Logic Synthesis from Polynomials with Coefficients in the Field of Rationals

Bhavani Sampathkumar¹, Bailey Martin¹, Ritaja Das¹, Priyank Kalla¹, and Florian Enescu²

¹Electrical & Computer Engineering, University of Utah, Salt Lake City, UT, USA
²Mathematics & 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 $\tilde{f}$ over the finite field ($\mathbb{F}_2$) of two elements, such that $\tilde{f}$ has the same variety (zero-set) as $f$. As $\mathbb{F}_2$ is isomorphic to Boolean algebra, $\tilde{f}$ 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 translation $\tilde{f} \in \mathbb{F}_2$ resulting in 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 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. While our approach relies upon concepts from algebraic geometry, it turns out that the (de)composition of $\tilde{f}_B$ resembles a positive Davio decomposition, as implemented in functional decision diagrams [9] [10]. We implement our approach to transform $f$ to $\tilde{f}_B$ as a recursive algorithm, with both an explicit representation of $\tilde{f}$ as a set of monomial terms, and also with an implicit set representation using the CUDD decision diagrams package [11].

Example 1.1. 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 $\tilde{f} = a_0a_1b_0b_0+a_0b_0b_1+a_1b_0$ 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 \( x \) where \( f(x) = 0 \), we have \( \tilde{f}(x) = 0 \). Whereas when \( f(x) \neq 0 \), we have \( \tilde{f}(x) = 1 \). Then, a Boolean function can be generated from \( f \) as \( f_\mathbb{B} = (a_0a_1b_0b_1) \oplus (a_0b_0b_1) \oplus (a_1b_0) \).

\[
\begin{array}{cccc|c|c}
\hline
\{a_1a_0b_0b_1\} & f & \{a_1a_0b_0b_1\} & f & f \\
\hline
0000 & 0 & 0 & 1000 & 0 & 0 \\
0001 & 0 & 0 & 1001 & -27 & 1 \\
0010 & 0 & 0 & 1010 & -27 & 1 \\
0011 & 0 & 0 & 1011 & -27 & 1 \\
0100 & 0 & 0 & 1100 & 0 & 0 \\
0101 & 0 & 0 & 1101 & -27 & 1 \\
0110 & 0 & 0 & 1110 & 0 & 0 \\
0111 & -2 & 11 & 1111 & -2021 & 1 \\
\hline
\end{array}
\]

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

Paper Organization: The following section, Section III covers the notation used and the preliminary background. Section IV reviews related previous work. Section V formally states the problem, proves the existence of such a translation of \( f \) in \( \mathbb{Q} \) to \( \tilde{f} \) in \( \mathbb{F}_2 \). The algorithm and implementation is described in Section VI whereas Section VII concludes the experimental results. Section VIII 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 \( \{x_1, …, x_n\} \) denote a set of variables. Let \( \mathbb{K} \) be any field, e.g. \( \mathbb{K} = \mathbb{Q} \) or \( \mathbb{K} = \mathbb{F}_2 \); then \( R = \mathbb{K}[x_1, …, x_n] \) denotes the ring of polynomials in variables \( x_1, …, x_n \), with coefficients in \( \mathbb{K} \). A polynomial \( f \in R \) is written as a sum of terms, \( f = c_1X_1 + c_2X_2 + … + c_iX_i \), 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 \( x_1^\alpha \cdot x_2^\beta \cdot … \cdot x_n^\gamma \), with \( \alpha_i \in \mathbb{Z}_{\geq 0} \).

In our work, the variables \( \{x_1, …, x_n\} \) correspond to nets in a circuit, and the gates of the circuit are modeled using a set of polynomials \( F = \{f_1, …, f_s\} \). Given a set of polynomials \( F = \{f_1, …, f_s\} \) from \( R \), the ideal generated by \( F \) is \( J = \langle F \rangle = \{h_1f_1 + h_2f_2 + … + h_sf_s \mid h_1, …, h_s \in R \} \). The polynomials \( f_1, …, f_s \) form the basis of ideal \( J \).

Let \( \mathbb{K} = (x_1, …, x_n) \in \mathbb{K}^n \) be a point in the affine space, and \( f \) a polynomial in \( R \). If \( f(\mathbb{K}) = 0 \), we say that \( f \) vanishes on \( \mathbb{K} \). 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: \( V_\mathbb{K}(J) = V_\mathbb{K}(f_1, …, f_s) = \{\mathbb{K} \in \mathbb{K}^n : \forall f \in J, f(\mathbb{K}) = 0 \} \).

