Word-Level Polynomial Abstraction From Circuits Using Gröbner Bases

Tim Pruss

Graduate Student
Electrical and Computer Engineering, University of Utah
pruss.tim@gmail.com

Master's Thesis Proposal

Agenda

- Focus
  - Extraction of word-level representations of Galois field circuits
- Motivation
  - Galois fields, hardware applications & their abstraction
- Target problems
  - Given a Galois field $\mathbb{F}_{2^k}$ and circuit C, with k-bit inputs and outputs
  - Derive a polynomial representation for C over $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$
  - Word-level abstraction as a canonical polynomial representation
- Approach: **Computer Algebra Techniques**
  - Nullstellensatz + Gröbner basis methods + Elimination ordering
  - Challenge: Complexity of Gröbner basis algorithm
  - Proposed Contribution: An approach based on the FGLM algorithm to **obviate** the Gröbner basis computation for extraction of the canonical polynomial representation.

Motivation

- Wide applications of Galois field circuits
  - **Cryptography**: RSA, Elliptic Curve Cryptography (ECC)
  - Error Correcting Codes, Digital Signal Processing, etc.

- Bugs in hardware can leak secret keys [*Biham et al.*, "Bug Attacks", Crypto 2008]

- Data-path size in ECC crypto-systems can be very large
  - In $\mathbb{F}_{2^k}$, $k = 163, 233, \ldots$ (NIST standard)
  - ECC-point addition for encryption, decryption, authentication
  - Custom arithmetic architectures – hard to verify
  - Synthesized circuits are "easier" to verify

- Why use computer algebra?
  - Algebraic nature (finite field) of the computation (polynomial)
  - Abstraction infeasible with contemporary verification tools

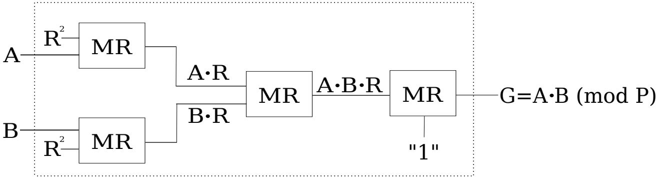


Figure : *Montgomery* multiplier over GF(2^k)

- Main operations in ECC rely on point additions and doubling operations on elliptic curves over Galois fields
- Multiplication and iterative squaring operations are usually implemented using custom-designed Galois field multipliers
- Given a hierarchically designed Montgomery multiplier, we will first extract polynomials AR, BR, ABR from the sub-circuit blocks.
- We can then apply our approach at a higher-level, to extract the function of the entire circuit.

Galois Field Overview

Galois field $\mathbb{F}_q$ is a finite field with q elements, $q = p^k$

- Commutative Ring with unity, associate, distributive laws
- Closure property: $+, -, \times$, inverse $(\div)$
- $\mathbb{F}_p \equiv (\mathbb{Z} \pmod{p})$, where $p =$ prime, is a field
  - $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

Our interest: $\mathbb{F}_q = \mathbb{F}_{2^k}$, i.e. $q = 2^k$

- $\mathbb{F}_{2^k}$: k-dimensional extension of $\mathbb{F}_2$
  - k-bit bit-vector, AND/XOR arithmetic

To construct $\mathbb{F}_{2^k}$

- $\mathbb{F}_{2^k} \equiv \mathbb{F}_2[x] \pmod{P(x)}$
- $P(x) \in \mathbb{F}_2[x]$, irreducible polynomial of degree k

Field Elements: *e.g.* $\mathbb{F}_8$

Consider: $\mathbb{F}_{2^3} = \mathbb{F}_2[x] \pmod{x^3+x+1}$
$A \in \mathbb{F}_2[x]$
$A \pmod{x^3+x+1} = a_2x^2 + a_1x + a_0$. Let $P(\alpha) = 0$:

- $\langle a_2, a_1, a_0 \rangle = \langle 0,0,0 \rangle = 0$
- $\langle a_2, a_1, a_0 \rangle = \langle 0,0,1 \rangle = 1$
- $\langle a_2, a_1, a_0 \rangle = \langle 0,1,0 \rangle = \alpha$
- $\langle a_2, a_1, a_0 \rangle = \langle 0,1,1 \rangle = \alpha + 1$
- $\langle a_2, a_1, a_0 \rangle = \langle 1,0,0 \rangle = \alpha^2$
- $\langle a_2, a_1, a_0 \rangle = \langle 1,0,1 \rangle = \alpha^2 + 1$
- $\langle a_2, a_1, a_0 \rangle = \langle 1,1,0 \rangle = \alpha^2 + \alpha$
- $\langle a_2, a_1, a_0 \rangle = \langle 1,1,1 \rangle = \alpha^2 + \alpha + 1$

Multiplication in GF(2^4)

Input:
$A = (a_3 a_2 a_1 a_0)$
$B = (b_3 b_2 b_1 b_0)$
$A = a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3$
$B = b_0 + b_1 \cdot \alpha + b_2 \cdot \alpha^2 + b_3 \cdot \alpha^3$

Irreducible Polynomial:
$P = (11001)$
$P(x) = x^4 + x^3 + 1, \quad P(\alpha) = 0$

Result:
$A \times B \pmod{P(x)}$

Multiplication over GF(2^4)

$$
\begin{array}{ccccccc}
 & & & a_3 & a_2 & a_1 & a_0 \\
\times & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & a_3 \cdot b_0 & a_2 \cdot b_0 & a_1 \cdot b_0 & a_0 \cdot b_0 \\
 & & a_3 \cdot b_1 & a_2 \cdot b_1 & a_1 \cdot b_1 & a_0 \cdot b_1 & \\
 & a_3 \cdot b_2 & a_2 \cdot b_2 & a_1 \cdot b_2 & a_0 \cdot b_2 & & \\
a_3 \cdot b_3 & a_2 \cdot b_3 & a_1 \cdot b_3 & a_0 \cdot b_3 & & & \\
\hline
s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

In polynomial expression:
$S = s_0 + s_1 \cdot \alpha + s_2 \cdot \alpha^2 + s_3 \cdot \alpha^3 + s_4 \cdot \alpha^4 + s_5 \cdot \alpha^5 + s_6 \cdot \alpha^6$

