SEQUENTIAL CIRCUIT VERIFICATION AT WORD LEVEL USING ALGEBRAIC GEOMETRY

Xiaojun Sun
A Ph.D Dissertation Proposal
Electrical & Computer Engineering, University of Utah
Spring Semester 2015

Abstract

This PhD dissertation proposal addresses the problem of sequential circuit verification at the word-level and is based on concepts derived from algebraic geometry. Analyzing the sequential circuits at word level is an efficient way of *abstraction*, which may lower the complexity of verification by efficient representation of the state-space. We propose to model the verification properties and the gate-level sequential circuit implementations over Galois fields of the type $\mathbb{F}_{2^k}$ by means of polynomial ideals and their canonical representations — Gröbner bases — at the level of k-bit words. Subsequently, techniques from algebraic geometry can be used to reason about the state-space by analyzing varieties of these ideals.

We propose to apply these techniques to traverse a finite state machine (FSM) for reachability analysis at the word-level, and also to implicitly unroll sequential arithmetic circuits to verify their function. Moreover, as unsatisfiable (UNSAT) cores play an important role in modern abstraction-refinement techniques for verification, we propose to investigate a word-level analogue of the UNSAT core problem using the Gröbner basis algorithm. The proposal will not only derive new algorithms and techniques, but will also consider efficient CAD implementations to target sequential equivalence and model checking problems.

I. INTRODUCTION

Verification of sequential circuits is challenging, and therefore a topic of continuous research. With the increasing size of new integrated circuits, sequential circuit designers face much more complicated problems of design errors in specification models and implementations. These errors are usually modeled as "bad" states, and the circuits/functional components are modeled as state machines. Once state reachability is analyzed, the existence of errors can be identified by judging whether "bad" states are reachable from certain initial states. Reachability analysis can be performed by various techniques such as extensive simulation, temporal logic model checking and property directed reachability (PDR). In this proposal we discuss a new promising approach that applies implicitly on the word level and borrows concepts from commutative algebra and algebraic geometry.

There are several advantages in exploiting word-level verification: on the one hand, a number of circuit designs have their datapaths and/or system-level models described at the word-level, which include many register transfer level (RTL) descriptions, arithmetic components and cryptographic systems. Using word-level information to do verification is convenient and beneficial for interpreting the circuit's function. On the other hand, using word-level instead of bit-level information is one way of "abstraction". Abstraction is a key technique to reduce the state space of a finite state machine when analyzing a sequential circuit. It is usually done by combining sets of states with similar properties. During state reachability analysis, if we use bit-level variables to represent the states, the representations may become too large to handle (e.g. BDD explosion). However, when a "bundle" of bit-level variables are represented as only one word-level variable, the set of reachable states can be represented by a word-level constraint expression. Word-level verification techniques are effective in lowering the complexity of state reachability analysis.

Algebraic geometry can provide a symbolic representation of both bit-level (Boolean) variables and word-level variables *in one unified framework*. The basic idea is that values of state variables can be represented as solutions to a finite set of polynomials. Gröbner basis (GB) techniques can be subsequently applied to transform such polynomial systems into a unique canonical form, thus facilitating verification. To represent Boolean circuits at word level, polynomial functions over Galois fields provide a *bit-precise word-level model*. Particularly, arbitrary word-level datapaths can be modeled as polynomial functions over $\mathbb{F}_{2^k}$ – i.e. the Galois field of 2^k elements, where k represents the bit-vector word length. In this fashion, the states and transition relations can be represented as polynomial functions over $\mathbb{F}_{2^k}$ and the power of algebraic geometry can be applied to implement symbolic reasoning about state reachability. This kind of implicit state analysis and abstraction can overcome scalability issues encountered by explicit bit-blasting. Recent work of T. Pruss [1], J. Lv [2], A. Lvov [3] and M. Brickenstein [4] shows that it is possible to make application of algebraic geometry practical for circuits.

This proposal investigates a wide gamut of symbolic computation and algebraic geometry techniques for reachability analysis, abstraction and verification of sequential circuits applied at the word level.

The essence of our approach relies on bit-level to word-level abstraction. Any Boolean function over k-bit vectors $f : \mathbb{B}^k \to \mathbb{B}^k$ can be uniquely mapped to a polynomial function over k-bit words in the Galois field $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ [1]. Therefore, the transition

relation of sequential circuits can be modeled at word level. The set of states (Boolean encoding) can be represented as (word-level) elements over Galois fields. Verification requires operations such as image computations, set intersections, unions and complements, unsatisfiability (UNSAT) proofs and analysis, etc. The challenge is to discover efficient algorithmic techniques to perform all such operations at the word level. While this challenging task is the topic of this research, we demonstrate the feasibility of this approach in the example below.

