WORD-LEVEL FINITE STATE MACHINE TRAVERSAL USING GRÖBNER BASIS

Xiaojun Sun*, Priyank Kalla*, Florian Enescu†
*Electrical & Computer Engineering, University of Utah
†Mathematics & Statistics, Georgia State University {xiaojuns, kalla}@ece.utah.edu, fenescu@gsu.edu

I. PRELIMINARIES

A. FSM model for sequential circuits

A finite state machine (FSM) is a mathematical model of computation for designing and analyzing sequential logic circuits. If a FSM's primary outputs depend on primary inputs and present state inputs, it is named as a *Mealy machine*; the formal definition is as follows:

Definition I.1. *A Mealy machine is an n-tuple $\mathcal{M} = (\Sigma, O, S, S^0, \Delta, \Lambda)$ where*

- *Σ is the input label, O is the output label;*
- *S is the set of states, $S^0 \subseteq S$ is the set of initial states;*
- *$\Delta: \ S \times \Sigma \to S$ is the next state transition function;*
- *$\Lambda: \ S \times \Sigma \to O$ is the output function.*

The other kind of FSM is *Moore machine*, its difference from Mealy machine is that its primary outputs only depend on the present states, i.e. the output function is defined as

$$\Lambda: \ S \to O$$

Typical sequential circuits can be depicted as Fig.1(a). Primary inputs $x_1, \dots, x_m \in \Sigma$, and primary outputs $z_1, \dots, z_n \in O$. Signals $s_1, \dots, s_k$ are present state (PS) variables, $t_1, \dots, t_k$ are next state (NS) variables. We can define 2 k-bit words denoting the PS/NS variables as there are k flip-flops in the datapath: $S = (s_1, \dots, s_k)$, $T = (t_1, \dots, t_k)$. Transition function at bit level are defined as $\Delta_i : t_i = \Delta_i(s_1, \dots, s_k, x_1, \dots, x_m)$. In some cases, arithmetic computations are implemented as Moore machines where input operands are loaded into register files R and the FSM is executed for k clock cycles. We can simplify them to the model in Fig.1(b).

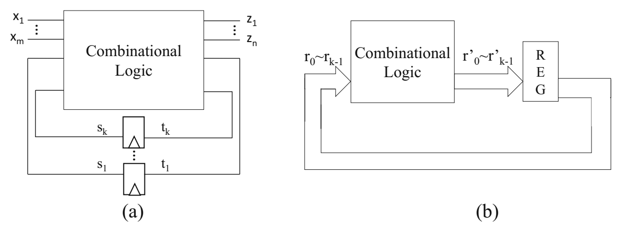


Fig. 1: FSM models of sequential circuits

B. Commutative algebra and algebraic geometry preliminaries

A **field** $\mathbb{F}$ is a set of elements, including 0 and 1 (unity), allowing for associative and commutative addition and multiplication; and every non-zero element has a multiplicative inverse. A **finite field** or **Galois filed** is a field with a finite number (q) of elements, and is denoted by $\mathbb{F}_q$, where $q = p^k$ is a power of a prime integer p. In our work, $q = 2^k$ for a given k, where k represents the datapath (bit-vector) word-lengths, or the number of memory elements (state registers) in finite state machines.

Let the field $\mathbb{F}_2 = \{0, 1\}$ ($\equiv \mathbb{B}$), and let $\mathbb{F}_2[X]$ denote the set (ring) of all univariate polynomials in variable X with coefficients from $\mathbb{F}_2$. Then, the Galois field $\mathbb{F}_{2^k}$ is constructed as $\mathbb{F}_{2^k} = \mathbb{F}_2[X] \pmod{P(X)}$, where $P(X)$ is an irreducible polynomial over $\mathbb{F}_2$. Let α be a root of the irreducible polynomial $P(X)$, i.e. $P(\alpha) = 0$. Any element $A \in \mathbb{F}_{2^k}$ can be represented as $A = \sum_{i=0}^{k-1} a_i \alpha^i$, where $a_i \in \mathbb{F}_2$. The field $\mathbb{F}_{2^k}$ is therefore, a k-dimensional *extension* of the base field $\mathbb{F}_2$: so, $\mathbb{F}_{\mathbf{2^k}} \supset \mathbb{F}_{\mathbf{2}}$. Consequently, all operations of addition and multiplication in $\mathbb{F}_{2^k}$ are performed modulo the irreducible polynomial $P(\alpha)$ and coefficients are reduced modulo 2.

Boolean variables in field $\mathbb{B}$ can be easily mapped to elements in $\mathbb{F}_2$. Since $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$, these Boolean operators are interpreted as functions over $\mathbb{F}_{2^k}$ (where $+$ and $\cdot$ are addition and multiplication performed modulo 2):

$$\begin{aligned}
&a \wedge b \to a \cdot b \\
&a \oplus b \to a + b \\
&\neg a \to 1 + a \\
&a \bar{\oplus} b \to 1 + a + b \\
&a \vee b \to a + b + a \cdot b
\end{aligned}$$

Using these mappings we can write Boolean functions in form of polynomials over $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$. These concepts provide a mechanism to represent and manipulate both bit-level ($\mathbb{F}_2$) and k-bit word-level constraints **in one unified mathematical domain $\mathbb{F}_{2^k}$ — a concept we exploit for abstraction.** Consider Ex.**??**, polynomials for transition functions (bit-level outputs) f_1, f_2 are over $\mathbb{F}_2$, i.e. $f_1, f_2 \subseteq \mathbb{F}_2 \subset \mathbb{F}_{2^k}$, and polynomials containing word-level variables $f_3, f_4, f_5 \subseteq \mathbb{F}_{2^k}$. All polynomials belong to unified domain $f_1, f_2, \dots, f_5 \subseteq \mathbb{F}_{2^k}$.

It is well-known that every Boolean mapping between k dimensional Boolean spaces $f : \mathbb{B}^k \to \mathbb{B}^k$ can be construed as a function over Galois fields $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. Moreover, every function $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ is a polynomial function: i.e. f can be represented by way of a unique, minimal, canonical polynomial $\mathcal{F}(X)$, and the work of [1] shows how to efficiently derive such polynomial representations from circuits — another concept that makes our approach feasible.