S should be further reduced $\pmod{P(x)}$

$$
\begin{array}{ccc|cccc|cc}
s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & & \\
\hline
 & & & s_4 & 0 & 0 & s_4 & \Leftarrow & s_4 \cdot \alpha^4 \pmod{P(\alpha)} \\
 & & & s_5 & 0 & s_5 & s_5 & \Leftarrow & s_5 \cdot \alpha^5 \pmod{P(\alpha)} \\
 & & + & s_6 & s_6 & s_6 & s_6 & \Leftarrow & s_6 \cdot \alpha^6 \pmod{P(\alpha)} \\
\hline
 & & & g_3 & g_2 & g_1 & g_0 & &
\end{array}
$$

$$s_4 \cdot \alpha^4 \pmod{\alpha^3 + \alpha + 1} = s_4 \cdot \alpha^3 + s_4$$
$$s_5 \cdot \alpha^5 \pmod{\alpha^3 + \alpha + 1} = s_5 \cdot \alpha^3 + s_5 \cdot \alpha + s_5$$
$$s_6 \cdot \alpha^6 \pmod{\alpha^3 + \alpha + 1} = s_6 \cdot \alpha^3 + s_6 \cdot \alpha^2 + s_6 \cdot \alpha + s_6$$

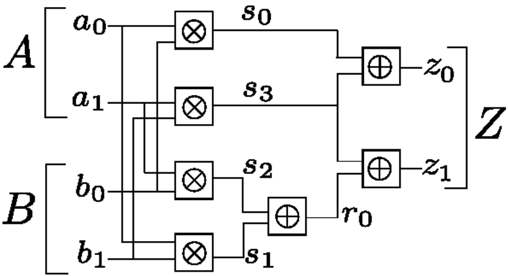
$$G = g_0 + g_1 \cdot \alpha + g_2 \cdot \alpha^2 + g_3 \cdot \alpha^3$$

A Mathematical Problem

- Given circuit implementation C of a function $f : Y = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$ and given $P(x)$, s.t. $P(\alpha) = 0$.
  - Primary Input: $A = \{a_0, \dots, a_{k-1}\}$
    - Note: we allow multiple primary inputs.
    - i.e. $f : Y = \mathcal{F}(A, B, \dots)$
  - Primary Output $Z = \{z_0, \dots, z_{k-1}\}$
  - $A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{k-1}\alpha^{k-1}$
  - $Z = z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1}$
- What is the specification $Y = \mathcal{F}(A)$ of C?

Mathematically:

- Model the circuit (gates) as polynomials $\{f_1, \dots, f_s\} \in \mathbb{F}_{2^k}[x_1, \dots, x_d]$
- Polynomial specification becomes $Y + \mathcal{F}(A)$
- $Y + \mathcal{F}(A)$ vanishes on the variety of $V(f_1, \dots, f_s)$



Model circuit as polynomials in $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$:

$$z_0 = s_0 + s_3 \quad \Longrightarrow \quad f_1 : z_0 + s_0 + s_3$$

$$s_0 = a_0 \cdot b_0 \quad \Longrightarrow \quad f_2 : s_0 + a_0 \cdot b_0$$

$$\vdots$$

$$A + a_0 + a_1\alpha, \quad B + b_0 + b_1\alpha, \quad Z + z_0 + z_1\alpha$$

Let $\mathbb{F}_q = GF(2^k)$:

- $\mathbb{F}_q[x_1, \ldots, x_n]$: ring of all polynomials with coefficients in $\mathbb{F}_q$
- Given a set of polynomials:
  - $f, f_1, f_2, \ldots, f_s \in \mathbb{F}_q[x_1, \ldots, x_n]$
  - Find solutions to $f_1 = f_2 = \cdots = f_s = 0$
- Variety: Set of ALL solutions to a given system of polynomial equations: $V(f_1, \ldots, f_s)$
  - In $\mathbb{R}[x, y]$, $V(x^2 + y^2 - 1) = \{all\ points\ on\ circle : x^2 + y^2 - 1 = 0\}$
  - In $\mathbb{R}[x]$, $V(x^2 + 1) = \emptyset$
  - In $\mathbb{C}[x]$, $V(x^2 + 1) = \{(\pm i)\}$
- Variety depends on the ideal generated by the polynomials.
- Reason about the Variety by analyzing the Ideals

Definition

Ideals of Polynomials: Let $f_1, f_2, \ldots, f_s \in \mathbb{F}_q[x_1, \ldots, x_n]$. Let

$$J = \langle f_1, f_2 \ldots, f_s \rangle = \{f_1 h_1 + f_2 h_2 + \cdots + f_s h_s\}$$

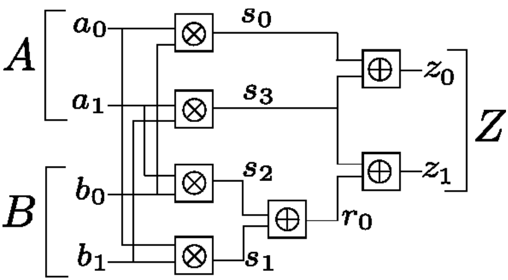
$J = \langle f_1, f_2 \ldots, f_s \rangle$ is an ideal generated by $f_1, \ldots, f_s$ and the polynomials are called the generators.

Definition

Vanishing Ideal: For any subset V of $\mathbb{F}_q^d$, the ideal of polynomials that vanish on V, called the *vanishing ideal of* V, is defined as:
$I(V) = \{f \in \mathbb{F}_q[x_1, \ldots, x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$.

If a polynomial f vanishes on a variety V, then $f \in I(V)$.

Example Formulation



- Polynomials for all the gates: $f_1, \ldots, f_s$; ideal $J = \langle f_1, \ldots, f_s \rangle$
- Circuit polynomial function: $f : Z = A \times B$
- f "agrees with" all solutions to $f_1 = \cdots = f_s = 0$
- f vanishes on variety $V_{\mathbb{F}_{2^k}}(J)$?

Definition

Vanishing Polynomials: Polynomials of the form $\{x^q - x\}$ over $\mathbb{F}_q$. Let $F_0 = \{x_1^q - x_1, \ldots,\ x_d^q - x_d\}$, then $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d\rangle$ is the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$.

- For any Galois field $\mathbb{F}_q$, let $J \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal, and let $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d\rangle$ be the ideal of all vanishing polynomials.
- Let $V_{\mathbb{F}_q}(J)$ denote the variety of J over $\mathbb{F}_q$.
- Then, $I(V_{\mathbb{F}_q}(J)) = J + J_0 = J + \langle x_1^q - x_1, \ldots,\ x_d^q - x_d\rangle$.