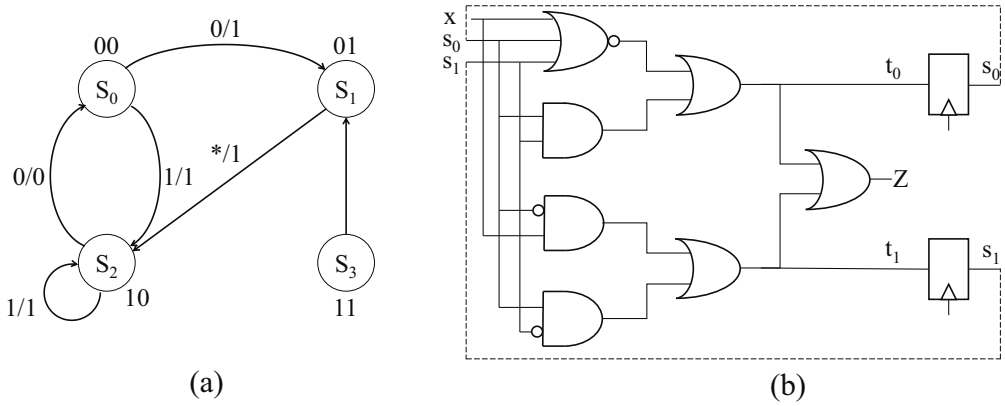

Fig. 1: 2-bit FSM example circuit

Example I.1. Motivating example: *Consider Fig.1(a), a Mealy finite state machine (FSM) with 4 states encoded by a 2-bit Boolean vector, and Fig.1(b) as its gate-level circuit implementation. Variable x is the primary input (PI), $\{s_0, s_1\}$ are present state (PS) variables, Z is primary output (PO) and $\{t_0, t_1\}$ are next state (NS) variables. The 4 states are encoded as $\{00, 01, 10, 11\}$, which can be interpreted as elements in $\mathbb{F}_4$ according to Table I. We can also translate all gates (Boolean operators) to polynomial functions over $\mathbb{F}_4$.*

TABLE I: Mapping bit-vector state encoding to elements in $\mathbb{F}_4$, Boolean operators to functions over $\mathbb{F}_4$

Bit-vector	Element in $\mathbb{F}_4$	Boolean operator	Function over $\mathbb{F}_4$
00	0	$a \wedge b$	$a \cdot b$
01	1	$a \oplus b$	$a + b$
10	α	$\bar{a}$	$1 + a$
11	$1 + \alpha$	$a \vee b$	$a + b + a \cdot b$

We first obtain polynomials f_1 and f_2, denoting transition relations for NS $\{t_0, t_1\}$:

$$\begin{aligned} f_1: &\quad t_0 - (xs_0s_1 + xs_0 + xs_1 + x + s_0 + s_1 + 1) \\ f_2: &\quad t_1 - (xs_0 + x + s_0s_1 + s_0) \end{aligned}$$

Since we model reachability at word level, let $S = \{s_0, s_1\}$ represent a word-level PS variable in $\mathbb{F}_4$, its relation to bit-level variables is $S = s_0 + s_1 \cdot \alpha$, where α is the primitive element of $\mathbb{F}_4$. Similarly we can define a word-level NS variable T. They can be both interpreted in polynomial form:

$$\begin{aligned} f_3: &\quad S - s_0 - s_1\alpha \\ f_4: &\quad T - t_0 - t_1\alpha \end{aligned}$$

The initial state can also be represented in polynomial form. Assume the initial state is $\{00\}$ (0 in $\mathbb{F}_4$), it corresponds to the solution of the polynomial equation $(S - 0) = 0$. So its polynomial form is

$$f_5: \; S$$

We can then use the Gröbner basis (GB) computation to compute a canonical form of this system of polynomials $f_1, \ldots, f_5$ representing possible value of NS variable T. Note that if we eliminate the PI variable x from the ideal $\langle f_1, \ldots, f_5 \rangle$ while computing GB, the result implies an existential quantification on x.

After computing GB with a lexicographic term order $\{x, s_0, s_1, t_0, t_1\} > S > T$, the result contains a polynomial only in the word-level variable T: $\mathbf{g_T : T^2 + (\alpha + 1)T + \alpha}$. Notice that g_T is a polynomial in word-level variable T, it has 2 roots which encode 2 states of the machine $T = 1(01)$ and $T = \alpha(10)$.

The above example shows how algebraic geometry can be applied to induce one-step reachability at the word level. It is possible to combine our approach in this example with other techniques inspired by concepts and theorems from algebraic geometry to help reachability analysis for FSMs, as well as further formal verification problems on sequential circuits.