We represent Boolean circuits by way of polynomials over $\mathbb{F}_{2^k}$. If we take indeterminates $x_1, x_2, \dots, x_n$, an arbitrary combination of their finite product $x_1^{d_1} \cdot x_2^{d_2} \cdots x_n^{d_n}, d_i \geq 0$ is a **monomial**. A **polynomial** $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$ is a finite sum of terms, where $c_1, \dots, c_t$ are coefficients and $X_1, \dots, X_t$ are monomials. The set of *all* such polynomials with coefficients from $\mathbb{F}_{2^k}$ forms a **multivariate polynomial ring** denoted $\mathbb{F}_{2^k}[x_1,\dots,x_n]$. A monomial ordering $X_1 > X_2 \cdots > X_t$ is imposed on the polynomials to process them systematically. Then, $LT(f) = c_1X_1, LM(f) = X_1$ denote the leading term and the leading monomial of f, respectively.

Multivariate polynomial division will play a key role in our algorithmic techniques. Division is implemented as *cancellation of terms.* Given polynomials f,g, if cX is a term in f that is divisible by $LT(g)$, then $f \xrightarrow{g} r$ denotes a one-step reduction (division) of f by g, resulting in remainder $r = f - \frac{cX}{LT(g)} \cdot g$. This has the effect of cancelling the term cX from f.

Example I.1. *If $f = e + cd$ and $g = c + ab$, then the term cd in f can be canceled by $LT(g) = c$: $r = f - \frac{cd}{c} g = e + abd$.*

Similarly, f reduces to r modulo the set of polynomials $F = \{f_1,\dots,f_s\}$, denoted $f \xrightarrow{F}_+ r$, such that no term in r is divisible (cancellable) by the $LT(f_i)$ of any polynomial in $f_i \in F$.

In verification, we have to analyze the *solutions to a set of polynomials.* The set of all solutions to a system of polynomial equations $f_1 = \cdots = f_s = 0$ is defined as the affine variety:

Definition I.2. *Given a set of polynomials $f_1,\dots,f_s$ over ring $\mathbb{F}_q[x_1,\dots,x_n]$, their* **affine variety**

$$V(f_1,\dots,f_s) = \{(a_1,\dots,a_n) \in (\mathbb{F}_q)^n | f_1(a_1,\dots,a_n) = \cdots = f_s(a_1,\dots,a_n) = 0\}$$

Generally we can find many sets of polynomials with the same variety, which are linear combinations of given set of polynomials. This set is defined as follows:

Definition I.3. Ideal of Polynomials: *Let $f_1,f_2,\dots,f_s \in \mathbb{F}[x_1,\dots,x_n]$. Define an ideal*

$$J = \langle f_1,f_2,\dots,f_s\rangle = \{f_1 \cdot h_1 + f_2 \cdot h_2 + \cdots + f_s \cdot h_s : h_1,\dots,h_s \in \mathbb{F}[x_1,\dots,x_n]\}$$

We call $J = \langle f_1,f_2,\dots,f_s\rangle$ an ideal generated by $f_1,\dots,f_s$ and these polynomials the **generators** *of ideal J.*

A practical problem is: given an ideal $J = \langle f_1,f_2,\dots,f_s\rangle$ and a polynomial f, we need to check if the variety of J can make the evaluation of f equal to 0, i.e. f vanishes on $V(J)$. This problem is usually described as ideal membership checking problem.

Definition I.4. Ideal membership: *Let $f_1,f_2,\dots,f_s \in \mathbb{F}[x_1,\dots,x_n]$, and $J = \langle f_1,f_2,\dots,f_s\rangle$ be an ideal over ring $\mathbb{F}[x_1,\dots,x_n]$. If*

$$f = f_1h_1 + f_2h_2 + \cdots + f_sh_s$$

then $f \in J$.

An ideal may have many generating sets. For example, we may have different set of polynomial generators denoting the same ideal, where they have the same variety: $\langle f_1,\dots,f_s\rangle = \langle h_1,\dots,h_r\rangle = \langle g_1,\dots,g_t\rangle$ such that $V(f_1,\dots,f_s) = V(h_1,\dots,h_r) = V(g_1,\dots,g_t)$. Therefore a canonical representation of an ideal is needed, which leads to the concept of Gröbner bases.

Definition I.5. *The set $G = \{g_1,\dots,g_t\}$ is called a* ***Gröbner basis*** *of J if and only if the leading term of all polynomials in J is divisible be the leading term of some polynomial g_i in G: i.e. $\forall f \in J, \exists g_i \in G$ s.t. $LT(g_i) \mid LT(f)$.*

An advantage of representing an ideal with GB is that it can serve as a decision procedure for ideal membership test when dividing a polynomial f by a GB, i.e.

$$G = GB(J) \Longleftrightarrow \forall f \in J, f \xrightarrow{g_1,g_2,\dots,g_t}_+ 0$$

Gröbner basis can be reduced by eliminating redundant elements. **A reduced GB is a canonical representation of the ideal under a given monomial ordering**. Given an ideal $J = \langle f_1,\dots,f_s\rangle$, $G = \{g_1,\dots,g_t\}$ is the GB of J, it can be computed by Buchberger's algorithm (refer to textbook [2]).

Another advantage of using GB representation is that GB computation can work as a *quantification procedure*. In the following part we will introduce the concept of *vanishing polynomials*, *elimination ideal*, etc. as the bases of this theory.

Fermat's little theorem over $\mathbb{F}_q$: For any $\alpha \in \mathbb{F}_q, \alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ vanishes $(= 0)$ over $\mathbb{F}_q$, and is called a vanishing polynomial. We denote by $J_0 = \langle x_1^q - x_1,\dots,x_d^q - x_d\rangle$ the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1,\dots,x_d]$. When $q = 2^k, x^q - x = x^q + x$ as $-1 = +1$ over $\mathbb{F}_{2^k}$.

Gröbner bases can be used to *eliminate* (i.e. quantify) variables from an ideal. Given ideal $J = \langle f_1,\dots,f_s\rangle \subset \mathbb{F}_q[x_1,\dots,x_d]$, the l^{th} elimination ideal J_l is the ideal of $\mathbb{F}_q[x_{l+1},\dots,x_d]$ defined by $J_l = J \cap \mathbb{F}_q[x_{l+1},\dots,x_d]$. Variable elimination can be achieved by computing a Gröbner basis of J w.r.t. elimination orders:

Theorem I.1. *(Elimination theorem[3]) Let $J \subset \mathbb{F}_{2^k}[x_1,\dots,x_d]$ be an ideal and let G be a Gröbner basis of J with respect to a lexicographic (LEX) ordering where $x_1 > x_2 > \cdots > x_d$. Then for every $0 \le l \le d$, the set $G_l = G \cap \mathbb{F}_{2^k}[x_{l+1},\dots,x_d]$ is a Gröbner basis of the l-th elimination ideal J_l.*