Given circuit C which implements a polynomial function $Y = \mathcal{F}(A)$ over $\mathbb{F}_q[x_1, \ldots, x_n]$

- $Y + \mathcal{F}(A)$ is the polynomial specification of C
- $J = \langle f_1, f_2 \ldots, f_s \rangle$, Polynomials from the design
- $J_0 = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$, Vanishing polynomials generated
- $J + J_0 = \langle f_1, f_2 \ldots, f_s, \;\; x_1^q - x_1, \ldots, x_n^q - x_n \rangle$; Variety $V(J + J_0) =$ circuit configuration
- $Y + \mathcal{F}(A) \in J + J_0$

Our problem: Find the polynomial specification $Y + \mathcal{F}(A) \in (J + J_0)$

Requires the computation of a Gröbner basis of $J + J_0$

Specification Abstraction Requires a Gröbner Basis

- Different generators can generate the same ideal
- $\langle f_1, \cdots, f_s \rangle = \cdots = \langle g_1, \cdots, g_t \rangle$
- Some generators are a "better" representation of the ideal
- A **Gröbner basis** is a "canonical" representation of an ideal

Given $F = \{f_1, f_2, \cdots, f_s\}$, Compute a Gröbner Basis $G = \{g_1, g_2, \cdots, g_t\}$, such that $I = \langle F \rangle = \langle G \rangle$

$$V(F) = V(G)$$

Buchberger's Algorithm

INPUT : $F = \{f_1, \ldots, f_s\}$
OUTPUT : $G = \{g_1, \ldots, g_t\}$
$G := F;$
REPEAT
  $G' := G$
  For each pair $\{f, g\}, f \neq g$ in G' DO
    $S(f, g) \xrightarrow{G'}_+ r$
    IF $r \neq 0$ THEN $G := G \cup \{r\}$
UNTIL $G = G'$

$$S(f, g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g$$

$L = \mathrm{LCM}(lm(f), lm(g))$, $lm(f)$: leading monomial of f

- For $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, Complexity $GB(J + J_0) : q^{O(n)}$
- GB complexity very sensitive to **term ordering**
- A term order has to be imposed for systematic polynomial computation

Let $f = 2x^2yz + 3xy^3 - 2x^3$

- LEX $x > y > z$: $\quad f = -\mathbf{2x^3} + 2x^2yz + 3xy^3$
- DEGLEX $x > y > z$: $\quad f = \mathbf{2x^2yz} + 3xy^3 - 2x^3$
- DEGREVLEX $x > y > z$: $\quad f = \mathbf{3xy^3} + 2x^2yz - 2x^3$

Recall, S-polynomial depends on term ordering:

$$S(f, g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g; \qquad L = \mathrm{LCM}(lm(f), lm(g))$$

- Let $J \subset \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal
- Let G be a Gröbner basis of J with respect to a lex ordering where $x_1 > x_2 > \cdots > x_d$.
- Then for every $0 \leq l \leq d$:
  - The set $G_l = G \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d]$ is a Gröbner basis of the lth elimination ideal J_l.

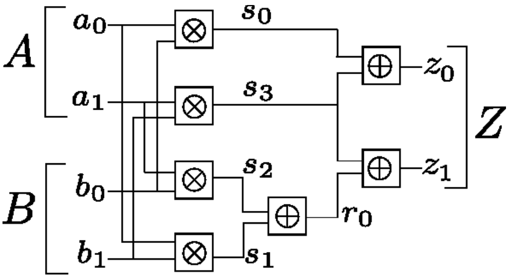
The lth elimination ideal does not contain variables $x_1, \ldots, x_l$, nor do the generators of it.

Elimination Term Ordering Example

- Let ideal $I = \langle f_1, f_2, f_3 \rangle$ where
  - $f_1 = x^2 + y + z - 1$
  - $f_2 = x + y^2 + z - 1$
  - $f_3 = x + y + z^2 - 1$
- The Gröbner basis of I with lex order $(x > y > z)$ is
  - $g_1 = x + y + z^2 - 1$
  - $g_2 = y^2 - y - z^2 + z$
  - $g_3 = 2yz^2 + z^4 - z^2$
  - $g_4 = z^6 - 4z^4 + 4z^3 - z^2$
- Notice that g_2 and g_3 only contain variables y and z
  - Eliminates variable x
- Similarly, g_4 only contains the variable z and eliminates x and y

Derived from applying elimination theorem to our problem set

- Given a circuit C implementing $Y = \mathcal{F}(A)$ over $\mathbb{F}_q$
- Using the variable order $x_1 > x_2 > \cdots > x_d > Y > A$
  - $x_1, \ldots, x_d$ are the circuit polynomials
- Impose a lex term order $>$ on the polynomial ring $R = \mathbb{F}_q[x_1, \ldots, x_d, Y, A]$.
- This elimination term order $>$ is defined as the **Abstraction Term Order**.
- If we compute a Gröbner basis G of ideal $(J + J_0)$ using the abstraction term order $>$
  - G will contain the vanishing polynomial $A^q - A$ as the only polynomial with only A as the support variable
  - G will contain a polynomial of the form $Y + \mathcal{F}(A)$
  - $Y + \mathcal{F}(A)$ is a unique, canonical, polynomial representation of C over $\mathbb{F}_q$

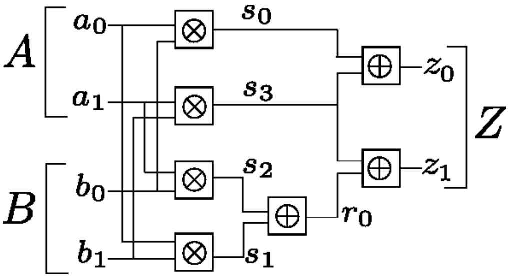


$(z_0 > z_1 > r_0 > s_0 > s_3 > s_1 > s_2 > a_0 > a_1 > b_0 > b_1 > Z > A > B)$

$$f_1 : s_0 + a_0 \cdot b_0; \quad f_2 : s_1 + a_0 \cdot b_1; \quad f_3 : s_2 + a_1 \cdot b_0; \quad f_4 : s_3 + a_1 \cdot b_1$$
$$f_5 : r_0 + s_1 + s_2; \quad f_6 : z_0 + s_0 + s_3; \quad f_7 : z_1 + r_0 + s_3; \quad f_8 : a_0 + a_1\alpha + A$$
$$f_9 : b_0 + b_1\alpha + B; \quad f_{10} : z_0 + z_1\alpha + Z$$