Contributions of the dissertation: Our approach to solve the sequential circuit verification problem and contribution of the thesis can be described as follows:

- This proposal will investigate techniques to perform sequential circuit verification at the word level using polynomial abstractions of Boolean circuits over Galois fields. The applications considered are exact reachability analysis and property directed reachability [5] [6]. Both sequential datapath circuits as well as random logic controllers will be considered.
- Algebraic geometry based techniques will be researched for reachability analysis. The concepts of Nullstellensatz, Gröbner bases and elimination ideals will be explored for the purpose of image computations. The canonical representations of polynomial ideals provided by Gröbner bases will be exploited for verification.
- Computing Gröbner bases for large circuits is infeasible due to its high computational complexity. Therefore, based on inspirations from [1] and [2], a significant part of this work will focus on improving the scalability of our approach.
- New paradigms for abstraction refinement using algebraic geometry will be explored. Such techniques require analysis of UNSAT proofs of the verification instances. UNSAT cores in polynomial calculus have not been researched before. In this proposal, we will investigate a new technique to identify minimal UNSAT cores using the Gröbner basis algorithm and apply it to verification.

All in all, this proposal aims to discover an altogether new paradigm for formal verification/model checking of sequential circuits at the word level using a combination of design automation, algebraic geometry and symbolic computation techniques. Bit-level Gröbner basis computation has been applied to model checking [7] [8] and satisfiability problems [4], but Gröbner basis at word level has never been explored before for circuit verification.

This proposal is organized as follows: Section II reviews preliminary finite state machine and algebraic geometry concepts related to ideals, varieties, Gröbner bases, elimination ideals, etc. Section III describes our proposed approach to perform implicit state enumeration at word level. An algorithm to extract UNSAT cores based on algebra geometry techniques is conjectured in Section IV. Section V outlines the proposed objectives. Finally, Section VI outlines the proposed research timeline.

II. Preliminaries

In this section, we will briefly review the preliminary concepts related to finite state machine models, Galois fields, algebraic geometry and Gröbner bases, as our approaches are based on a combination of these concepts.

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 II.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;*
- $\Delta: \ S \times \Sigma \rightarrow S$ *is the next state transition function;*
- $\Lambda: \ S \times \Sigma \rightarrow O$ *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

$$\Lambda: \ S \rightarrow O$$

Typical sequential circuits can be depicted as Fig.2(a). Primary inputs $x_1, \ldots, x_m \in \Sigma$, and primary outputs $z_1, \ldots, z_n \in O$. Signals $s_1, \ldots, s_k$ are present state (PS) variables, $t_1, \ldots, t_k$ 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: $S = (s_1, \ldots, s_k), \ T = (t_1, \ldots, t_k)$. Transition function at bit level are defined as $\Delta_i : t_i = \Delta_i(s_1, \ldots, s_k, x_1, \ldots, x_m)$. 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.2(b).

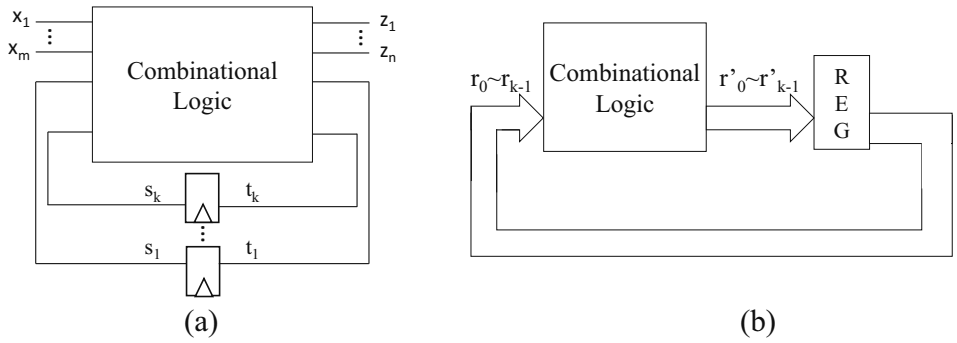

Fig. 2: FSM models of sequential circuits

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):

$$a \wedge b \rightarrow a \cdot b$$
$$a \oplus b \rightarrow a + b$$
$$\neg a \rightarrow 1 + a$$
$$a \bar{\oplus} b \rightarrow 1 + a + b$$
$$a \vee b \rightarrow a + b + a \cdot b$$

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.I.1, 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 $f_1, f_2, \ldots, f_5 \subseteq \mathbb{F}_{2^k}$.

