \langle x_1^2 - x_1, \dots, x_n^2 - x_n \rangle}_+ r$. Thus, our computations are effectively performed over the *quotient ring* $\frac{\mathbb{Q}[x_1, \dots, x_n]}{F_0R}$, where $F_0R = \langle x_1^2 - x_1, \dots, x_n^2 - x_n \rangle$ is the ideal generated by all BVPs in R. As a result, higher degree variables $x_i^k, k \geq 2$, are reduced to x_i in the computations since $x_i^k = x_i \pmod{J_0}$. Thus f comprises terms with only *multilinear monomials* $x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, with $\alpha_i \in \{0, 1\}$.

In other words, the Boolean idempotency on variables of f is enforced by considering the the ideal $\langle f, F_0R \rangle$. Moreover, the variety $V_{\mathbb{Q}}(f, F_0R) \subset \{0, 1\}^n$. Note that in Example I.1, $f = (4/3)a_0a_1b_0b_1 - 2a_0b_0b_1 - (2/7)a_1b_0$ has multilinear monomials, as it is already reduced $\pmod{F_0R = \langle a_0^2 - a_0, b_0^2 - b_0, a_1^2 - a_1, b_1^2 - b_1 \rangle}$. In the sequel, we will assume that every polynomial is reduced to its multilinear form $\pmod{F_0R}$.

III. Review of Related Work

Previous work has investigated the modeling of arithmetic circuits using commutative algebra and algebraic geometry. The work of [1] [2] [3] has addressed verification of arithmetic circuits using Gröbner basis techniques, but the synthesis problem has not been solved by them. The verification problem is formulated as an ideal membership test, which is a decision problem. However, the synthesis problem is a quantification problem, which is computationally more challenging.

The synthesis problem has been addressed in the context of rectification for engineering change orders (ECO) as well as for rectification of buggy arithmetic circuits. The ECO rectification approaches are based on SAT and Craig Interpolation based models [12], or Quantifier Boolean Formula (QBF) and iterative-SAT based models [13] [14]. Recently, there has been some interest in computing rectification functions for arithmetic circuits using computer algebra techniques, such as [15] [6] [7] [16], among others. Of these [15] [6] compute Boolean rectification functions from polynomial models over finite fields $\mathbb{F}_{2^k}$ of 2^k elements. Since such binary finite fields are k-dimensional extensions of $\mathbb{F}_2$, translating polynomials from $\mathbb{F}_{2^k}$ to AND-XOR networks is fairly straightforward.

In [16], a buggy integer arithmetic circuit is patched at all the primary outputs where the bug-effect is observable, by implementing half-adders and carry-propagate logic at those outputs. However, the paper does not provide a proof of soundness or completeness of their approach. The approach captures the functionality of a bug in a polynomial f with *integer coefficients* and attempts to patch the bug at the primary outputs using half and full adders. However, this approach cannot synthesize a Boolean function from a polynomial f with fractional coefficients.

The works that are closely related to this paper are by [8], and [7]. In [8], the author shows that a rectification polynomial f for an integer arithmetic circuit can be computed using the extended Gröbner basis algorithm, by operating over the ring $\mathbb{Q}[x_1, \dots, x_n]$. Interestingly, this work discovers that f may have rational coefficients – which is not surprising as a polynomial is computed over $\mathbb{Q}$. Moreover, at an algebraic level, this rectification polynomial f can patch the bug, in the sense that replacing the polynomial f for a buggy gate passes the verification test. However, no efficient logic synthesis approach was presented to generate logic circuits from f.

Rao et al. in [7] address the problem of synthesizing a Boolean rectification function $\widetilde{f_{\mathbb{B}}}$ from a polynomial $f \in \mathbb{Q}[x_1, \dots, x_n]$, such that f and $\widetilde{f_{\mathbb{B}}}$ have the same zero-set. Their approach is as follows: i) They compute a *reduced Gröbner basis* $G = GB(f, J_0 = \langle F_0 R \rangle) = \{g_1, \dots, g_t\}$. ii) Then, the polynomial $\widetilde{f} \in \mathbb{F}_2[x_1, \dots, x_n]$ is computed using the formula

$$\widetilde{f} = (1 + g_1)(1 + g_2) \cdots (1 + g_t) + 1 \pmod 2.$$

Then $\widetilde{f}$ is translated to an AND-XOR network from $\mathbb{F}_2$, which can be given to a logic synthesis tool for optimized implementation.