$J = \langle f_1, \ldots, f_{10} \rangle$

Abstraction Term Order Example

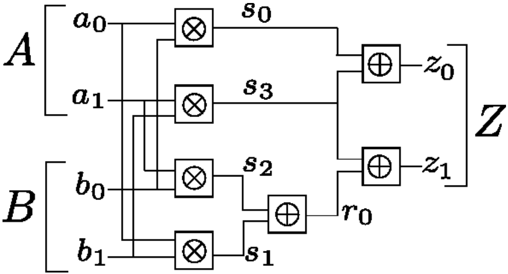


$(z_0 > z_1 > r_0 > s_0 > s_3 > s_1 > s_2 > a_0 > a_1 > b_0 > b_1 > Z > A > B)$

$$f_{11} : a_0^2 + a_0; \quad f_{12} : a_1^2 + a_1\alpha; \quad f_{13} : b_0^2 + b_0; \quad f_{14} : b_1^2 + b_1; \quad f_{15} : s_0^2 + s_0;$$
$$f_{16} : s_1^2 + s_1; \quad f_{17} : s_2^2 + s_2; f_{18} : s_3^2 + s_3; \quad f_{19} : r_0^2 + r_0; \quad f_{20} : z_0^2 + z_0$$
$$f_{21} : z_1^2 + z_1; \quad f_{22} : A^4 + A; \quad f_{23} : B^4 + B; \quad f_{24} : Z^4 + Z$$

$J_0 = \langle f_{11}, \ldots, f_{24} \rangle$

Abstraction Term Order Example



$(z_0 > z_1 > r_0 > s_0 > s_3 > s_1 > s_2 > a_0 > a_1 > b_0 > b_1 > Z > A > B)$
Compute the Gröbner basis, G, of $\{J + J_0\}$ with respect to abstraction term ordering $>$. $G = \{g_1, \ldots, g_{14}\}$

$$g_1 : B^4 + B; \quad g_2 : b_0 + b_1\alpha + B; \quad g_3 : a_0 + a_1\alpha + A; \quad g_4 : A^4 + A;$$
$$g_5 : s_0 + s_1\alpha + s_2(\alpha + 1) + Z; \quad g_6 : r_0 + s_1 + s_2; \quad g_7 : z_1 + r_0 + s_3$$
$$g_7 : z_0 + z_1\alpha + Z; \quad \mathbf{g_9} : \mathbf{Z} + \mathbf{A} * \mathbf{B}; \quad g_{10} : b_1 + B^2 + B; \quad g_{11} : a_1 + A^2 + A$$
$$g_{12} : s_3 + a_1 b_1; \quad g_{13} : s_2 + a_1 b_1\alpha + a_1 B; \quad g_{14} : s_1 + a_1 b_1\alpha + b_1 A$$

Table : Runtime of Gröbner Basis Computation

Word Size (k)	**Number of Polynomials (d)**	**Time (minutes)**
16	$1,871$	2.4
24	$3,135$	12
32	$5,549$	22.6
40	$8,587$	266
48	$12,327$	NA (Out of Memory)

- Mastrovito multiplier circuits
- Extract Boolean gate-level operators J and vanishing polynomials J_0
- Compute Gröbner basis of $J + J_0$ with respect to our term order $>$
  - Resulting Gröbner basis contains a polynomial $Y + A \times B$
- Gröbner basis computed using Singular's "slimgb" command
  - Run on a 64-bit Ubuntu machine with a 2.4GHz CPU and 8Gb of RAM
  - Unable to perform Gröbner basis computations of multipliers beyond 40-bit word inputs

FGLM Algorithm

- Takes as input a Gröbner basis, G_1, and two monomial term orderings, $>_a$ and $>_b$.
  - G_1 must be a Gröbner basis over term ordering $>_a$
- Converts G_1 to a Gröbner basis over term ordering $>_b$

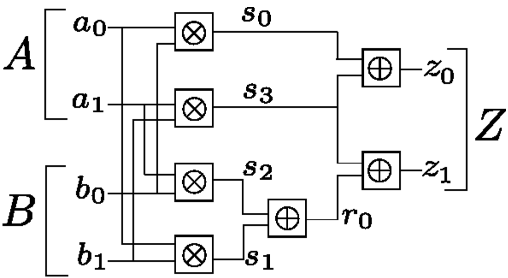
Applying FGLM to our approach:

- Given a circuit C which performs $Y = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$
- Find the Gröbner basis, G_1, of $\{J + J_0\}$ in a convenient term ordering
- Using FGLM, convert G_1 to a Gröbner basis, G_2, over abstraction term ordering $>$
- G_2 will contain a polynomial $Y + \mathcal{F}(A)$

Past contribution by Lv:

- Given a circuit C which implements $Y = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$
- Using the lex variable order $Y > A > x_1 > x_2 > \cdots > x_d$
  - Where $x_1 \ldots x_d$ are bit-level circuit variables in reverse topographical order
- $J + J_0$ is itself a Gröbner basis

No Gröbner basis computation necessary!
We denote this term order as $>_1$



$(Z > A > B > z_0 > z_1 > r_0 > s_0 > s_3 > s_1 > s_2 > a_0 > a_1 > b_0 > b_1)$

- Same J and J_0 as shown previously
- $J + J_0$ denote a Gröbner basis over term ordering $>_1$

Proposed Approach

Proposed approach to extract a unique polynomial representation $Y = \mathcal{F}(A)$ of circuits over F_{2^k} while obviating Gröbner basis calculation:

- Let C be a circuit performs the function $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$
- Extract the polynomials from the circuit (ideal J) and vanishing polynomials (ideal J_0)
- Assign the monomial ordering $>_1$; this makes $J + J_0$ a minimal Gröbner basis, G_1
- Use the FGLM algorithm to transform G_1 to a Gröbner basis, G_2, over abstraction ordering $>$
- G_2 will contain a polynomial in the form of $Y + \mathcal{F}(A)$, where Y is the word output and A is the word input of the circuit
    - $Y + \mathcal{F}(A)$ is the **unique polynomial representation** for C over $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$

FGLM Algorithm Overview

- FGLM starts by taking the least monomial in our abstraction term ordering, A.
- Starting with $m = 0$, it computes $[A^m \bmod G_1]$
  - Stores the remainder, r
- Checks to see if the remainder is a combination of any previous remainders calculated thus far.
- If so, it adds this representation to G_2 and moves on to the next monomial, else it increments m.