It is well-known that every Boolean mapping between k dimensional Boolean spaces $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$ can be construed as a function over Galois fields $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$. Moreover, every function $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$ 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 $x_1, x_2, \ldots, x_n$, 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 $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ are monomials. The set of *all* such polynomials with coefficients from $\mathbb{F}_{2^k}$ forms a **multivariate polynomial ring** denoted $\mathbb{F}_{2^k}[x_1, \ldots, x_n]$. A monomial ordering $X_1 > X_2 \cdots > X_t$ is imposed on the polynomials to process them systematically. Then, $LT(f) = c_1 X_1, LM(f) = X_1$ 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 II.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 $F = \{f_1, \ldots, f_s\}$, 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 II.2. *Given a set of polynomials $f_1, \ldots, f_s$ over ring $\mathbb{F}_q[x_1, \ldots, x_n]$, their* **affine variety**

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

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

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

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

A practical problem is: given an ideal $J = \langle f_1, f_2, \ldots, f_s \rangle$ 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 II.4. Ideal membership: *Let $f_1, f_2, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$, and $J = \langle f_1, f_2, \ldots, f_s \rangle$ be an ideal over ring $\mathbb{F}[x_1, \ldots, x_n]$. If*

$$f = f_1 h_1 + f_2 h_2 + \cdots + f_s h_s$$

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: $\langle f_1, \ldots, f_s \rangle = \langle h_1, \ldots, h_r \rangle = \langle g_1, \ldots, g_t \rangle$ such that $V(f_1, \ldots, f_s) = V(h_1, \ldots, h_r) = V(g_1, \ldots, g_t)$. Therefore a canonical representation of an ideal is needed, which leads to the concept of Gröbner bases.

Definition II.5. *The set $G = \{g_1, \ldots, g_t\}$ 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.

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

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 $J = \langle f_1, \ldots, f_s \rangle$, $G = \{g_1, \ldots, g_t\}$ is the GB of J, it can be computed by Buchberger's algorithm (refer to textbook [9]).

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 $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$. 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 $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$, the l^{th} elimination ideal J_l is the ideal of $\mathbb{F}_q[x_{l+1}, \ldots, x_d]$ defined by $J_l = J \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d]$. Variable elimination can be achieved by computing a Gröbner basis of J w.r.t. elimination orders:

Theorem II.1. *(Elimination Theorem[10]) Let $J \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ 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 $0 \leq l \leq d$, the set $G_l = G \cap \mathbb{F}_{2^k}[x_{l+1}, \ldots, x_d]$ is a Gröbner basis of the l-th elimination ideal J_l.*

We describe an application of elimination ideals using following example borrowed from [10]:

Example II.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 II.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 $A = (a_0, \ldots, a_{k-1})$ and output $R = (r_0, \ldots, r_{k-1})$ (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 II.1. *(From [1]) Given a combinational circuit* C *with* k*-bit input* $A = (a_0, \ldots, a_{k-1})$ *and* k*-bit output* $R = (r_0, \ldots, r_{k-1})$. *Denote by* $x_1, \ldots, x_d$ *all the bit-level variables of* C. *Let* $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d, R, A]$ *denote all the polynomials corresponding to the logic gates of the circuit. Let* $J_0 = \langle x_1^2 - x_1, \ldots, x_d^2 - x_d, R^q - R, A^q - A \rangle$ *be the vanishing ideal, so that* $J + J_0 = \langle f_1, \ldots, f_s,\ \ x_1^2 - x_1, \ldots, x_d^2 - x_d, R^q - R, A^q - A \rangle$. *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* $x_1, \ldots, x_d$ *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.I.1, 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.

III. 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 II.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 [11], 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.1. *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.1(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:

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

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 Δ, initial state S^0

1 $from^0 = reached = S^0$;
2 **repeat**
3 $\quad i \leftarrow i+1$;
4 $\quad to^i \leftarrow \mathrm{Img}(\Delta, from^{i-1})$;
5 $\quad new^i \leftarrow to^i \cap \overline{reached}$;
6 $\quad reached \leftarrow reached \cup new^i$;
7 $\quad from^i \leftarrow new^i$;
8 **until** $new^i == 0$;
9 **return** *reached*

Example III.1. *For the circuit in Fig. 1 (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** [12]. 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.1(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 III.1. *(***Sum/Product of Ideals***) If* $I = \langle f_1, \ldots, f_r \rangle$ *and* $J = \langle g_1, \ldots, g_s \rangle$ *are ideals in* $\mathbb{F}[x_1, \ldots, x_n]$*, then the* **sum** *of* I *and* J *is defined as*