While the approach theoretically solves the problem above, computing a reduced Gröbner basis $G = GB(f, J_0)$ is infeasible due to its very high complexity, even for small rectification functions for 8-bit multipliers (shown in Table II in the sequel). Therefore, there is a need for a non-Gröbner basis based approach to practically compute $\widetilde{f}$ from f.

This paper addresses the above problem by relating the variety $V_{\mathbb{Q}}(f, F_0 R)$ and $V_{\mathbb{F}_2}(\widetilde{f}, F_0 S)$. We do not explain how a rectification polynomial computation may generate f with rational coefficients. We refer the reader to [8] (Chapter 6) and [7] for more details. We only address the problem of synthesizing Boolean functions $\widetilde{f_{\mathbb{B}}}$ from f.

IV. TRANSLATING POLYNOMIALS FROM $\mathbb{Q}$ TO $\mathbb{F}_2$

Problem Statement: Given the polynomial $f \in R = \mathbb{Q}[x_1, \dots, x_n]$, derive a corresponding polynomial $\widetilde{f} \in \mathbb{F}_2[x_1, \dots, x_n]$, such that f and $\widetilde{f}$ have the same variety when their variables are restricted to 0 and 1. More precisely, given f, find $\widetilde{f}$ such that

$$\underbrace{V_{\mathbb{Q}}(f, F_0 R)}_{\text{(0,1)-tuple in } \mathbb{Q}} = \underbrace{V_{\mathbb{F}_2}(\widetilde{f}, F_0 S)}_{\text{(0,1)-tuple in } \mathbb{F}_2}. \quad (1)$$

Note that, although they are distinct sets, there is a one-to-one correspondence between the set $\{0, 1\}^n \subset \mathbb{Q}^n$ and $\{0, 1\}^n \subset \mathbb{F}_2^n$ where 0 in $\mathbb{Q}$ corresponds to 0 in $\mathbb{F}_2$ and 1 in $\mathbb{Q}$ corresponds to 1 in $\mathbb{F}_2$. We denote this correspondence simply by the identity function to avoid cumbersome notations. Moreover, $V_{\mathbb{Q}}(f, F_0 R) = V_{\mathbb{F}_2}(\widetilde{f}, F_0 S)$ implies that: i) for all points $\underline{x} \in V_{\mathbb{Q}}(f, F_0 R)$ we have $f(\underline{x}) = \widetilde{f}(\underline{x}) = 0$; and ii) for all points $\underline{x} \notin V_{\mathbb{Q}}(f, F_0 R)$, we have that $\widetilde{f}(\underline{x}) = 1$, as $\widetilde{f}$ only evaluates to 0 or 1.

Approach: We describe the transformation approach as follows. To begin with, assume that $f \in R = \mathbb{Q}[x_1, \dots, x_n]$ is reduced $\pmod{F_0 R}$, so that f consists of only multilinear terms. Impose a lexicographic term order on f, with variable order $x_1 > x_2 > \cdots > x_n$. Without loss of generality, assume $x_i = x_1$ is the largest variable in the order that appears in f (otherwise, we can relabel the variables). Then one can factorize f w.r.t. x_1, so that $f = hx_1 + g$, where $h, g \in \mathbb{Q}[x_2, \dots, x_n]$ and $h \neq 0$. We state and prove the following result.

Theorem IV.1 (The Translation Theorem)**.** Let f be a polynomial in $R = \mathbb{Q}[x_1, \dots, x_n]$, written in the form $f = hx_1 + g$ as shown above. Then there exists a polynomial $\widetilde{f} \in \mathbb{F}_2[x_1, \dots, x_n]$ where f and $\widetilde{f}$ have the same variety as $\{0, 1\}$-tuples, i.e. $V_{\mathbb{Q}}(f, F_0 R) = V_{\mathbb{F}_2}(\widetilde{f}, F_0 S)$.