We describe an application of elimination ideals using following example borrowed from [3]:

Example I.2. *Consider polynomials $f_1 : x^2 - y - z - 1$; $f_2 : x - y^2 - z - 1$; $f_3 : x - y - z^2 - 1$ and ideal $J = \langle f_1,f_2,f_3\rangle \subset \mathbb{C}[x,y,z]$. Gröbner basis $G = GB(J)$ w.r.t. LEX term order equals to $g_1 : x - y - z^2 - 1$; $g_2 : y^2 - y - z^2 - z$; $g_3 : 2yz^2 - z^4 - z^2$; $g_4 : z^6 - 4z^4 - 4z^3 - z^2$. From observation, we find that the polynomial g_4 only contains variable z (x,y eliminated), and polynomials g_2, g_3, g_4 only contain variables y,z (x eliminated). According to theorem I.1, $G_1 = G \cap \mathbb{C}[y,z] = \{g_2,g_3,g_4\}$ is the Gröbner basis of the 1^{st} elimination ideal of J and $G_2 = G \cap \mathbb{C}[z] = \{g_4\}$ is the 2^{nd} elimination ideal of J, respectively.*

C. Application of elimination theorem on circuit verification

Assume that we are given a circuit (combinational component) with input $A = (a_0,\dots,a_{k-1})$ and output $R = (r_0,\dots,r_{k-1})$ (both can be represented by word level variables in $\mathbb{F}_{2^k}$). We can describe this circuit with an elimination ideal $J + J_0$, where J is the ideal generated by the polynomials corresponding to circuit gates and J_0 is the ideal of vanishing polynomials. The authors of [1] showed that for any combinational logic block, a canonical word-level polynomial representation can be derived through Gröbner bases computed with elimination orders:

Lemma I.1. *(From [1]) Given a combinational circuit C with k-bit input $A = (a_0, \dots, a_{k-1})$ and k-bit output $R = (r_0, \dots, r_{k-1})$. Denote by $x_1, \dots, x_d$ all the bit-level variables of C. Let $J = \langle f_1, \dots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \dots, x_d, R, A]$ denote all the polynomials corresponding to the logic gates of the circuit. Let $J_0 = \langle x_1^2 - x_1, \dots, x_d^2 - x_d, R^q - R, A^q - A \rangle$ be the vanishing ideal, so that $J + J_0 = \langle f_1, \dots, f_s, \;\; x_1^2 - x_1, \dots, x_d^2 - x_d, R^q - R, A^q - A \rangle$. Compute Gröbner basis $G = GB(J + J_0)$ w.r.t. lex term order with $x_1 > x_2 > \cdots > x_d > R > A$. Then $G_d = G \cap \mathbb{F}_{2^k}[R, A]$ eliminates the internal variables $x_1, \dots, x_d$ of the circuit. G_d also contains the word-level polynomial $R = \mathcal{F}(A)$ which canonically represents the function of the circuit with only word level variables R and A.*

This lemma shows an application of GB computations over an elimination ideal. Since it abstracts the function of a combinational circuit, we call the term order $primary\ inputs\ and\ intermediate\ variables > word\ level\ output > word\ level\ inputs$ as *abstraction term order* (ATO). If we further eliminate word-level input, the result will be a polynomial containing only the word-level output variable. In a sequential circuit such as Ex.**??**, the output of combinational logic serves as the next state variable. Polynomial g_T in the example is the desired projection; i.e. using GB computation on elimination ideal and eliminating to NS variables provides us the canonical representation of reachable states in next time frame.

II. Reachability Analysis of Finite State Machines (FSMs) using Algebraic Geometry

In our approach we use symbolic state reachability with algebraic geometry concepts. It is an abstraction based on word operand definition of datapaths in circuits, and it can be applied to arbitrary FSMs by bundling a set of bit-level variables together as one or several word-level variables. The abstraction polynomial, encoding the reachable state space of the FSM, is obtained through computing a GB of an elimination ideal using elimination term order based on theorem I.1.

The motivation behind our approach can be described as follows:

- Any finite set of points is the variety (solutions) of a polynomial ideal. Since the state-space of a FSM is finite, it can be construed as the variety of an ideal corresponding to the function (circuit implementation, gates) of the sequential circuit.
- Using canonical representations of polynomial ideals (Gröbner bases), the transition functions and the sets of states can be used for FSM traversal and property checking.
- In our work, Gröbner bases are considered in lieu of BDDs or SAT-solvers. Furthermore, instead of working in the Boolean domain $\mathbb{B}$, we model the circuit and the properties over the Galois field $\mathbb{F}_{2^k}$. This allows to perform reachability analysis at the level of k-bit words, as $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$.

Reachability analysis is a useful tool when checking sequential equivalence as well as other invariants. Conceptually, the state-space of a FSM is traversed in a breadth-first manner, as shown in Algorithm 1. In general, the algorithm operates on the FSM $\mathcal{M} = (\sum, O, S, S^0, \Delta, \Lambda)$ underlying a sequential circuit. In such cases, the transition function Δ and the initial states are represented and manipulated using Boolean representations such as BDDs or SAT solvers. The variables $from, reached, to, new$ represent characteristic functions of sets of states. Starting from the initial state $from^i = S^0$, the algorithm computes the states reachable in 1-step from $from^i$ in each iteration. In line 4 of algorithm 1, the **image computation** is used to compute the 1-step next reachable states. In [4], the author implicitly represents the states and transition relations by Boolean functions and Boolean vectors, respectively.

Let us describe the application of the algorithm on the FSM circuit of Fig.**??**. *We will first describe its operation at the Boolean level, and then describe how this algorithm can be implemented using algebraic geometry at word level.*

In Line 1 of the BFS algorithm, assume that the initial state is S_3 in Fig.**??**(b), which is encoded as $S_3 = \{11\}$. Using Boolean variables s_0, s_1 for the present states, $from^0 = s_0 \cdot s_1$ is represented as a Boolean formula.