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

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

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

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

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

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

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

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

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

Theorem III.2. *Let* I, J *be ideals with vanishing polynomials over* $\mathbb{F}_{2^k}[x_1, \ldots, x_n]$*, then*

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

Let $J_0 = \langle x_1^{2^k} - x_1, \ldots, x_n^{2^k} - x_n \rangle$ 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 III.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 $V(I) = \{a \mid a \in \mathbb{F}_{2^2} \text{ and } f(a) = 0\} = \{1, \alpha\}$. 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 $\quad i \leftarrow i + 1$;
4 $\quad to^i \leftarrow$GB w/ elimination term order$\langle J_{ckt}, J_0, from^{i-1} \rangle$;
5 $\quad new^i \leftarrow$generator of $\langle to^i \rangle + (\langle T^4 - T \rangle : \langle reached \rangle)$;
6 $\quad reached \leftarrow$generator of $\langle reached \rangle \cdot \langle new^i \rangle$;
7 $\quad from^i \leftarrow new^i(S \setminus T)$;
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 III.2. *We apply Algorithm 2 to Ex.I.1 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*

$$f_1 : t_0 - (xs_0s_1 + xs_0 + xs_1 + x + s_0 + s_1 + 1) \qquad f_5 : x^2 - x$$
$$f_2 : t_1 - (xs_0 + x + s_0s_1 + s_0) \qquad f_6 : s_0^2 - s_0, \quad f_7 : s_1^2 - s_1$$
$$f_3 : S - s_0 - s_1 \qquad f_8 : t_0^2 - t_0, \quad f_9 : t_1^2 - t_1$$
$$f_4 : T - t_0 - t_1 \qquad f_{10} : S^4 - S, \; f_{11} : T^4 - T$$

$$J_{ckt} = \langle f_1, f_2, f_3, f_4 \rangle \qquad J_0 = \langle f_5, f_6, \ldots, f_{11} \rangle$$

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

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

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.

A. Research objective: To make the approach scalable

At the time of writing this proposal, the theoretical concepts behind the word-level FSM traversal have been studied and validated. A completed algorithm has been devised and preliminary experiments were conducted for reachability analysis using the SINGULAR computer algebra tool [13]. While the approach is shown to work correctly and completely, it is practically infeasible. The complexity lies in the Gröbner basis computation.

Over Galois fields of the type $\mathbb{F}_{2^k}$, computing the reduced Gröbner basis of the ideal $(J+J_0)$ requires both memory and space $2^{k^{O(n)}}$; where J is an arbitrary polynomial ideal, J_0 is the vanishing ideal, and n is the number of variables in the system (circuit). In this research, I wish to draw inspirations from the work of [2] and [1] to overcome the complexity of GB computations.

In [2], the authors analyze the given circuit topology, and derive a term order to represent the polynomials. They show that this term order renders the set of polynomials itself a Gröbner basis of the ideal. In [1], the authors use a similar concept to derive a *refinement of the abstraction term order* (RATO) to simplify the Gröbner basis computation. However, the above concepts are applied to combinational circuit verification. I will investigate how similar techniques can be explored for the computations that are needed in Algorithm 2 for reachability analysis to make the approach scalable. I believe that it is possible to make the approach scalable, as I have recently demonstrated in a paper [14] which has been accepted for publication at the *Design Automation and Test in Europe Conference*, 2015.

B. Research objective : Application to 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.2(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 $Tr: R_{i-1} \rightarrow R_i$ 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.3(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 α: $\beta = \alpha^t$. 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[15] – 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.3(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.

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 $J = \langle f_1, \ldots, f_s\rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d, R, R', A, A', B, B']$, where $x_1, \ldots, x_d$ 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