Proof: Consider $f = hx_1 + g$ in R. Let $(\underline{x_1}, \underline{x_2}, \dots, \underline{x_n}) \in \{0, 1\}^n$ be a point which we denote as $(\underline{x_1}, \underline{y})$, where $\underline{y} = (\underline{x_2}, \dots, \underline{x_n})$. Then:

$$f(\underline{x_1}, \underline{y}) = 0 \quad (2)$$

$$\implies h(\underline{y}) \cdot \underline{x_1} + g(\underline{y}) = 0 \quad (3)$$

$$\implies h(\underline{y}) \cdot \underline{x_1} = -g(\underline{y}). \quad (4)$$

From Eqn. (4), we have that when $\underline{x_1} = 1, h(\underline{y}) = -g(\underline{y})$, or $h(\underline{y}) + g(\underline{y}) = 0$. Also, $\underline{x_1} = 0$ implies that $g(\underline{y}) = 0$. So, from Eqn. (4) we conclude that

$$V_{\mathbb{Q}}(f, F_0 R) = \{(1, \underline{y}) : \underline{y} \in V_{\mathbb{Q}}(h + g, F_0 R)\} \cup \{(0, \underline{y}) : \underline{y} \in V_{\mathbb{Q}}(g, F_0 R)\}. \quad (5)$$

In other words, $V_{\mathbb{Q}}(f, F_0 R)$ consists of tuples $(1, \underline{y})$ where $h + g$ vanishes on $\underline{y}$, and tuples $(0, \underline{y})$ where g vanishes on $\underline{y}$.

This observation allows to prove the theorem by *induction*. The verification for $n = 1$ (one variable) is straightforward. When x_1 is the only variable, then $f = hx_1 + g$ is such that h, g are constants in $\mathbb{Q}$. We can consider the following 4 cases:

1) Case 1: $h = 0, g = 0$. Then $\widetilde{f} = 0$.
2) Case 2: $h \neq 0, g = 0$. Then $\widetilde{f} = x_1$, because $V_{\mathbb{Q}}(f = hx_1, F_0 R) = V_{\mathbb{F}_2}(\widetilde{f} = x_1, F_0 S) = \{(0)\}$.
3) Case 3: $h = 0, g \neq 0$. Then $\widetilde{f} = 1$, because a non-zero constant has no roots.
4) Case 4: $h \neq 0, g \neq 0$. In this case, Eqn. (5) tells us that if $h + g = 0$, then $\widetilde{f} = x_1 + 1$; otherwise, $\widetilde{f} = 1$.

Thus, for every $f(x_1) \in \mathbb{Q}[x_1]$, there exists a corresponding $\widetilde{f} \in \mathbb{F}_2[x_1]$ with $V_{\mathbb{Q}}(f, x_1^2 - x_1) = V_{\mathbb{F}_2}(\widetilde{f}, x_1^2 - x_1)$.

Now assume that the theorem statement is true for polynomials with up to and including $n - 1$ variables. Therefore, we can find $\widetilde{p}$ and $\widetilde{g} \in \mathbb{F}_2[x_2, \dots, x_n]$ that satisfy the following:

1) The polynomial $\widetilde{p}$ is such that $V_{\mathbb{F}_2}(\widetilde{p}, F_0S) = V_{\mathbb{Q}}(h+g, F_0R) \subseteq \{0,1\}^{n-1}$, i.e. the varieties of $\widetilde{p}$ and $h+g$ are equal as $\{0,1\}$-tuples. That is, for a polynomial $h+g$ in $n-1$ variables, there exists a corresponding $\widetilde{p}$ with coefficients in $\mathbb{F}_2$.
2) The polynomial $\widetilde{g}$ is such that $V_{\mathbb{F}_2}(\widetilde{g}, F_0S) = V_{\mathbb{Q}}(g, F_0) \subseteq \{0,1\}^{n-1}$.