How exactly FGLM computes if r is a combination of previous remainders, and how it finds this combination, is still being investigated

FGLM Algorithm Example

- $A^0 = 1$
- $A^1 = a_0 + a_1\alpha$
- $A^2 = a_0 + a_1 \cdot (\alpha + 1)$
- $A^3 = a_0 \cdot a_1 + a_0 + a_1$
- $A^4 = a_0 + a_1\alpha = A$

A^4 can be composed of A, so $A + A^4$ is added to Gröbner basis

- $B^1 = b_0 + (\alpha) \cdot b_1$
- $B^2 = b_0 + (\alpha + 1) \cdot b_1$
- $B^3 = b_0 \cdot b_1 + b_0 + b_1$
- $B^4 = b_0 + (\alpha) \cdot b_1 = B$

B^4 can be composed of B, so $B + B^4$ is added to Gröbner basis

- $Z^1 = a_0 \cdot b_0 + a_0 \cdot b_1 \cdot \alpha + a_1 \cdot b_0 \cdot \alpha + a_1 \cdot b_1 \cdot \alpha^2$
  $= (a_0 + a_1 \cdot \alpha) \cdot (b_0 + b_1 \cdot \alpha) = A \cdot B$

$Z + A \cdot B$ is added to Gröbner basis

FGLM-Related Research

- FGLM continues to convert every monomial to the new term order
- However, we only care about the word-level variables found in the $Y + \mathcal{F}(A)$ polynomial
- We can make the FGLM more efficient for our approach by restricting it to only compute the word-level variables
- Singular contains an FGLM implementation ('fglm' command)
  - Propose to modify Singular's implementation

Proposed Contributions

- Explore the implementation of Singular's FGLM algorithm.
- Develop an efficient CAD tool FGLM implementation which only performs ordering conversions on the required monomials.
- Research the complexity and feasibility of our given approach over large circuits.
- Apply the approach to elliptic curve cryptography circuits - particularly hierarchically designed multipliers and point addition circuits.

Proposed Timeline

- Spring 2013: Research FGLM algorithm in more detail. Study and analyze the source code of Singular's FGLM implementation.
- Early Summer 2013: Develop modified FGLM implementation. Run experiments on circuits of various sizes using proposed approach with the modified FGLM algorithm.
- Late Summer 2013: Evaluate data. Write Thesis.

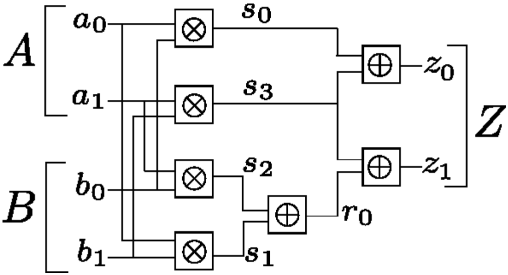
OLD STUFF

Using Our Topological Term Order:

- $F = \{f_1, \ldots, f_s\}$ is a Gröbner Basis of $J = \langle f_1, \ldots, f_s \rangle$
- $F_0 = \{x_1^q - x_1, \ldots, x_n^q - x_n\}$ is also a Gröbner basis of J_0
- But we have to compute a Gröbner Basis of $J + J_0 = \langle f_1, f_2 \ldots, f_s, \quad x_1^q - x_1, \ldots, x_n^q - x_n \rangle$
- We show that $\{f_1, f_2 \ldots, f_s, \quad x_1^q - x_1, \ldots, x_n^q - x_n\}$ is a Gröbner basis!!
- From our circuit: $f_i = x_i + P$
- Only pairs to consider: $S(f_i, \quad x_i^q - x_i)$ in Buchberger's Algorithm:

$$S(f_i, \quad x_i^q - x_i) \xrightarrow{J}_+ P^q - P \xrightarrow{J_0}_+ 0$$

Conclusion: Our term order makes $\{f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_n^q - x_n\}$ a Gröbner Basis



$$f_1 : s_0 + a_0 \cdot b_0,\ lm = s_0; \quad s_0^q - s_0$$

$$f_2 : s_2 + a_1 \cdot b_0,\ lm = s_2; \quad s_2^2 - s_2$$

- Every gate: $f_i : x_i + P \in J$
- Every vanishing polynomial: $x_i^q - x_i \in J_0$

$S(f_i, \quad x_i^q - x_i) \xrightarrow{J}_+ P^q - P \xrightarrow{J_0}_+ 0$
$\{f_1, \ldots, f_s, \quad x_1^q - x_1, \ldots, x_n^q - x_n\}$ is a Gröbner basis

Our Overall Approach

- Given the circuit, perform reverse topological traversal
- Derive the term order to represent the polynomials for every gate
- The set: $\{F, F_0\} = \{f_1, \ldots, f_s, \;\; x_1^q - x_1, \ldots, x_n^q - x_n\}$ is a Gröbner Basis
- Obtain: $f \xrightarrow{F,F_0}_+ r$
- If $r = 0$, the circuit is correct
- If $r \neq 0$, then r contains only the primary input variables
- Any SAT assignment to $r \neq 0$ generates a counter-example
- Counter-example found in no time as r is simplified by Gröbner basis reduction

Prior Work

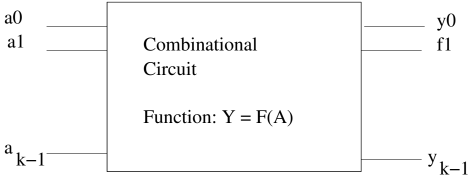
Wienand *et al* CAV'2008: Similar approach for verification of integer multipliers

- Works over rings $\mathbb{Z}_{2^k}$
- They derive the same term order: $f \xrightarrow{F}_{+} g$
- Then the circuit is correct if g is a *vanishing polynomial*; $g \in F_0$ over $\mathbb{Z}_{2^k}$
- But they do not investigate if F, F_0 is a Gröbner basis....

Mukopadhyaya, TCAD 2007 ($<$ 16-bit circuits), our own approach VLSI Design 2012, other theorem proving papers....

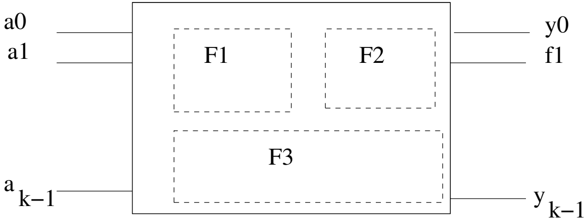
BLUVERI from IBM, *A. Lvov, et al.*, FMCAD 2012.