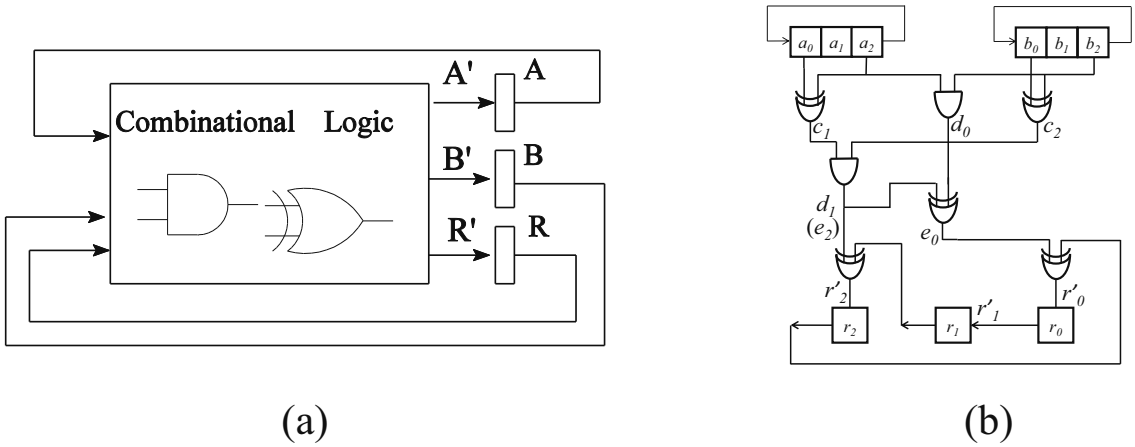

Fig. 3: A 3-bit RH-SMPO and its Moore FSM model

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 3 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 "$\setminus$" 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.

ALGORITHM 3: 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 $\quad i \leftarrow i+1$;
5 $\quad G \leftarrow$GB$(\langle J + J_0 + from_{i-1}(R,A,B)\rangle)$ with ATO;
6 $\quad to_i(R',A',B') \leftarrow G \cap \mathbb{F}_{2^k}[R',A',B',R,A,B]$;
7 $\quad from_i \leftarrow to_i(\{R,A,B\} \setminus \{R',A',B'\})$;
8 **until** $i == k$;
9 **return** $from_k(R_{final})$

Example III.3. *We demonstrate our approach to verify the 3-bit RH-SMPO circuit of Fig.3. 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 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 3. "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 3, we find a polynomial in variables R', A_{init}, B_{init} in $to_1 : R' + (\alpha^2)A_{init}^4 B_{init}^4 + (\alpha^2+\alpha)A_{init}^4 B_{init}^2 + (\alpha^2+\alpha)A_{init}^4 B_{init} + (\alpha^2+\alpha)A_{init}^2 B_{init}^4 + (\alpha^2+\alpha+1)A_{init}^2 B_{init}^2 + (\alpha^2)A_{init}^2 B_{init} + (\alpha^2+\alpha)A_{init} B_{init}^4 + (\alpha^2)A_{init} B_{init}^2$.

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

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 $from_3(R_{final}) = R_{final} + A_{init} \cdot B_{init}$, 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 [14].

IV. Using Algebraic Geometry to Assist in Abstraction refinement

A. Previous work

Traditional methods to analyze sequential circuits require state variable encoding as well as a complete/exact state enumeration with the initial state and transition function. With the size of modern sequential circuits increasing, it is infeasible to apply conventional methods any longer. Abstraction is a technique to minimize the state space representation by combining states with certain similarities. Sometimes it can effectively lower the number of states that require analysis by orders of magnitude, without affecting the properties we need to verify.

At first, abstraction was done manually by designers. In 2000, E. Clarke [16] proposed an automated abstraction: first, initial abstraction is set up by dividing variables to different clusters based on transition conditions; then, an ordinary model checking is applied on this abstraction. If a counterexample is detected, it is further checked to be concrete or not. For a false abstract state or a spurious path , the initial abstraction is refined by further splitting the false abstract state. A heuristic algorithm keeps refining the abstraction until a real counterexample is found or the property is verified. This technique is based on binary decision diagrams (BDDs), which implies a potential risk of memory explosion; meanwhile it still requires an initial abstraction to start with.

In 2004 L. Zhang and M. Hsiao [17] proposed another abstraction method based on CNF-SAT. This paper uses simplest latch abstraction – by removing "irrelevant" latches from the analysis. The key idea is to pick those "irrelevant" latches (variables in *k*-bounded model checking, *k*-BMC) according to SAT/UNSAT analysis from the *k*-BMC. It improves the *k*-BMC procedure[18]: linear temporal logic (LTL) properties to check for can be translated to a (big) CNF formula and fed to a SAT solver. If SAT, a concrete counterexample is found. If UNSAT, the original method increases the bound *k*. In Zhang's paper, the clauses for UNSAT (UNSAT cores[1]) are checked for variables that do not appear. Thus an abstracted model is obtained and checked with the original property. If the abstracted model checking fails, then bound *k* is increased. This method still has disadvantages: it is counterexample-independent, which means it cannot utilize information from invariants.

In 2005, H. Jain et al.[19] improved the abstraction refinement technique of [16], where they use CNF-SAT to perform the refinement instead of using BDDs. The new approach is applied to verify RTL Verilog and was known to be successful.