Let $\widetilde{h} = \widetilde{p} - \widetilde{g}$. As above, considering the polynomial $\widetilde{f} = \widetilde{h}x_1 + \widetilde{g} = (\widetilde{p} - \widetilde{g})x_1 + \widetilde{g}$ over $\mathbb{F}_2$, and a point $(\underline{x_1}, \underline{y}) \in V_{\mathbb{F}_2}(\widetilde{f}, F_0S) \subset \{0,1\}^n$, we see that:

$$\widetilde{f}(\underline{x_1}, \underline{y}) = 0 \tag{6}$$
$$\implies (\widetilde{p}(\underline{y}) - \widetilde{g}(\underline{y})) \cdot \underline{x_1} = -\widetilde{g}(\underline{y}) \tag{7}$$
$$\implies (\widetilde{p}(\underline{y}) - \widetilde{g}(\underline{y})) \cdot \underline{x_1} = \widetilde{g}(\underline{y}), \tag{8}$$

as $-1 = +1 \pmod 2$. Therefore,

$$\underline{x_1} = 1 \implies \widetilde{p}(\underline{y}) - \widetilde{g}(\underline{y}) = \widetilde{g}(\underline{y}) \tag{9}$$
$$\implies \widetilde{p}(\underline{y}) = 0 \tag{10}$$
$$\implies \underline{y} \in V_{\mathbb{F}_2}(\widetilde{p}, F_0) \tag{11}$$
$$\underline{x_1} = 0 \implies \widetilde{g}(\underline{y}) = 0 \tag{12}$$
$$\implies \underline{y} \in V_{\mathbb{F}_2}(\widetilde{g}, F_0). \tag{13}$$

In conclusion,

$$V_{\mathbb{F}_2}(\widetilde{f}, F_0S) = \{(1, \underline{y}) : \underline{y} \in V_{\mathbb{F}_2}(\widetilde{p}, F_0S)\} \cup \{(0, \underline{y}) : \underline{y} \in V_{\mathbb{F}_2}(\widetilde{g}, F_0S)\}.$$

But, $\{(1, \underline{y}) : \underline{y} \in V_{\mathbb{F}_2}(\widetilde{p}, F_0S)\} \cup \{(0, \underline{y}) : \underline{y} \in V_{\mathbb{F}_2}(\widetilde{g}, F_0S)\}$
$= \{(1, \underline{y}) : \underline{y} \in V_{\mathbb{Q}}(h+g, F_0R)\} \cup \{(0, \underline{y}) : \underline{y} \in V_{\mathbb{Q}}(g, F_0R)\}$.

This proves our claim that for every $f \in \mathbb{Q}[x_1, \dots, x_n]$, there exists a corresponding $\widetilde{f} \in \mathbb{F}_2[x_1, \dots, x_n]$, s.t.

$$V_{\mathbb{Q}}(f, F_0R) = V_{\mathbb{F}_2}(\widetilde{f}, F_0S).$$

■

V. Algorithm & Implementation

The proof of the theorem inspires an algorithm to recursively compute $\widetilde{f}$ – actually, directly the Boolean function $\widetilde{f}_{\mathbb{B}}$ – from f. Impose a lex term order on the given f, with variable order $x_1 > x_2 > \cdots > x_n$. Assume that f is already multilinear in the variables; if not, then it can be reduced $\pmod{F_0R}$ to make it multilinear. Decompose $f = hx_1 + g$, by dividing f by variable x_1, and obtaining the quotient h and remainder g in $\mathbb{Q}[x_2, \dots, x_n]$. Let $p = h + g$. Recursively decompose polynomials p, g, until the decomposed polynomials contain only *one term* cX. Then, recursion bottoms out as follows: i) if the decomposed polynomial p (resp. g) is 0, return $\widetilde{p} = 0$ (resp. $\widetilde{g} = 0$); ii) if p (resp. g) is a non-zero constant, return $\widetilde{p} = 1$ (resp. $\widetilde{g} = 1$); iii) if $p = cX \in R$ (resp. $g = cX$), is a single term, then return $\widetilde{p} = X \in \mathbb{F}_2[X]$ (resp. $\widetilde{g} = X$). Reconstruct $\widetilde{f} = (\widetilde{p} - \widetilde{g}) \cdot x_i + \widetilde{g}$. Since $-1 \equiv +1 \pmod 2 \equiv \oplus$, the required Boolean expression can be reconstructed as: $\widetilde{f}_{\mathbb{B}} = (\widetilde{p} \oplus \widetilde{g}) \cdot x_i \oplus \widetilde{g}$.