The *transition function* Δ is given by Boolean equations of the flip-flops of the circuit: $t_i = \Delta_i(s, x)$, where t_i is the next state variable, s represents the present state variables and x represents the input variables. The *Transition relation of the FSM* is then represented as:

$$T(s, x, t) = \prod_{i=1}^{n} (t_i \overline{\oplus} \Delta_i) \tag{1}$$

where n is the number of flip flops, and $\overline{\oplus}$ is XNOR operation. Let $from$ denote the set of initial states, then the image of the initial states, under the transition function Δ is finally computed as:

$$to = \text{Img}(\Delta, from) = \exists_s \ \exists_x \ [T(s, x, t) \cdot from] = \exists_s \ \exists_x \ \prod_{i=1}^{n} (t_i \overline{\oplus} \Delta_i) \cdot from \tag{2}$$

Here, the notation $\exists x(f)$ represents the *existential quantification of f w.r.t. variable x.*

ALGORITHM 1: Breadth-first Traversal Algorithm for Reachability Analysis of FSMs

```
Input: Transition functions $\Delta$, initial state $S^0$
1 $from^0 = reached = S^0$;
2 repeat
3     $i \leftarrow i + 1$;
4     $to^i \leftarrow \text{Img}(\Delta, from^{i-1})$;
5     $new^i \leftarrow to^i \cap \overline{reached}$;
6     $reached \leftarrow reached \cup new^i$;
7     $from^i \leftarrow new^i$;
8 until $new^i == 0$;
9 return $reached$
```

Example II.1. *For the circuit in Fig.* **??** *(b), we have the transition functions of the machine as:*

$$\Delta_1 : t_0 \overline{\oplus}((\overline{x+s_0+s_1})+s_0 s_1)$$
$$\Delta_2 : t_0 \overline{\oplus}(\overline{s_0}x+\overline{s_1}s_0)$$
$$from : from^0 = s_0 \cdot s_1$$

When the formula of Eqn. (2) is applied to compute 1-step reachability, $to = \exists_{s_0,s_1,x}(\Delta_1 \cdot \Delta_2 \cdot from^0)$, we obtain $to = \overline{t_0} \cdot t_1$, which denotes the state $S_1 = \{01\}$ reached in 1-step from S_3.

In the next iteration, the algorithm uses state $S_1 = \{01\}$ as the current (initial) state, and computes $S_2 = \{10\} = t_0 \cdot \overline{t_1}$ as the next reachable state, and so on.

Our objective is to model the transition functions Δ as a polynomial ideal J, and to perform the image computations (Algorithm 1, line 4) using Gröbner bases over Galois fields. **This requires to perform quantifier elimination; which can be accomplished using the Gröbner basis computation over Galois fields by using elimination ideals** [5]. Finally, the set union, intersection and complement operations are also to be implemented in algebraic geometry.

FSM Traversal at word-level over $\mathbb{F}_{2^k}$: The state transition graph (STG) shown in Fig.**??**(a) uses a 2-bit Boolean vector to represent 4 states $\{S_0, S_1, S_2, S_3\}$. We map these states to elements in $\mathbb{F}_{2^2}$, where $S_0 = 0, S_1 = 1, S_2 = \alpha, S_3 = \alpha+1$ where $\alpha^2+\alpha+1 = 0$.

Initial state: In Line 1 of the algorithm, the initial state is specified by means of a corresponding polynomial $f = \mathcal{F}(S) = S-1-\alpha$. Notice that if we consider the ideal generated by the initial state polynomial, $I = \langle f \rangle$, then its variety $V(I) = 1+\alpha$, corresponds to the state encoding $S_3 = \{11\} = 1+\alpha$, where a polynomial in word-level variable S encodes the initial state.

Set operations: When executing Line 5 and Line 6, we use **union**, **intersection** and **complement** of varieties over $\mathbb{F}_{2^k}$:

Definition II.1. *(***Sum/Product of Ideals***) If $I = \langle f_1, \dots, f_r \rangle$ and $J = \langle g_1, \dots, g_s \rangle$ are ideals in $\mathbb{F}[x_1, \dots, x_n]$, then the* **sum** *of I and J is defined as*

$$I+J = \langle f_1, \dots, f_r, g_1, \dots, g_s \rangle$$

And the **product** *of I and J is defined as*

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

With concepts of ideal sums and products, we can obtain the intersection and union of affine varieties as:

Theorem II.1. *If I and J are ideals in $\mathbb{F}[x_1, \dots, x_n]$, then $\mathbf{V}(I+J) = \mathbf{V}(I) \bigcap \mathbf{V}(J)$ and $\mathbf{V}(I \cdot J) = \mathbf{V}(I) \bigcup \mathbf{V}(J)$.*

In Line 5 of Alg.1, we need to compute the complement of a set of states. Assume that J denotes a polynomial ideal whose variety $V(J)$ denotes a set of states. We require the computation of another polynomial ideal J', such that $V(J') = \overline{V(J)}$. We have recently proved (the proofs are omitted) that this computation can be performed using the concept of **ideal quotient:**

Definition II.2. *(***Quotient of Ideals***) If I and J are ideals in $\mathbb{F}[x_1, \dots, x_n]$, then $I : J$ is the set*

$$\{f \in \mathbb{F}[x_1, \dots, x_n] \mid f \cdot g \in I, \forall g \in J\}$$

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

For elimination ideals in circuit verification problems, we can obtain the complement of a variety through the following result:

Theorem II.2. *Let J be ideals with single univariate polynomial about variable x over $\mathbb{F}_{2^k}[x]$, vanishing ideal $J_0 = \langle x^{2^k} - x \rangle$. then*

$$\mathbf{V}(J_0 : J) = \mathbf{V}(J_0) - \mathbf{V}(J)$$

To prove this theorem, we need following lemma:

Lemma II.1. *Let $f, g \in \mathbb{F}_{2^k}[x]$, then $(f : g) = \left(\frac{f}{gcd(f,g)}\right)$.*

Proof. Let $d = gcd(f,g)$. So, $f = df_1, g = dg_1$ with $gcd(f_1, g_1) = 1$.

Note that $f_1 = \frac{f}{gcd(f,g)}$.

Take $h \in (f : g)$. So, $hg \in (f)$, that is $hg = f \cdot r$ with $r \in \mathbb{F}_{2^k}[x]$.

Therefore, $hdg_1 = df_1 r$, and so $hg_1 = f_1 r$. But since $gcd(g_1, f_1) = 1$ we have that f_1 divides h. Hence $h \in (f_1)$. Conversely, let $h \in (f_1)$. Then $h = s \cdot f_1$, with $s \in \mathbb{F}_{2^k}[x]$. So, $hg = hdg_1 = sf_1 dg_1 = sg_1 f \in (f)$. Therefore, $h \in (f : g)$. □

Using this lemma we can easily prove the proposition:

Proof. Since $\mathbb{F}_{2^k}[x]$ is a principal ideal domain, then$J = (g)$ for some polynomial$g \in \mathbb{F}_{2^k}[x]$.

Let $d = gcd(g, x^{2^k} - x)$. So, $g = dg_1, x^{2^k} - x = df_1$, with $gcd(f_1, g_1) = 1$.