An ideal may have many different sets of generators that have the same variety; i.e. it is possible to have \( J = \langle f_1, …, f_s \rangle = \langle g_1, …, g_t \rangle \). Such \( V_\mathbb{K}(f_1, …, f_s) = V_\mathbb{K}(g_1, …, g_t) \). A Gröbner basis (GB) of an ideal is one such generating set \( G = \{g_1, …, g_t\} \), 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 \( \{f_1, …, f_s\} \) and computes its GB \( G = \{g_1, g_2, …, g_t\} \).

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

In this work, we consider \( f \in R = \mathbb{Q}[x_1, …, x_n] \), where \( f \) is derived from a circuit, so \( x_1, …, x_n \) take only Boolean values 0 or 1. Equivalently, each variable \( x_i \) is idempotent in that \( x_i^2 = x_i \) or \( x_i^2 = \bar{x}_i \). We call the polynomial \((x_i^2 - x_i)\) a Boolean vanishing polynomial (BVP) as it vanishes on \( x_i = 0, 1 \). Let \( F_0 = \{x_i^2 - x_i, …, x_n^2 - x_n\} \) be the set of all BVPs in \( \mathbb{Z}[x_1, …, x_n] \), and \( J_0 = \langle F_0 \rangle \). Note that there are natural maps from \( \mathbb{Z}[x_1, …, x_n] \) to \( \mathbb{Q}[x_1, …, x_n] \) and to \( S = \mathbb{F}_2[x_1, …, x_n] \), which lead to the ideals \( F_0R \) in \( R \) and \( F_0S \) in \( S \), generated by \( F_0 \) and \( F_0S \) in \( S \). In other words, the same set of BVPs \( x_i^2 - x_i : i = 1, …, n \) 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 \((\text{mod } J_0)\), i.e. \( f \) is divided by BVPs for all variables and the resulting remainder \( r \) is taken as the result: denoted \( f \) \((\text{mod } J_0) = f \) \((\text{mod } J_0) = x_i^2 - x_i \). Thus, our computations are effectively performed over the quotient ring \( \mathbb{F}_2[x_1, …, x_n] \), where \( F_0R = \langle x_i^2 - x_i, …, x_n^2 - x_n \rangle \) is the ideal generated by all BVPs in \( R \). As a result, higher degree variables \( x_i^2, k \geq 2 \), are reduced to \( x_i \) in the computations since \( x_i^2 = x_i \) (mod \( J_0 \)). Thus \( f \) comprises terms with only multilinear monomials \( x_1^\alpha \cdot x_2^\beta \cdot … \cdot x_n^\gamma \), with \( \alpha_i \in \{0, 1\} \).

In other words, the Boolean idempotency on variables of \( f \) is enforced by considering the ideal \( (f, F_0R) \). Moreover, the variety \( V_\mathbb{K}(J, F_0R) \subset \{0, 1\}^n \). Note that in Example \( 1 \) \( f = (4/3)\alpha_0a_1b_1b_0 - 2a_0b_0b_1 - (2/7)ab_0 \) has multilinear monomials, as it is already reduced \((\text{mod } F_0R = \langle a_0^2 - a_0b_0 - b_0, a_1^2 - a_1, b_1^2 - b_1 \rangle) \). In the sequel, we will assume that every polynomial is reduced to its multilinear form \((\text{mod } F_0R) \).

III. REVIEW OF RELATED WORK

Previous work has investigated the modeling of arithmetic circuits using commutative algebra and algebraic geometry. The work of \([11\, 12\, 13\) 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 \([15\, 16\, 17\, 18\), among others. Of these \([15\, 16\) 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 [6], 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 [3], and [7]. In [3], 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 $\mathbb{Q}[x_1, \ldots, x_n]$. 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 $f_{\bar{f}}$ from a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$, such that $f$ and $f_{\bar{f}}$ have the same zero-set. Their approach is as follows: i) They compute a reduced Gröbner basis $G = GB(f, J_0) = \langle g_1, \ldots, g_i \rangle$. ii) Then, the polynomial $\bar{f} \in \mathbb{F}_2[x_1, \ldots, x_n]$ is computed using the formula

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

Then $\bar{f}$ 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 ineffective 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 $f$ from $f_{\bar{f}}$.

This paper addresses the above problem by relating the variety $V_{\bar{f}}(f, F_0R)$ and $V_{\bar{f}}(f, F_0S)$. We do not explain how a rectification polynomial computation may generate $f$ with rational coefficients. We refer the reader to [3] (Chapter 6) and [7] for more details. We only address the problem of synthesizing Boolean functions $f_{\bar{f}}$ from $f$.