Algorithm 1 shows the recursive procedure. Lines 2-7 show the terminal cases of recursion. Lines 9-12 decompose $f = hx_i + g$, and compute compute $h, g, p = h + g$. Line 9 applies a heuristic to select a variable x_i for factorization/division. From the polynomial f, we select the variable that has the highest *activity* – i.e. the variable that appears in the most number of terms in f. This is done so to keep the recursion tree as balanced as possible, or to bottom-out the recursion early. When $\widetilde{p}, \widetilde{g}$ are returned by the recursive calls in lines 13-14, the Boolean function $\widetilde{f}_{\mathbb{B}}$ (or equivalently the polynomial $\widetilde{f} \in \mathbb{F}_2[x_1, \dots, x_n]$) is constructed in line 15, where $\oplus$ denotes addition (mod 2), or the XOR operation.

Algorithm 1: Compute $\widetilde{f}_{\mathbb{B}}$ from f

Input : $f \in \mathbb{Q}[x_1, x_2, \dots, x_n], x_1 > \cdots > x_n$
Output: Boolean function $\widetilde{f}_{\mathbb{B}}$ with same zeros as f

1 **function:** $Poly\mathbb{Q}to\mathbb{F}_2(f)$
2 **if** $f == 0$ **then**
3   return $\widetilde{f}_{\mathbb{B}} = 0$;
4 **else if** $f == c \neq 0$ **then**
    `// f is a non-zero constant c`
5   return $\widetilde{f}_{\mathbb{B}} = 1$;
6 **else if** $f ==$ *single term* cX **then**
7   return $\widetilde{f}_{\mathbb{B}} = X$; `//` $V_{\mathbb{Q}}(cX, F_0) = V_{\mathbb{F}_2}(X, F_0)$
8 **else**
9     $x_i = select\_high\_activity\_variable(f)$;
    `// Decompose f w.r.t.` $x_i := hx_i + g$
10     $h = f/x_i$; `// h =quotient, division by` x_i
11     $g = f \pmod{x_i}$;`// g = remainder`
12     $p = h + g$;
13     $\widetilde{p} = Poly\mathbb{Q}to\mathbb{F}_2(p)$; `//` $\widetilde{p}$ `has the same variety as` $V_{\mathbb{Q}}(h+g)$
14     $\widetilde{g} = Poly\mathbb{Q}to\mathbb{F}_2(g)$; `//` $\widetilde{g}$ `has the same variety as` $V_{\mathbb{Q}}(g)$
15     $\widetilde{f}_{\mathbb{B}} = (\widetilde{p} \oplus \widetilde{g}) \cdot x_i \oplus \widetilde{g}$;
16   return $\widetilde{f}_{\mathbb{B}}$;
17 **end**

Example V.1. We demonstrate the algorithm on the polynomial $f = (4/3)a_0a_1b_0b_1 - 2a_0b_0b_1 - (2/7)a_1b_0$. The highest activity variable is b_0, so we factorize $f = ((4/3)a_0a_1b_1 - 2a_0b_1 - (2/7)a_1)b_0$. We denote by subscript i, the decomposed polynomials p_i, g_i at recursion-level i.

At the first recursion level: $h_1 = (4/3)a_0a_1b_1 - 2a_0b_1 - (2/7)a_1$, and $g_1 = 0$. Then $p_1 = h_1 + 0 = (4/3)a_0a_1b_1 - 2a_0b_1 - (2/7)a_1$. Since $g_1 = 0, \widetilde{g_1} = 0$.