Polynomial Interpolation from Circuits



- Circuit: $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$
- $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$ or $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$
- Interpolate a polynomial from the circuit: $Y = F(A)$
- $A = a_0 + a_1\alpha + \ldots a_{k-1}\alpha^{k-1}, \quad Y = y_0 + y_1\alpha + \ldots y_{k-1}\alpha^{k-1}$
- Compute Gröbner basis of circuit polynomials with Elimination order: circuit-variables $> Y > A$
- Obtain $Y = F(A)$ as a unique, canonical, polynomial representation from the circuit

Polynomial Interpolation from Circuits



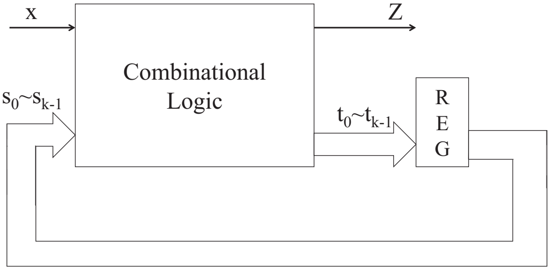
Hierarchical Interpolation

- Partition the circuit into sub-circuits
- Interpolate Polynomials $F_1, F_2, \ldots$ from Partitions
- Re-compute Gröbner basis of $\{F_1, F_2, \ldots\}$
- Eliminate internal variables to obtain $Y = F(A)$

Conclusions

- Formal Verification of large Galois Field circuits
- Computer algebra approach:
  - Nullstellensatz+Gröbner Bases methods
  - Engineering $\rightarrow$ a term order to obviate Gröbner basis computation
  - Can verify upto 163-bit circuits
  - NIST specified 163-bit field.... practical verification!
- Our approach relies only on polynomial division
- Complexity of polynomial division: Polynomial in the size of $f_1, \ldots, f_s$
- Almost the same time to catch bugs
- Conventional approaches fail miserably.....
- Future Work: Verify sequential GF-arithmetic Circuits
  - State-space traversal: Quantifier Elimination over Gröbner Basis

Typical Sequential Circuit



- Primary input(s): x, primary output(s): Z
- Pseudo inputs: $\{s_0, s_1, \ldots, s_{k-1}\}$
- Pseudo outputs: $\{t_0, t_1, \ldots, t_{k-1}\}$

A FSM Example

Figure : State Transition Graph

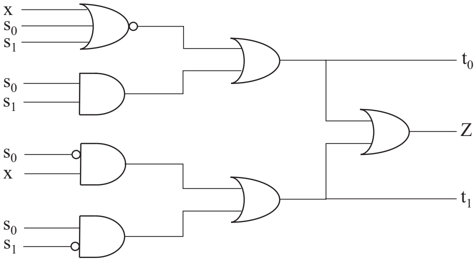


Figure : Corresponding Gate-level Circuit

ALGORITHM 1: Breadth-first Traversal Algorithm

Input: Transition functions Δ, initial state S^0

$from^0 = reached = S^0$;

repeat

  $i \leftarrow i + 1$;

  $to^i \leftarrow \mathrm{Img}(\Delta, from^{i-1})$;

  $new^i \leftarrow to^i \cap \overline{reached}$;

  $reached \leftarrow reached \cup new^i$;

  $from^i \leftarrow new^i$;

until $new^i == 0$;

return *reached*

Image function, states intersection, union and complement in this algorithm will be implemented in computer algebra and algebraic geometry.

Implement Image Function in Computer Algebra

- State variables (word-level) S, T and sets of states such as $from^i$, to^i can always be represented as varieties of ideals.
- Boolean operators can always be converted to operations in $\mathbb{F}_2$

Boolean operator	operation in $\mathbb{F}_2$
$\overline{a}$	$1 + a$
a *and* b	ab
a *or* b	$a + b + ab$
$a \oplus b$	$a + b$

Table : Some Boolean operators and corresponding operations in $\mathbb{F}_2$

- An *elimination ideal* can be built from circuit gates, pseudo input/output word definition and vanishing polynomials

Implement Image Function in Computer Algebra(2)

Elimination ideal to model Image function for example STG:

- Transition functions (bit-level):
$$\begin{aligned} f_1: &\quad t_0 - (\overline{x} \textit{ and } \overline{s_0} \textit{ and } \overline{s_1}) \textit{ or } (s_0 \textit{ and } s_1) \\ f_2: &\quad t_1 - (\overline{s_0} \textit{ and } x) \textit{ or } (s_0 \textit{ and } \overline{s_1}) \end{aligned}$$
- Word variable definitions:
$$\begin{aligned} f_3: &\quad S - s_0 - s_1\alpha \\ f_4: &\quad T - t_0 - t_1\alpha \end{aligned}$$
- Vanishing polynomials: $f_6 : x^2 - x; f_7 : t_0^2 - t_0; f_8 : t_1^2 - t_1; f_9 : S^4 - S; f_{10} : s_0^2 - s_0; f_{11} : s_1^2 - s_1; f_{12} : T^4 - T$

Add the current state (for example, add initial states in first iteration $f_5 : S$), compute Gröbner basis for ideal $J = \langle f_1, \dots, f_{12} \rangle$ under elimination term order

intermediate bit-level signals $>$ *bit-level primary inputs/outputs* $> S > T$

result will include a univariate polynomial about *next states* T.

Definition

(**Sum of Ideals**) If I and J are ideals in $k[x_1, \dots, x_n]$, then the **sum** of I and J, denoted by $I + J$, is the set

$$I + J = \{f + g \mid f \in I \text{ and } g \in J\}.$$

Furthermore, if $I = \langle f_1, \dots, f_r \rangle$ and $J = \langle g_1, \dots, g_s \rangle$, then $I + J = \langle f_1, \dots, f_r, g_1, \dots, g_s \rangle$.

Definition

(**Product of Ideals**) If I and J are ideals in $k[x_1, \dots, x_n]$, then the **product** of I and J, denoted by $I \cdot J$, is defined to be the ideal generated by all polynomials $f \cdot g$ where $f \in I$ and $g \in J$. Furthermore, let $I = \langle f_1, \dots, f_r \rangle$ and $J = \langle g_1, \dots, g_s \rangle$, then

$$I \cdot J = \langle f_i g_j \mid 1 \leq i \leq r, 1 \leq j \leq s \rangle.$$

Algebraic Geometry Concepts(2)

Definition

(**Quotient of Ideals**) If I and J are ideals in $k[x_1, \ldots, x_n]$, then $I : J$ is the set

$$\{f \in k[x_1, \ldots, x_n] \mid f \cdot g \in I, \forall g \in J\}$$

and is called the **ideal quotient** of I by J.