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

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

$$V_{\bar{f}}(f, F_0R) = V_{\bar{f}}(f, F_0S).$$

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, $V_{\bar{f}}(f, F_0R) = V_{\bar{f}}(f, F_0S)$ implies that: i) for all points $\bar{x} \in V_{\bar{f}}(f, F_0R)$ we have $f(\bar{x}) = f(\bar{x}) = 0$; and ii) for all points $\bar{x} \notin V_{\bar{f}}(f, F_0R)$, we have that $f(\bar{x}) = 1$, as $\bar{f}$ only evaluates to 0 or 1.

**Approach:** We describe the transformation approach as follows. To begin with, assume that $f \in R = \mathbb{Q}[x_1, \ldots, x_n]$ is reduced (mod $F_0R$), 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 $x_1 = x_1$ 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 $h, g \in \mathbb{Q}[x_2, \ldots, x_n]$ and $h \neq 0$. We state and prove the following result.

**Theorem IV.1** (The Translation Theorem). Let $f$ be a polynomial in $R = \mathbb{Q}[x_1, \ldots, x_n]$, written in the form $f = hx_1 + g$ as shown above. Then there exists a polynomial $\bar{f} \in \mathbb{F}_2[x_1, \ldots, x_n]$ where $f$ and $\bar{f}$ have the same variety as $\{0, 1\}$-tuples, i.e. $V_{\bar{f}}(f, F_0R) = V_{\bar{f}}(f, F_0S)$.

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

$$f(x_1, y) = 0$$

$$\Rightarrow h(y) \cdot x_1 + g(y) = 0$$

$$\Rightarrow h(y) \cdot x_1 = -g(y).$$

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

$$V_{\bar{f}}(f, F_0R) = \{(1, y) : y \in V_{\bar{f}}(h + g, F_0R)\} \cup \{(0, y) : y \in V_{\bar{f}}(g, F_0R)\}.$$ 

In other words, $V_{\bar{f}}(f, F_0R)$ consists of tuples $(1, y)$ where $h + g$ vanishes on $y$, and tuples $(0, y)$ where $g$ vanishes on $y$.

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 $f = 0$.

2) Case 2: $h \neq 0, g = 0$. Then $f = x_1$, because $V_{\bar{f}}(f = hx_1, F_0R) = V_{\bar{f}}(f = x_1, F_0S) = \{(0)\}$.

3) Case 3: $h = 0, g \neq 0$. Then $f = 1$, 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 $f = x_1 + 1$; otherwise, $f = 1$.

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

Now assume that the theorem statement is true for polynomials with up to and including $n - 1$ variables. Therefore, we can find $\bar{p} \in \mathbb{F}_2[2, x_2, \ldots, x_n]$ that satisfy the following:
1) The polynomial $\tilde{p}$ is such that $V_{F_2}(\tilde{p}, F_0 S) = V_{Q}(h + g, F_0 R) \subseteq \{0, 1\}^{n-1}$, i.e. the varieties of $\tilde{p}$ and $h + g$ are equal as $(0,1)$-tuples. That is, for a polynomial $h + g$ in $n-1$ variables, there exists a corresponding $\tilde{p}$ with coefficients in $F_2$.

2) The polynomial $\tilde{g}$ is such that $V_{F_2}(\tilde{g}, F_0 S) = V_{Q}(g, F_0) \subseteq \{0, 1\}^{n-1}$.

Let $\tilde{h} = \tilde{p} - \tilde{g}$. As above, considering the polynomial $\tilde{f} = \tilde{h} x_1 + \tilde{g} = (\tilde{p} - \tilde{g}) x_1 + \tilde{g}$ over $F_2$, and a point $(x_1, y) \in V_{F_2}(\tilde{f}, F_0 S) \subseteq \{0, 1\}^n$, we see that:

$$\tilde{f}(x_1, y) = 0$$

$$\implies (\tilde{p}(y) - \tilde{g}(y)) \cdot x_1 = -\tilde{g}(y)$$

$$\implies (\tilde{p}(y) - \tilde{g}(y)) \cdot x_1 = \tilde{g}(y).$$

Accordingly, as $-1 = +1 \pmod{2}$, therefore, $x_1 = 1 \implies \tilde{p}(y) - \tilde{g}(y) = \tilde{g}(y)$

$$\implies \tilde{p}(y) = 0$$

$$\implies y \in V_{F_2}(\tilde{p}, F_0).$$

$$x_1 = 0 \implies \tilde{p}(y) = 0$$

$$\implies y \in V_{F_2}(\tilde{g}, F_0).$$

In conclusion,

$$V_{F_2}(\tilde{f}, F_0 S) = \{(1, y) : y \in V_{F_2}(\tilde{p}, F_0 S)\} \cup \{(0, y) : y \in V_{F_2}(\tilde{g}, F_0 S)\}.$$}