At second recursion level, $f = p_1 = (4/3)a_0a_1b_1 - 2a_0b_1 - (2/7)a_1$. Expansion variable selected is a_0. Then $h_2 = (4/3)a_1b_1 - 2b_1, g_2 = -(2/7)a_1, p_2 = (4/3)a_1b_1 - 2b_1 - (2/7)a_1$. Since g_2 is terminal case, $\widetilde{g_2} = a_1$, and so on.

Fig. 1 depicts the recursion tree. The returned Boolean functions $\widetilde{p}, \widetilde{g}$ at each recursion step are shown in red color. The final returned Boolean function $\widetilde{f}_{\mathbb{B}} = (a_0a_1b_0b_1) \oplus (a_0b_0b_1) \oplus (a_1b_0)$, which matches the one in Example I.1.

While the algorithm exhibits exponential worst-case complexity, experiments in the sequel show that the number

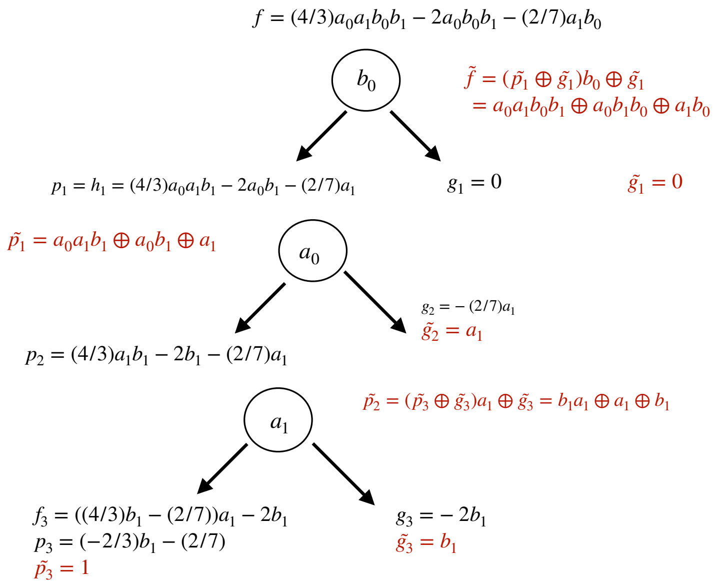

Fig. 1: Recursion tree and the $\widetilde{f_{\mathbb{B}}}$ computation for Example V.1.

of recursive calls is often much less than 2^n. Also note that in line 15 of the algorithm, $\widetilde{f_{\mathbb{B}}} = (\widetilde{p} \oplus \widetilde{g}) \cdot x_i \oplus \widetilde{g}$ resembles a positive Davio decomposition. The positive Davio decomposition decomposes a Boolean function $\widetilde{f}$ based on its cofactors $\widetilde{f_x} = \widetilde{f}(x = 1)$ and $\widetilde{f_{x'}} = \widetilde{f}(x = 0)$ as: $\widetilde{f} = (\widetilde{f_x} \oplus \widetilde{f_{x'}}) \cdot x \oplus \widetilde{f_{x'}}$. In our case, $\widetilde{p}, \widetilde{g}$ correspond to the positive and negative cofactors of $\widetilde{f_{\mathbb{B}}}$ w.r.t. x_i, respectively.

A. *Implementation*

Our recursive algorithm is a stand-alone software program implemented in C++. We have built a custom polynomial data structure for a polynomial $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$ where C_i are coefficients and X_i are monomials, and f is defined as a list of terms. A term is implemented as a typedef structure of coefficients (c_i) and monomials (X_i), a monomial is a vector of tuples of the form $[x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}]$ where $\alpha_i \in 0, 1$. A coefficient is a typedef structure of sign, numerator and denominator. We have also implemented functions to impose a lex term order for a given polynomial as well as a function to perform multivariate polynomial division. At every recursion level, we divide the polynomial f by x_i and obtain the p and g polynomials, and compute the $\tilde{p}$ and $\tilde{g}$. We recombine $\tilde{p}$ and $\tilde{g}$ to obtain $\tilde{f} = ((\tilde{p} \oplus \tilde{g})x_i) + \tilde{g}$. This stand-alone tool is used for experiments.