Concepts are adopted by following theorems:

Theorem

If I and J are ideals in $k[x_1, \ldots, x_n]$, then $\mathbf{V}(I + J) = \mathbf{V}(I) \bigcap \mathbf{V}(J)$ and $\mathbf{V}(I \cdot J) = \mathbf{V}(I) \bigcup \mathbf{V}(J)$.

Theorem

If I, J are ideals with only one generator, then $\mathbf{V}(I : J) = \mathbf{V}(I) - \mathbf{V}(J)$.

ALGORITHM 2: Algebraic Geometry based Traversal Algorithm

Input: Input-output circuit characteristic polynomial ideal I_{ckt}, initial state polynomial $\mathcal{F}(S)$

$from^0 = reached = \mathcal{F}(S)$;

repeat

$i \leftarrow i + 1$;

$to^i \leftarrow$GB w/ elimination term order$\langle I_{ckt}, from^{i-1}\rangle$;

$new^i \leftarrow$generator of $\langle to^i\rangle + (\langle T^4 - T\rangle : \langle reached\rangle)$;

$reached \leftarrow$generator of $\langle reached\rangle \cdot \langle new^i\rangle$;

$from^i \leftarrow new^i(S \setminus T)$;

until $new^i == 1$;

return *reached*

Example Executing New Traversal Algorithm

State encodings are mapped to varieties of ideals, e.g.:

$$\{00, 01\} \rightarrow \{0, 1\} = V_{\mathbb{F}_{2^2}}(\langle T^2 + T \rangle)$$

$$\{01, 10, 11\} \rightarrow \{1, \alpha, 1 + \alpha\} = V_{\mathbb{F}_{2^2}}(\langle T^3 + 1 \rangle)$$

- Iteration 0: Assume initial state is $\{00\} \rightarrow \{0\}$
- Iteration 1: $\mathit{reached} = \mathit{from}^0 = 0 = V_{\mathbb{F}_{2^2}}(\langle S \rangle), \mathit{to}^1 = \{1, \alpha\} = V_{\mathbb{F}_{2^2}}(\langle T^2 + (1 + \alpha)T + \alpha \rangle), \mathit{new}^1 = \mathit{to}^1, \mathit{reached} = \{0, 1, \alpha\} = V_{\mathbb{F}_{2^2}}(\langle T^3 + (1 + \alpha)T^2 + \alpha T \rangle)$
- Iteration 2: $\mathit{from}^1 = \mathit{new}^1(S \setminus T) = \{1, \alpha\} = V_{\mathbb{F}_{2^2}}(\langle S^2 + (1 + \alpha)S + \alpha \rangle), \mathit{to}^2 = \{0, \alpha\} = V_{\mathbb{F}_{2^2}}(\langle T^2 + \alpha T \rangle), \mathit{new}^2 = 1,$ **Terminate**
- Return $\mathit{reached} = \{0, 1, \alpha\} = V_{\mathbb{F}_{2^2}}(\langle T^3 + (1 + \alpha)T^2 + \alpha T \rangle)$

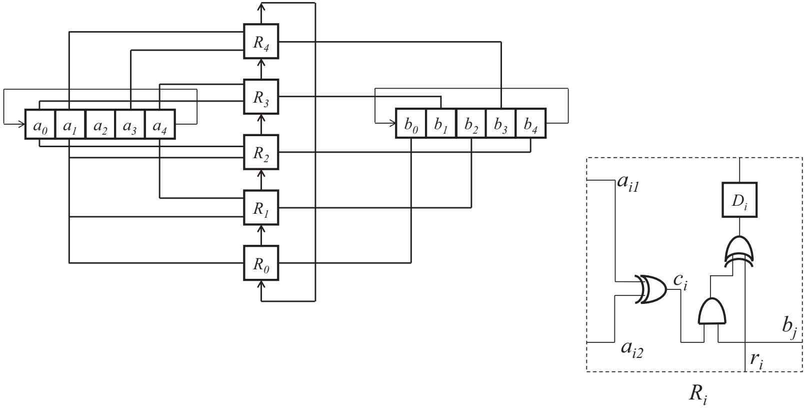


Figure : 5-bit Normal Basis Angew's Sequential Multiplier with Parallel Output (SMPO)

SMPO Protocol

- **Initial** $R_0 = R_1 = R_2 = R_3 = R_4 = 0$
- **Clock 1** $R_0 = a_1 b_0, R_1 = b_2(a_1 + a_4), R_2 = b_4(a_0 + a_1), R_3 = b_1(a_4 + a_0), R_4 = b_3(a_1 + a_3)$
- **Clock 2**
  $R_0 = b_3(a_1 + a_3) + a_0 b_4, R_1 = a_1 b_0 + b_1(a_0 + a_3), R_2 = b_2(a_1 + a_4) + b_3(a_4 + a_0), R_3 = b_4(a_0 + a_1) + b_0(a_3 + a_4), R_4 = b_1(a_4 + a_0) + b_2(a_0 + a_2)$
- ...
- **Clock 5** $R_0 = c_0, R_1 = c_1, R_2 = c_2, R_3 = c_3, R_4 = c_4$, i.e. $R = A \cdot B$.

Compose Elimination Ideal for 5-bit SMPO

An elimination ideal for the first clock cycle:

- **Gate descriptions:** $a_1 + a_4 + c_1, a_1 + a_0 + c_2, a_0 + a_4 + c_3, a_1 + a_3 + c_4, a_1 b_0 + r_4 + R_0, c_1 b_2 + r_0 + R_1, c_2 b_4 + r_1 + R_2, c_3 b_1 + r_2 + R_3, c_4 b_3 + r_3 + R_4;$
- **Word-level variables:** $A + a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18}, B + b_0\alpha^5 + b_1\alpha^{10} + b_2\alpha^{20} + b_3\alpha^9 + b_4\alpha^{18}, r + r_0\alpha^5 + r_1\alpha^{10} + r_2\alpha^{20} + r_3\alpha^9 + r_4\alpha^{18}, R + R_0\alpha^5 + R_1\alpha^{10} + R_2\alpha^{20} + R_3\alpha^9 + R_4\alpha^{18};$
- **Vanishing polynomials:** $a_0^2 + a_0, a_1^2 + a_1, a_2^2 + a_2, a_3^2 + a_3, a_4^2 + a_4, b_0^2 + b_0, b_1^2 + b_1, b_2^2 + b_2, b_3^2 + b_3, b_4^2 + b_4, r_0^2 + r_0, r_1^2 + r_1, r_2^2 + r_2, r_3^2 + r_3, r_4^2 + r_4, R_0^2 + R_0, R_1^2 + R_1, R_2^2 + R_2, R_3^2 + R_3, R_4^2 + R_4, c_1^2 + c_1, c_2^2 + c_2, c_3^2 + c_3, c_4^2 + c_4, A^{32} + A, B^{32} + B, r^{32} + r, R^{32} + R;$
- **Feedback input:** r_{in}.