Then $J_0 : J = (f_1)$ by Lemma II.1.

Let $x \in \mathbf{V}_{\mathbb{F}_{2^k}}(J_0) - \mathbf{V}_{\mathbb{F}_{2^k}}(J)$. Then $x \in \mathbb{F}_{2^k}$, but $g(x) \neq 0$.

Since $x^{2^k} = x$, we see that either $d(x) = 0 or f_1(x) = 0$. Note that since $g(x) \neq 0$, we can conclude that $d(x) = 0$. In conclusion, $f_1(x) = 0$ and $x \in \mathbf{V}_{\mathbb{F}_{2^k}}(f_1)$.

Now let $x \in V\mathbf{V}_{\mathbb{F}_{2^k}}(f_1)$. Therefore $f_1(x) = 0$. So, $x^{2^k} - x = 0$ which gives $x \in \mathbf{V}_{\mathbb{F}_{2^k}}(J_0) = \mathbb{F}_{2^k}$.

Assume now that $x\mathbf{V}_{\mathbb{F}_{2^k}}(g)$. Then $0 = g(x) = d(x)g_1(x)$.

So, $d(x) = 0$ or $g_1(x) = 0$.

If $g_1(x) = 0$, since $f_1(x) = 0$ we get that f_1, g_1 share a root and this contradicts the fact that $gcd(f_1, g_1) = 1$.

If $d(x) = 0$ then since $f_1(x) = 0$ and $x^{2^k} - x = df_1$ we get that $x^{2^k} - x$ has a double root.

But this is impossible since the derivative of $x^{2^k} - x$ is -1.

So, $x \notin \mathbf{V}_{\mathbb{F}_{2^k}}(g)$ and this concludes the proof. □

Let $J_0 = \langle x_1^{2^k} - x_1, \dots, x_n^{2^k} - x_n \rangle$ denote the ideal of all vanishing polynomials in $\mathbb{F}_{2^k}$. Then, we have $V(J_0) = (\mathbb{F}_{2^k})^n$; i.e. the variety of vanishing ideal contains all possible valuations of variables, so it constitutes the **universal set**. Subsequently, the **absolute complement** can be computed as:

Corollary II.1.

$$V(J') = \overline{\mathbf{V}(J)} = \mathbf{V}(J_0 : J)$$

In other words, the ideal J', computed as $J' = J_0 : J$ is such that $V(J') = \overline{V(J)}$.

Since we can add vanishing polynomials for state sets $from^i, to^i, reached$, it is feasible to use above corollary. We will demonstrate the application of this important concept through an example.

Assume $I = \langle f \rangle = \langle T^2 + (1+\alpha)\cdot T + \alpha \rangle$, we can add vanishing polynomial $T^4 - T$ to this ideal such that its variety $V(I) = \{a \mid a \in \mathbb{F}_{2^2} \textit{ and } f(a) = 0\} = \{1, \alpha\}$. For complement set of variety $\{1,\alpha\}$, the universal set is the variety of ideal of vanishing polynomial $V(\langle T^4 - T\rangle) = \{0, 1, \alpha, 1+\alpha\}$, so $\overline{V(\langle T^2 + (1+\alpha)\cdot T + \alpha\rangle)} = V(\langle T^4 - T\rangle) - V(\langle T^2 + (1+\alpha)\cdot T + \alpha\rangle)$, which equals to $V(\langle T^4 - T\rangle : \langle T^2 + (1+\alpha)\cdot T + \alpha\rangle) = V(\langle T^2 + (1+\alpha)\cdot T\rangle)$, the result is $\{0, 1+\alpha\}$.

According to the discussion above, the BFS traversal can be completely implemented using algebraic geometry. as in Algorithm 2.

ALGORITHM 2: Algebraic Geometry based Traversal Algorithm

```
Input: Input-output circuit characteristic polynomial ideal $J_{ckt}$,
       initial state polynomial $\mathcal{F}(S)$
1 $from^0 = reached = \mathcal{F}(S)$;
2 repeat
3     $i \leftarrow i + 1$;
4     $to^i \leftarrow$GB w/ elimination term order$\langle J_{ckt}, J_0, from^{i-1}\rangle$;
5     $new^i \leftarrow$generator of $\langle to^i \rangle + (\langle T^{2^k} - T\rangle : \langle reached \rangle)$;
6     $reached \leftarrow$generator of $\langle reached \rangle \cdot \langle new^i \rangle$;
7     $from^i \leftarrow new^i(S \setminus T)$;
8 until $new^i == 1$;
9 return $reached$
```

Here $from^i, to^i, new^i$ are all univariate polynomials in variables S or T, due to GB computation with elimination term order.

Example II.2. *We apply Algorithm 2 to Ex.**??** to execute the FSM traversal. Let the initial state $from^0 = S$ (i.e. state $\{00\}$), in the first iteration we compose an elimination ideal J with*

$$f_1 : t_0 - (xs_0s_1 + xs_0 + xs_1 + x + s_0 + s_1 + 1)$$
$$f_2 : t_1 - (xs_0 + x + s_0s_1 + s_0)$$
$$f_3 : S - s_0 - s_1$$
$$f_4 : T - t_0 - t_1$$

$$J_{ckt} = \langle f_1, f_2, f_3, f_4 \rangle$$

$$f_5 : x^2 - x$$
$$f_6 : s_0^2 - s_0, \quad f_7 : s_1^2 - s_1$$
$$f_8 : t_0^2 - t_0, \quad f_9 : t_1^2 - t_1$$
$$f_{10} : S^4 - S, \ f_{11} : T^4 - T$$

$$J_0 = \langle f_5, f_6, \ldots, f_{11} \rangle$$

and $from^0$. Compute the reduced GB for $J = J_{ckt} + J_0 + \langle from^0 \rangle$, with elimination term order

$$\{x, s_0, s_1, t_0, t_1\} \ (\textit{all PI and bit level variables}) > S \ (\textit{PS word}) > T \ (\textit{NS word})$$

the result contains a polynomial generator with only T, we assign it to next state

$$to^1 = T^2 + (\alpha+1)T + \alpha$$

denoting $\{01, 10\}$. Since formerly reached state $reach = T$, its complement is

$$\langle T^4 - T\rangle : \langle T \rangle = T^3 + 1$$

(i.e. states $\{01, 10, 11\}$). Then the newly reached state in this iteration is

$$\langle T^3 + 1, T^2 + (\alpha+1)T + \alpha\rangle = \langle T^2 + (\alpha+1)T + \alpha\rangle$$