Two versions of the tool are implemented where the $\widetilde{p}, \widetilde{g}$, and $\widetilde{f_{\mathbb{B}}}$ are computed using: i) explicit set representations using the polynomial data-structure described above; and ii) using implicit representations, particularly the ROBDD representation using the CUDD package [11].

VI. Experiments

Using our tool, we have conducted some experiments to compute a Boolean function $\widetilde{f_{\mathbb{B}}}$, given a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$. The polynomials f have been taken from the work of [6], [8], and [7]. These approaches address the problem of computing rectification functions from buggy integer multiplier circuits. The tools developed as part of [6] [8] [7] perform symbolic algebra based computations to compute a rectification polynomial f to patch a buggy circuit. These tools integrate a Gröbner basis reduction on circuits using *amulet* [2] and *revsca* [3], with an extended Gröbner Basis computation using the Singular computer algebra tool [17]. Once a rectification polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$ is computed, [7] performs a *reduced* Gröbner basis computation $GB(f, J_0)$ to compute $\widetilde{f_{\mathbb{B}}}$, which is computationally very prohibitive. Our objective is to replace the expensive $GB(f, J_0)$ computation with our $Poly\mathbb{Q}to\mathbb{F}_2(f)$ algorithm.

The experiments are conducted on a desktop computer with a 3.5GHz Intel CoreTM i7-4770K Quad-core CPU, 16 GB RAM, running 64-bit Linux OS. Table II presents the results on computing a Boolean function corresponding to rectification polynomials for the following multiplier structures: i) a multiplier with simple partial product generators, array multiplier architecture with a ripple-carry adder in the final stage, denoted *sp-ar-rc*, and ii) an architecture with simple partial product generation, Wallace tree structure and a carry lookahead adder in the final stage of the design, denoted *sp-wt-cl*. The columns denote the datapath size of the faulty benchmarks, the number of terms in input polynomial f, the number of variables in f, the number of terms in the output polynomial in $\mathbb{F}_2$ (explicit approach), the number of BDD nodes (implicit approach), and the execution time taken by our recursive algorithms for both approaches, the maximum number of recursive calls, and the time taken by GB based approach [7].

As it can be seen from the results, the ROBDD based implementation outperforms the explicit approach, as well as the GB-based approach. For example, consider the second row in SP-AR-RC structure, a case of a rectification polynomial

TABLE II: Experimental results to compute a Boolean function $\widetilde{f_{\mathbb{B}}}$ from a polynomial f with coefficients in $\mathbb{Q}$; n = operand width/word-length of benchmark multiplier designs; t = time taken to for the proposed recursive algorithm, in seconds; d = max recursive calls; GBC = time taken to compute a rectification function over $\mathbb{F}_2$ with a GB based approach [7], Time-Out (TO) = 15000s, NA = a rectification polynomial f in $\mathbb{Q}$ couldn't be computed by [8] [7].

Benchmarks	*n*	**Polynomials over $\mathbb{Q}$** (input data)		**Polynomials over $\mathbb{F}_2$** (output: explicit representation)			**BDD Construction** (output: implicit representation)		*d*= # of recursive calls	*GBC*
		# of Terms	# of vars	# of Terms	# of vars	t (sec)	# of BDD nodes	t (sec)		
SP-AR-RC	4	22	8	14	8	0.1	23	0.06	82	1.2
	8	14338	16	6254	16	128.3	5326	81.96	68484	TO
	16	718	32	702	32	24	1327	11.8	11619	7696
	32	226	64	190	64	16.5	654	8.9	4944	0.2
	64	3	4	3	4	0.1	7	0.01	4	0.5
	128	14	15	32767	15	12925	16	85	16383	TO
	256	16	17	TO	TO	TO	18	751.74	65535	TO
SP-WT-CL	4	82	8	12	8	0.3	93	0.2	1029	0.3
	8	492	16	101	16	2.1	147	1.6	1291	3.7
	16	28099	28	16009	28	10436	291	4569	2921618	1872
	32	NA	NA	NA	NA	NA	NA	NA	NA	NA
	64	3	2	1	2	0.009	3	0.006	2	1.8