Fast Abstraction without GB computation

Definition

A lexicographic order constrained by following relation $>_r$: "circuit variables ordered reverse topologically" $>$ "designated word-level output" $>$ "word-level inputs" is called the *Refined Abstraction Term Order (RATO)*.

Example

The elimination ideal for 5-bit SMPO could be rewritten under RATO:

$$
\begin{aligned}
&(R_0, R_1, R_2, R_3, R_4) > (r_0, r_1, r_2, r_3, r_4) \\
&> (c_1, c_2, c_3, c_4, b_0, b_1, b_2, b_3, b_4) \\
&> (a_0, a_1, a_2, a_3, a_4) > R > r > (A, B)
\end{aligned}
$$

Under RATO, most polynomials have relatively prime leading terms/monomials (which means $Spoly \xrightarrow{J+J_0}_{+} 0$) except one pair: word-level polynomial corresponding to outputs and its leading bit-level variable's gate description polynomial.

Example

Candidate pair for 5-bit SMPO is $(f_w, f_g), f_w = R_0 + r_4 + b_0 \cdot a_1, f_g = R_0\alpha^5 + R_1\alpha^{10} + R_2\alpha^{20} + R_3\alpha^9 + R_4\alpha^{18} + R$.
Result after reduction is an abstraction:

$$
\begin{aligned}
& Spoly(f_w, f_g) \xrightarrow{J+J_0}_{+} \\
& r_1 + (\alpha)r_2 + (\alpha^4 + \alpha^2)r_3 + (\alpha^3 + \alpha^2)r_4 + (\alpha^3)b_1a_1 \\
+ & (\alpha^4 + \alpha^2)b_1a_2 + (\alpha^3 + \alpha + 1)b_1a_3 + (\alpha^3 + \alpha)b_1a_4 + (\alpha + 1)b_1A \\
+ & (\alpha^4 + \alpha^2 + \alpha)b_2a_1 + (\alpha^4 + \alpha^3 + \alpha^2 + \alpha)b_2a_4 + (\alpha^3 + \alpha^2 + 1)b_3a_1 \\
+ & (\alpha)b_3a_3 + (\alpha^2 + \alpha + 1)b_4a_1 + (\alpha + 1)b_4a_2 + (\alpha^4 + \alpha^2)b_4a_3 \\
+ & (\alpha^4 + \alpha^3 + \alpha + 1)b_4a_4 + (\alpha^3 + 1)b_4A + (\alpha^4 + \alpha^3 + \alpha^2 + 1)a_1B \\
+ & (\alpha^4 + \alpha^3 + \alpha^2 + 1)R
\end{aligned}
$$

Use Gaussian elimination style approach, eliminate other bit-level variables except for one.

Example

Objective: Abstract polynomial $a_i + \mathcal{G}_i(A)$ from $f_0 : a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18} + A$. Eliminate variable a_0 by operation

$$\begin{aligned} f_1 = & f_0 \times \alpha^5 + f_0^2 : \\ & a_1 + (\alpha)a_2 + (\alpha^4 + \alpha^2)a_3 + (\alpha^3 + \alpha^2)a_4 \\ & + (\alpha^4 + \alpha^3 + \alpha^2 + 1)A^2 + (\alpha^2 + \alpha)A \end{aligned}$$

Recursively eliminate a_1 from f_1, a_2 from f_2, etc.

Example

For 5-bit SMPO example, the result is

$$\begin{cases} a_0 &= (\alpha+1)A^{16} + (\alpha^4+\alpha^3+\alpha)A^8 + (\alpha^3+\alpha^2)A^4 \\ & \quad +(\alpha^4+1)A^2 + (\alpha^2+1)A \\ a_1 &= (\alpha^2+1)A^{16} + (\alpha+1)A^8 + (\alpha^4+\alpha^3+\alpha)A^4 \\ & \quad +(\alpha^3+\alpha^2)A^2 + (\alpha^4+1)A \\ a_2 &= (\alpha^4+1)A^{16} + (\alpha^2+1)A^8 + (\alpha+1)A^4 \\ & \quad +(\alpha^4+\alpha^3+\alpha)A^2 + (\alpha^3+\alpha^2)A \\ a_3 &= (\alpha^3+\alpha^2)A^{16} + (\alpha^4+1)A^8 + (\alpha^2+1)A^4 \\ & \quad +(\alpha+1)A^2 + (\alpha^4+\alpha^3+\alpha)A \\ a_4 &= (\alpha^4+\alpha^3+\alpha)A^{16} + (\alpha^3+\alpha^2)A^8 + (\alpha^4+1)A^4 \\ & \quad +(\alpha^2+1)A^2 + (\alpha+1)A \end{cases}$$

By substitution of bit-level variables in remainder from RATO, get next state abstraction $R + \mathcal{F}(A, B)$

Results Comparing to SAT/ABC/BDD

	Word size of the operands k-bits			
Solver	11	18	23	33
Lingeling	593	*TO*	*TO*	*TO*
ABC	6.24	*TO*	*TO*	*TO*
BDD	0.1	11.7	1002.4	*TO*

Table : Runtime for verification of bug-free SMPO circuits over $\mathbb{F}_{2^k}$ for SAT, ABC and BDD based methods. TO = timeout of 14 hrs

Operand size k	36	66	82	89	100
#variables	183	333	413	448	503
#polynomials	2700	8910	13694	16109	20300
#terms	12996	43626	67322	79299	100100
Runtime(bug-free)	113	3673	15117	28986	50692
Runtime(buggy)	118	4320	15226	31571	58861

Table : Runtime (given in seconds) for verification of bug-free and buggy Angew's SMPO circuits over $\mathbb{F}_{2^k}$ using our approach