We add these states to formerly reached states

$$reach = \langle T \cdot T^2 + (\alpha+1)T + \alpha\rangle = \langle T^3 + (\alpha+1)T^2 + \alpha T\rangle$$

(i.e. states $\{00, 01, 10\}$). We update the present states for next iteration

$$from^1 = S^2 + (\alpha+1)S + \alpha$$

In the second iteration, we compute the reduced GB with the same term order for ideal $J = J_{ckt} + J_0 + \langle from^1 \rangle$. It includes a polynomial generator

$$to^2 = T^2 + \alpha T$$

denotes states $\{00, 10\}$. The complement of reached is

$$\langle T^4 - T\rangle : \langle T^3 + (\alpha+1)T^2 + \alpha T\rangle = T + 1 + \alpha$$

(i.e. states $\{11\}$). We compute the newly reached state

$$\langle T^2 + \alpha T, T + 1 + \alpha\rangle = \langle 1 \rangle$$

It means the newly reached state is empty, thus a fixpoint has been detected. The algorithm terminates and returns

$$reached = \langle T^3 + (\alpha+1)T^2 + \alpha T\rangle$$

as final reachable states.

Significance for using GB: Reduced GB is a unique, minimal and **canonical** representation of the circuit's function. Starting from a certain initial state and using a reduced GB to represent the transition function, reachable states can be computed and represented canonically. Then it becomes possible to identify when a fixpoint is reached (termination of the algorithm) by performing an equality check of polynomial ideals.

III. Refined Algorithm

Within the naive algorithm described in last section, Gröbner basis is computed for each iteration to attain the word-level next state polynomial representation. In practice, for a sequential circuit with complicated structure and large data-path, Gröbner basis computation is intractable. To overcome the high computational complexity brought by GB computing, we managed to find a method obviating directly computing GB; more specifically, we turn GB computation into one-step multivariate polynomial division. In this way, our proposed approach can be used on larger sequential circuit verification.

A. Refined abstraction term order (RATO)

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).*

In Buchberger's algorithm, the specification polynomial (Spoly) for each pair is calculated. In 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. Its remainder r lets following lemma hold:

Lemma III.1. *r will only contain primary inputs (bit-level) and word-level output; furthermore, there will be one and only one term containing word-level output whose monomial is word-level output itself rather than higher order form.*

Proof. First proposition is easy to prove by contradiction. Second part, the candidate pair of polynomials only have one term of single word-level output variable (say it is R) and this term is the last term under RATO, which means there is only one term with R in Spoly. Meanwhile in other polynomials from $J + J_0$ there is no such term containing R, so this term will be kept to remainder r, in first degree. □

Example III.1. *After adding intermediate bit-level signal a, b, c, d, the elimination ideal for example circuit (Ex.??) could be rewritten under RATO:*

$$(t_0, t_1) > (a, b, c, d)$$
$$> T > (x, s_0, s_1)$$

Thus the candidate pair is $(f_w, f_g), f_w = t_0 + a \cdot b + a + b, f_g = t_0 + t_1\alpha + T$. Result after reduction is:

$$Spoly(f_w, f_g) \xrightarrow{J+J_0}_+$$
$$T + s_0s_1x + s_0s_1\alpha + (1+\alpha)s_0x + (1+\alpha)s_0 + s_1x + s_1 + (1+\alpha)x + 1$$

The remainder satisfies Lemma III.1.

B. Bit-level Variable Substitution (BLVS)

The remainder from *Spoly* contains bit-level PS variables, and our objective is to get a polynomial contains only word-level PS variables . One possible method is to rewrite bit-level variables in term of function on word-level variables, i.e. $s_i = \mathcal{G}(S)$, then do substitution. A Gaussian-elimination-fashion approach could be applied to compute corresponding $\mathcal{G}(S)$ efficiently.

Example III.2. Objective*: Abstract polynomial $s_i + \mathcal{G}_i(S)$ from $f_0 : s_0 + s_1\alpha + S$.*

First, compute $f_0^2 : s_0 + s_1\alpha^2 + S^2$. Apparently variable s_0 can be eliminated by operation

$$f_1 = f_0 + f_0^2 :$$
$$(\alpha^2 + \alpha)s_1 + S^2 + S$$

Now we can solve univariate polynomial equation $f_1 = 0$ and get solution

$$s_1 = S^2 + S$$

Use this solution we can easily solve equation $f_0 = 0$. The result is

$$s_0 = \alpha S^2 + (1+\alpha)S$$

In this approach it is easy to get word-level variable representation for each bit-level PS variables. By substitution, a new polynomial in word-level PS/NS variables could be attained.

After processing with RATO and BLVS techniques, we get a polynomial in form of $f_T = T + \mathcal{F}(S, x)$ denoting the transition function. We add a polynomial about S to define the present states f_S, as well as a polynomial guarantee free-end signals for primary inputs $f_x = x^2 + x$, using elimination term order $S > x > T$, we can generate a GB out of elimination ideal $\langle f_T, f_S, f_x \rangle$, which contains a univariate polynomial denoting next states. The new scalable algorithm can be depicted as following:

ALGORITHM 3: Refined Algebraic Geometry based Traversal Algorithm

```
Input: Input-output circuit characteristic polynomial ideal $J_{ckt}$,
       initial state polynomial $\mathcal{F}(S)$
1  $from^0 = reached = \mathcal{F}(S)$;
2  $f_T$ =Reduce w/ RATO($J_{ckt}$);
3  repeat
4      $i \leftarrow i + 1$;
5      $to^i \leftarrow$GB w/ elimination TO$\langle f_T, from^{i-1}, f_x \rangle$;
6      $new^i \leftarrow$generator of $\langle to^i \rangle + (\langle T^4 - T \rangle : \langle reached \rangle)$;
7      $reached \leftarrow$generator of $\langle reached \rangle \cdot \langle new^i \rangle$;
8      $from^i \leftarrow new^i(S \setminus T)$;
9  until $new^i == 1$;
10 return $reached$
```

C. PI Partition

Using above techniques we can get a remainder polynomial with only word-level PS/NS variables. However in most cases, the number of bit-level PIs will be too large for the last-step Gröbner basis computation. Therefore it is necessary to convert bit-level PIs to word-level PI variables.

Assume we have k-bit datapath and n-bit PIs. In Galois field, we need to carefully partition n PIs to guarantee states of all partitions can be covered by single univariate polynomials.