Abstraction usually means over-approximation, i.e. properties that are true on the abstracted machine are also valid for the original states. Craig's interpolation is a sort of abstraction which may be either over-approximation or under-approximation. In 2003, K. McMillan used Craig's interpolation to improve *k*-BMC [20]. Initially a *k*-length path to failure state *F* is picked and transformed to a CNF formula, then split at first transition into a prefix and a suffix. An interpolation *P* is computed between the prefix and suffix. *P* over-approximates 1-step reachable states, and under-approximates states that cannot reach *F* in *k* steps. If no counterexample is exposed, the path is split again at the second transition; in this way a precise abstraction can be found for all reachable states.

k-BMC with interpolation is a purely incremental model checking approach, and the interpolation procedure relies on UNSAT core analysis. To overcome these weaknesses, a hybrid model checker named as IC3 is developed [5] [6]. IC3 works incrementally to find out inductive subclauses of negations of reached states, meanwhile it is monolithic when computing

[1]UNSAT core is a subset of the original CNF clauses which are also unsatisfiable.

over-approximations to sets of reachable states within $1, 2, \ldots, k$ steps. It is proved to be more efficient than interpolation based model checking although using similar mechanisms.

B. Exploiting UNSAT cores for abstraction refinement

By exploring previous work, we learn that most state-of-art abstraction refinement techniques require information mining from UNSAT proofs of intermediate abstractions. Here we use an abstraction refinement algorithm from [17] to explain that how an UNSAT proof is utilized in such techniques.

ALGORITHM 4: Abstraction refinement using k-BMC

Input: M is the original machine, p is the property to check, k is the number of steps in k-BMC

1 $k =$ InitValue;
2 **if** *k-BMC(M, p, k) is* ***SAT*** **then**
3 **return** *"Found error trace"*
4 **else**
5 Extract UNSAT proof $\mathcal{P}$ of k-BMC;
6 $M' = ABSTRACT(M, \mathcal{P})$;
7 **end**
8 **if** *MODEL-CHECK(M', p) returns* ***PASS*** **then**
9 **return** *"Passing property"*
10 **else**
11 Increase bound k;
12 goto Line 2;
13 **end**

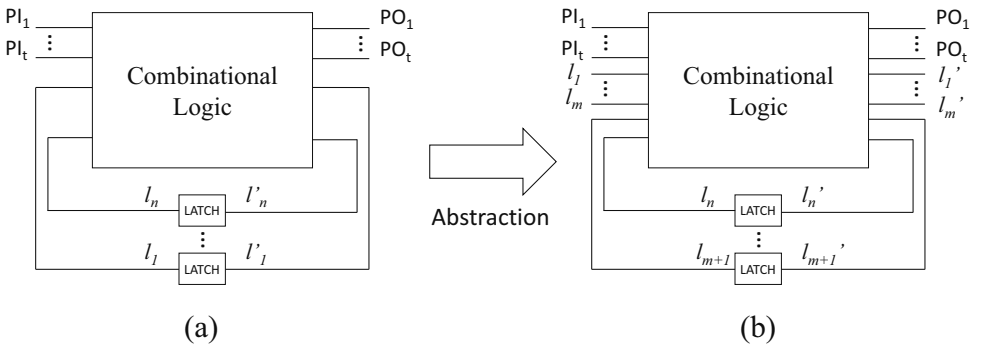

Fig. 4: Abstraction by reducing latches

Assume that we are given a sequential circuit with n latches as shown in Fig.4(a). This circuit can be modeled as a Mealy machine M and the states s can be explicitly encoded by bit-level latch variables $l_1, \ldots, l_n$. Algorithm 4 describes an approach to check if machine M violates property p. This algorithm relies on k-BMC technique, which works on the basis of CNF-SAT solving. The k-BMC represents the initial states I, the transition relation T and property p as CNF formulas.

The first "if-else" branch in Algorithm 4 can be explained as: we check if the conjunction of formulas

$$I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \neg p$$

is SAT or not, where s_i denotes the set of reached states in i-th time-frame. If the result is SAT, then a counterexample is found that violates property p. If the result is $UNSAT$, we cannot assert that p is satisfied for the original machine M because we only unrolled M for a given specific number of time-frames without any fixpoint detected. In this algorithm, we analyze the UNSAT core composed by a set of clauses whose conjunction is $UNSAT$. If there are some latch variables not included in this UNSAT core (denoted by L_{abs}), then we can assert that the evaluations of these variables will not affect the unsatisfiability

of original formula. Therefore, we can ignore them in the abstracted model. In practice, we turn these latches into primary inputs/outputs as shown in Fig.4(b) ($L_{abs} = \{l_1, \ldots, l_m\}$).

The second "if-else" branch means: if we do an ordinary model checking on the abstracted machine M' and find no error trace, we can assert that property p also holds on the original machine M. The reason of this assertion is that the abstracted states represented using abstracted latches cover the original states, which means M' is an over-approximation of M, such that $(M' \Longrightarrow p) \Longrightarrow (M \Longrightarrow p)$. If we find a violation on abstracted machine, then this abstracted model is not a suitable model to check p, so we have to increase the bound k to find a finer abstraction.