Algorithm 1 shows the recursive procedure. Lines 2-7 show the terminal cases of recursion. Lines 9-12 decompose $f = h x_1 + g$, and 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 $\tilde{p}, \tilde{g}$ are returned by the recursive calls in lines 13-14, the Boolean function $\tilde{f}_B$ (or equivalently the polynomial $\tilde{f} \in F_2[x_1, \ldots, x_n]$) is constructed in line 15, where $\oplus$ denotes addition (mod 2), or the XOR operation.

![Algorithm 1: Compute $f_B$ from $f$](image)

**Example V.1.** We demonstrate the algorithm on the polynomial $f = (4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1 b_0$. The highest activity variable is $b_0$, so we factorize $f = ((4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1 b_0).$ We denote by subscript $i$, the decomposed polynomials $p, g_i$ at recursion-level $i$.

At the first recursion level: $h_1 = (4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1$, and $g_1 = 0$. Then $p_1 = h_1 + 0 = (4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1.$ Since $g_1 = 0, g_1 = 0$.

At second recursion level, $f = p_1 = (4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1.$ Expansion variable selected is $a_0$. Then $h_2 = (4/3) a_0 a_1 b_1 - 2a_0 b_0 - (2/7) a_1$. Since $g_2 = 0, g_2 = 0$.

Fig. 1 depicts the recursion tree. The returned Boolean functions $\tilde{p}, \tilde{g}$ at each recursion step are shown in red color. The final returned Boolean function $\tilde{f}_B = (a_0 a_1 b_0 b_1) \oplus (a_0 b_0 b_1) \oplus (a_1 b_0)$, which matches the one in Example 1.
of recursive calls is often much less than $2^n$. Also note that in line 15 of the algorithm, $f = (\tilde{p} \oplus \tilde{g}) \cdot x_i \oplus \tilde{g}$ resembles a positive Davio decomposition. The positive Davio decomposition decomposes a Boolean function $\tilde{f}$ based on its cofactors $\tilde{f}_x = \tilde{f}(x = 1)$ and $\tilde{f}_x' = \tilde{f}(x = 0)$ as: $\tilde{f} = (\tilde{f}_x \oplus \tilde{f}_x') \cdot x \oplus \tilde{f}_x'$. In our case, $\tilde{p}, \tilde{g}$ correspond to the positive and negative cofactors of $f$ 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 atypedef 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 $\tilde{p}, \tilde{g}$, and $\tilde{f}$ 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 $\tilde{f}$, 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 $\tilde{f}$, which is computationally very prohibitive. Our objective is to replace the expensive $GB(f, J_0)$ computation with our $\text{PolyQto}\mathbb{R}_2(f)$ 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 \( \tilde{f} \) 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 \([8]\) \([7]\).

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>( n )</th>
<th>Polynomials over ( \mathbb{Q} ) (input data)</th>
<th>Polynomials over ( \mathbb{F}_2 ) (output: explicit representation)</th>
<th>BDD Construction (output: implicit representation)</th>
<th>( d ) = # of recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of Terms # of # of BDD t # of t # of g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Terms vars</td>
<td></td>
<td></td>
<td>nodes</td>
</tr>
<tr>
<td>SP-AR-RC</td>
<td></td>
<td>4 72</td>
<td>8</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 14338</td>
<td>16</td>
<td>6254</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 718</td>
<td>32</td>
<td>702</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32 226</td>
<td>64</td>
<td>190</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 14</td>
<td>15</td>
<td>22/67</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256 16</td>
<td>17</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>SP-WT-CL</td>
<td></td>
<td>4 82</td>
<td>8</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 492</td>
<td>16</td>
<td>101</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 20099</td>
<td>28</td>
<td>16009</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32 NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

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 \( \tilde{f}_B \) in the stipulated time, whereas the ROBDD was constructed. This is because when we reconstruct \( \tilde{f}_B \) 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 \( \tilde{f} \) over the finite field \( \mathbb{F}_2 \) of two elements, such that \( \tilde{f} \) has the same variety (zero-set) as \( f \). As \( \mathbb{F}_2 \) is isomorphic to Boolean algebra, \( \tilde{f} \) 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 \( \tilde{f} \), and have developed an algorithm to perform this transformation. We have shown that our approach results in a positive Davio decomposition of \( \tilde{f}_B \). 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