Proposition III.1. *Divide n-bit PIs to partitions $n_1, n_2, \ldots, n_s$ and let $n_1, n_2, \ldots, n_s$ bit word-level variables represent their evaluations in $\mathbb{F}_{2^k}$ accordingly. Then there always be $n_1, n_2, \ldots, n_s \mid k$.*

Again, assume a partition $n_i \mid k$ and corresponding word-level variable is P. Then we can use polynomial $P^{2^n_i} - P$ to represent all signals at free-end PIs, according to following theorem about **composite field**:

Theorem III.1. *Let $k = m\dot{n}_i$, such that $\mathbb{F}_{2^k} = \mathbb{F}_{(2^{n_i})^m}$. Let α be primitive root of $\mathbb{F}_{2^k}$, β be primitive root of the ground field $\mathbb{F}_{2^{n_i}}$. Then*

$$\beta = \alpha^{\omega}, \quad where \quad \omega = \frac{2^k - 1}{2^{n_i} - 1}$$

Example III.3. *In a sequential circuit, PS/NS inputs/outputs are 4-bit signals, which means we will use $\mathbb{F}_{2^4}$ as working field. PIs are partitioned to 2-bit vectors, which means the ground field is $\mathbb{F}_{2^2}$. In ground field we can represent all possible evaluations of this PI partition $\{p_0, p_1\}$ with*

$$P^4 + P, \quad where \quad P = p_0 + p_1 \cdot \beta$$

Using Thm.III.1 we get $\beta = \alpha^5$, so we can redefine word P as element from $\mathbb{F}_{2^4}$:

$$P = p_0 + p_1 \cdot \alpha^5$$

Using this method we can efficiently partition large size PIs to small number of word-level PI variables. One limitation of this approach is PIs cannot be partitioned when k is prime.

IV. Application: Unrolling Sequential Circuits for Sequential Arithmetic Verification

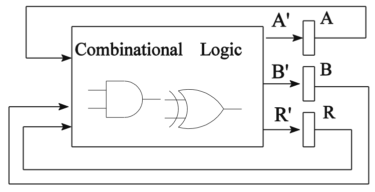
In above discussions, we propose an implicit state enumeration algorithm implemented by GB computations over an elimination ideal, which is usually applied on FSMs working as control units. For many arithmetic datapath components, state enumeration is not the best way to verify the function of circuit. On the one hand, the signals preloaded to the state variables in FSM are usually not specified such that we can hardly encode the initial states; on the other hand, instead of the set of globally reachable states, arithmetic verification focuses on exact reached states after k clock cycles which reflects desired arithmetic function. Considering these reasons, we modify our algebraic geometry based approach and apply it to implicit FSM unrolling.

Fig.1(b) shows a typical Moore FSM without primary inputs, R is the word-level state variable. Let word-level indeterminate R_{init} denote the initial preloaded value of R, and R_i denote the evaluation of R after i clock cycles (transitions). Transition relation $Tr : R_{i-1} \to R_i$ can be modeled similar to the image function in Algorithm 1. Assume that after k clock cycles, the machine should give a state output $R_k = \mathcal{F}(R_{init})$. To verify this result, we can implicitly unroll this FSM and execute transition function at word level for k times, which is $R_k = Tr(R_{k-1}) = Tr(Tr(\cdots Tr(R_{init})\cdots)) = Tr^k(R_{init})$.

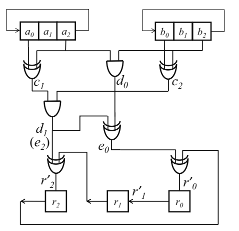
Sequential Galois field (GF) multiplier is a typical sequential arithmetic component widely used in cryptographic systems. Since there is no primary input in the design, it can be regarded as Moore FSM (Fig.2(a)) where $R = \sum_{i=0}^{k-1} r_i \beta^{2^i}$, $A = \sum_{i=0}^{k-1} a_i \beta^{2^i}$, $B = \sum_{i=0}^{k-1} b_i \beta^{2^i}$. Notice that β is the *normal element* which can be represented by a power of primitive element $\alpha: \ \beta = \alpha^t$. The states are encoded by evaluations of state variables, which can be implicitly verified by checking output-input function ($R = A \cdot B \pmod{P(\alpha)}$).

In this proposal we use the verification of a sequential GF multiplier – a 3-bit RH-SMPO invented by Reyhani-Masoleh and Hasan[6] – to illustrate our approach. The normal element is given as $\beta = \alpha^3$, and primitive polynomial $P(\alpha) = \alpha^3 + \alpha + 1$. The gate-level circuit is depicted in Fig.2(b).

We follow the typical sequential GF circuit model with word-level variables A, B, R denoting *present states (PS)* and A', B', R' denoting *next states (NS)* of the machine; where $A = \sum_{i=0}^{k-1} a_i \beta^{2^i}$ for the PS variables and $A' = \sum_{i=0}^{k-1} a'_i \beta^{2^i}$ for NS variables, and so on. Variables R (R') correspond to those that store the result, and A, B (A', B') store input operands. *E.g.,* for a GF multiplier, A_{init}, B_{init} (and $R_{init} = 0$) are the initial values (operands) loaded into the registers, and $R = \mathcal{F}(A_{init}, B_{init}) = A_{init} \times B_{init}$ is the final result after k-cycles. Our approach aims to find this polynomial representation for R.



(a)



(b)

Fig. 2: A 3-bit RH-SMPO and its Moore FSM model

Each gate in the combinational logic is represented by a Boolean polynomial. To this set of Boolean polynomials, we append the polynomials that define the word-level to bit-level relations for PS and NS variables ($A = \sum_{i=0}^{k-1} a_i \beta^{2^i}$). We denote this set of polynomials as ideal $J = \langle f_1, \dots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \dots, x_d, R, R', A, A', B, B']$, where $x_1, \dots, x_d$ denote the bit-level (Boolean) variables of the circuit. The ideal of vanishing polynomials J_0 is also included, and then the implicit FSM unrolling problem is setup for abstraction.

The configurations of the flip-flops are the states of the machine. *Since the set of states is a finite set of points, we can consider it as the variety of an ideal related to the circuit* . Moreover, since we are interested in the *function encoded* by the state variables (over k-time frames), we can *project this variety* on the word-level state variables, starting from the initial state A_{init}, B_{init}. Projection of varieties (geometry) corresponds to elimination ideals (algebra), and can be analyzed via Gröbner bases. Therefore, we employ a Gröbner basis computation with abstraction term order (ATO)[1]: we use a *lex term order* with *bit-level variables > word-level NS outputs > word-level PS inputs*. This allows to eliminate all the bit-level variables and derives a representation only in terms of words. Consequently, k-successive Gröbner basis computations implicitly unroll the machine, and provide word-level algebraic k-cycle abstraction for R' as $R' = \mathcal{F}(A_{init}, B_{init})$.