computed for a small 8-bit multiplier. The GB based approach of [7] fails when the size of the polynomial f is large. However, our recursion computes the desired Boolean function orders of magnitude faster. In case of the 256-bit circuit, our explicit approach failed to compute $\widetilde{f_{\mathbb{B}}}$ in the stipulated time, whereas the ROBDD was constructed. This is because when we reconstruct $\widetilde{f_{\mathbb{B}}}$ in Alg. 1 line 15, there is scope for logic simplification to reduce the number of terms. However, our explicit approach does not employ that simplification, so it unnecessarily processes more terms, which takes more time. For the entries are marked with NA, the approaches of [8] [7] could not compute a rectification polynomial f due to a potentially very large size of f.

VII. Conclusion

This paper has presented a new problem and approach to synthesize Boolean logic functions from multivariate polynomials with coefficients in the field of fractions ($\mathbb{Q}$), where the variables take only Boolean values. We have presented a recursive algorithm that takes as input a polynomial f over $\mathbb{Q}$ with binary variables, and derives a corresponding polynomial $\widetilde{f}$ over the finite field ($\mathbb{F}_2$) of two elements, such that $\widetilde{f}$ has the same variety (zero-set) as f. As $\mathbb{F}_2$ is isomorphic to Boolean algebra, $\widetilde{f}$ can be translated to a Boolean network or function by replacing the products and sums to AND and XOR operators, respectively. We have proved the existence of $\widetilde{f}$, and have developed an algorithm to perform this transformation. We have shown that our approach results in a positive Davio decomposition of $\widetilde{f_{\mathbb{B}}}$. We have applied our approach to translate rectification polynomials to Boolean functions and compared against GB-based methods. Our approach is implemented using both explicit and implicit set representations. Our ROBDD based implementation is orders of magnitude faster than that of a GB-based approach, particularly for large circuits and polynomials.

References

[1] D. Ritirc, A. Biere, and M. Kauers, "Column-Wise Verification of Multipliers Using Computer Algebra," in *Formal Methods in Computer-Aided Design (FMCAD)*, 2017, pp. 23–30.

[2] D. Kaufmann and A. Biere, "Amulet 2.0 for verifying multiplier circuits," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 12652, pp. 357 – 364, 2021.

[3] A. Mazhoon, D. Große, and R. Drechsler, "PolyCleaner: Clean your Polynomials before Backward Rewriting to Verify Million-Gate Multipliers," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

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

[5] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[6] 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 *Formal Methods in Computer-Aided Design (FMCAD)*, 2018, pp. 1–9.

[7] V. Rao, H. Ondricek, P. Kalla, and F. Enescu, "Rectification of integer arithmetic circuits using computer algebra techniques," in *IEEE International Conference on Computer Design (ICCD)*, Oct. 2021, pp. 186–195.

[8] A. Srinath, "Rectification of Integer Arithmetic Circuits," Master's thesis, ECE Dept., The University of Utah, USA, 2019.

[9] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams," in *EDAC*, 92, pp. 43–47.

[10] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Order Kronecker Function Decision Diagrams," in *DAC*, 1994, pp. 415–419.

[11] F. Somenzi, "Colorado Decision Diagram Package," Computer Programme, 1997.

[12] 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.

[13] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design," *Proceedings of the IEEE*, 2015.

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

[15] U. Gupta, I. Ilioaea, V. Rao, A. Srinath, P. Kalla, and F. Enescu, "Rectification of Arithmetic Circuits with Craig Interpolants in Finite Fields," in *VLSI-SoC: Design and Engineering of Electronics Systems Based on New Computing Paradigms*. Springer International Publishing, June 2019, vol. 561, ch. 5, pp. 79–106.

[16] N. A. Sabbagh and B. Alizadeh, "Arithmetic Circuit Correction by Adding Optimized Correctors Based on Groebner Basis Computation," in *Proc. Eur. Test Symp. (ETS)*, 2021, pp. 1–6.

[17] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "Singular 4-1-0 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2016.