Algorithm 4 describes our approach. In the algorithm, $from_i$ and to_i are polynomial ideals whose varieties are the valuations of word-level variables R, A, B and R', A', B' in the i-th iteration; and the notation "\" signifies that the *NS* in iteration (i) becomes the *PS* in iteration ($i+1$). Line 5 computes the Gröbner basis with the abstraction term order. Line 6 computes the elimination ideal, eliminating the bit-level variables and representing the set of reachable states up to iteration i in terms of the elimination ideal. These computations are analogous to those of image computations performed in FSM reachability.

Example IV.1. *We demonstrate our approach to verify the 3-bit RH-SMPO circuit of Fig.2. The normal element β in $\mathbb{F}_{2^3}$ is known to be $\beta = \alpha^3$, where α is the primitive element. The circuit can be described with an ideal by translating AND and*

ALGORITHM 4: Abstraction via implicit unrolling for Sequential GF circuit verification

```
Input: Circuit polynomial ideal $J$, vanishing ideal $J_0$, initial
       state ideal $R(=0), \mathcal{G}(A_{init}), \mathcal{H}(B_{init})$
1 $from_0(R,A,B) = \langle R, \mathcal{G}(A_{init}), \mathcal{H}(B_{init})\rangle$;
2 $i = 0$;
3 repeat
4     $i \leftarrow i+1$;
5     $G \leftarrow$GB$(\langle J + J_0 + from_{i-1}(R,A,B)\rangle)$ with ATO;
6     $to_i(R',A',B') \leftarrow G \cap \mathbb{F}_{2^k}[R',A',B',R,A,B]$;
7     $from_i \leftarrow to_i(\{R,A,B\} \setminus \{R',A',B'\})$;
8 until $i == k$;
9 return $from_k(R_{final})$
```

XOR gates accordingly. For the first iteration:

$$\begin{aligned} J =& d_0 + b_2 \cdot a_2, c_1 + a_0 + a_2, c_2 + b_0 + b_2, d_1 + c_1 \cdot c_2,\\ & e_0 + d_0 + d_1, e_2 + d_1, r_0' + r_2 + e_0, r_1' + r_0, r_2' + r_1 + e_2,\\ & A + a_0\alpha^3 + a_1\alpha^6 + a_2\alpha^{12}, B + b_0\alpha^3 + b_1\alpha^6 + b_2\alpha^{12},\\ & R + r_0\alpha^3 + r_1\alpha^6 + r_2\alpha^{12}, R' + r_0'\alpha^3 + r_1'\alpha^6 + r_2'\alpha^{12}; \end{aligned}$$

The last 4 polynomials are translated from the definition of word-level variables. This represents ideal "J" from line 5 in Algorithm 4. "J_0" is the ideal of vanishing polynomials in all bit-level variables (e.g. $a_0^2 - a_0$) and word-level variables (e.g. $A^8 - A$). "$from_{i-1}$" represents the set of current states for iteration i. In the first iteration, $from_0 = \{R, A_{init} + a_0\alpha^3 + a_1\alpha^6 + a_2\alpha^{12}, B_{init} + b_0\alpha^3 + b_1\alpha^6 + b_2\alpha^{12}\}$.

After the GB computation is performed with ATO, as line 6 in Algorithm 4, we find a polynomial in variables R', A_{init}, B_{init} in to_1 : $R' + (\alpha^2)A_{init}^4B_{init}^4 + (\alpha^2+\alpha)A_{init}^4B_{init}^2 + (\alpha^2+\alpha)A_{init}^4B_{init} + (\alpha^2+\alpha)A_{init}^2B_{init}^4 + (\alpha^2+\alpha+1)A_{init}^2B_{init}^2 + (\alpha^2)A_{init}^2B_{init} + (\alpha^2+\alpha)A_{init}B_{init}^4 + (\alpha^2)A_{init}B_{init}^2$.

Line 7 in Algorithm 4 simply replaces NS output R' with PS output R in this example; so in second iteration $from_1 = \{R' + (\alpha^2)A_{init}^4B_{init}^4 + (\alpha^2+\alpha)A_{init}^4B_{init}^2 + (\alpha^2+\alpha)A_{init}^4B_{init} + (\alpha^2+\alpha)A_{init}^2B_{init}^4 + (\alpha^2+\alpha+1)A_{init}^2B_{init}^2 + (\alpha^2)A_{init}^2B_{init} + (\alpha^2+\alpha)A_{init}B_{init}^4 + (\alpha^2)A_{init}B_{init}^2, A_{init} + a_2\alpha^3 + a_0\alpha^6 + a_1\alpha^{12}, B_{init} + b_2\alpha^3 + b_0\alpha^6 + b_1\alpha^{12}\}$.

Finally, after 3 iterations we obtain: $to_3 = \{\mathbf{R' + A_{init}B_{init}}, A_{init} + a_0'\alpha^3 + a_1'\alpha^6 + a_2'\alpha^{12}, B_{init} + b_0'\alpha^3 + b_1'\alpha^6 + b_2'\alpha^{12}\}$ as the image. The final result is $from3(R_{final}) = R_{final} + A_{init} \cdot B_{init}$, which verifies the multiplier.

Using our approach we successfully verified the function of an 100-bit RH-multiplier, which implies its capability to handle the verification of large designs. The results are published in [7].

References

[1] Tim Pruss, Priyank Kalla, and Florian Enescu, "Equivalence verification of large galois field arithmetic circuits using word-level abstraction via gröbner bases", *in Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, pp. 1–6. ACM, 2014.

[2] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms*, Springer-Verlag, 1997.

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

[4] Priyank Kalla and Maciej Ciesielski, "A comprehensive approach to the partial scan problem using implicit state enumeration", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, pp. 810–826, 2002.

[5] S. Gao, A. Platzer, and E. Clarke, "Quantifier Elimination over Finite Fields with Gröbner Bases", *in Intl. Conf. Algebraic Informatics*, 2011.

[6] Arash Reyhani-Masoleh and M Anwar Hasan, "Low complexity word-level sequential normal basis multipliers", *Computers, IEEE Transactions on*, vol. 54, pp. 98–110, 2005.

[7] Xiaojun Sun, Priyank Kalla, Tim Pruss, and Florian Enescu, "Formal verification of sequential galois field arithmetic circuits using algebraic geometry", *in Design Automation and Test in Europe, DATE 2015. Proceedings*. IEEE/ACM, 2015.