It is clear that UNSAT cores play an important role in abstraction refinement approaches. In [17] the UNSAT core is extracted using a conventional CNF-SAT solver, which will encounter the "bit-blasting" problem when the size of datapath (number of latches in Fig.4) is very large. **Here we propose a totally new method based on Gröbner basis computation to extract UNSAT cores, and we believe it may become an efficient method according to the following observation based on our experience:**

While computing GB over finite fields is exponential in the number of variables, GB computation is observed to be more efficient for UNSAT problems. The reason is discussed as follows:

Theorem IV.1. Weak Nullstellensatz: *Given ideal $J \subset \mathbb{F}[x_1, \ldots, x_n]$, its variety over algebraic closure of field $\mathbb{F}$ is empty if and only if its reduced Gröbner basis contains only one generator "1".*

$$\mathbf{V}_{\overline{\mathbb{F}}}(J) = \emptyset \Longleftrightarrow reducedGB(J) = \{1\}$$

It is well known that using Buchberger's algorithm and its variations to compute a GB has a very high time complexity and is usually not practicable. One reason is that the size of GB may explode even if the term ordering is carefully chosen. However if the reduced GB is 1, which means every term in the original polynomials will be canceled, the degree of remainders when computing GB with Buchberger's algorithm will be limited. Thus the number of polynomials in non-reduced GB is much smaller than usual. Instead of applying polynomial calculus to SAT solving, it may be more efficient to try it for UNSAT problems.

C. A demonstration of our conjecture

One important research topic about UNSAT problems is to identify UNSAT cores efficiently. An UNSAT core in a CNF formula denotes a subset of clauses which is still unsatisfiable. Here we re-define this concept in algebraic geometry:

Definition IV.1. *Assume a set of polynomials F and its subset $F_s \subset F$. If $\mathbf{V}(\langle F \rangle) = \mathbf{V}(\langle F_s \rangle) = \emptyset$, we call F_s an* **UNSAT core** *of F. Additionally, if F_s contains no other UNSAT core, we call it a* **minimal** *UNSAT core.*

We conjecture that based on observation of Buchberger's algorithm's execution, an UNSAT core can be identified.

Conjecture IV.1. *Buchberger's algorithm picks pairs of polynomials from a given set, computes their S-poly, then reduces this S-poly with the given set of polynomials. If the remainder is non-zero, it is added to the set of polynomials. By tracking S-poly computations and multivariate divisions that lead to remainder 1, we can obtain an UNSAT core. Moreover, we can identify a minimal UNSAT core with one-time execution of Buchberger's algorithm.*

Example IV.1. *A SAT problem is described with 8 CNF clauses:*

$$c_1 : \bar{a} \vee \bar{b} \qquad c_5 : x \vee y$$
$$c_2 : a \vee \bar{b} \qquad c_6 : y \vee z$$
$$c_3 : \bar{a} \vee b \qquad c_7 : b \vee \neg y$$
$$c_4 : a \vee b \qquad c_8 : a \vee x \vee \neg z$$

Using Boolean to polynomial mappings given in Table I, we can transform them to a set of polynomials F over ring $\mathbb{F}_2[a,b,x,y,z]$:

$$f_1 : ab \qquad f_5 : xy + y + x + 1$$
$$f_2 : ab + a \qquad f_6 : yz + y + z + 1$$
$$f_3 : ab + b \qquad f_7 : by + y$$
$$f_4 : ab + a + b + 1 \qquad f_8 : axz + az + xz + z$$

We compute its GB using Buchberger's algorithm with lexicographic term ordering $a > b > x > y > z$. Since this problem is UNSAT, we will stop when "1" is added to GB.

1) *First we compute $Spoly(f_1, f_2) \xrightarrow{F}_+ r_1$, remainder r_1 equals to a;*
2) *Update $F = F \cup r_1$;*
3) *Next we compute $Spoly(f_1, f_3) \xrightarrow{F}_+ r_2$, remainder r_2 equals to b;*
4) *Update $F = F \cup r_2$;*
5) *We can use a directed acyclic graph (DAG) to represent the process to get r_1, r_2, as Fig.5(a) shows;*
6) *Then we compute $Spoly(f_1, f_4) = s_3 = a + b + 1$, obviously $a + b + 1$ can be reduced (multivariate divided) by r_1 , the intermediate remainder $r_3 = b + 1$. It can be immediately divided by r_2, and the remainder is "1", we terminate the Buchberger's algorithm;*
7) *We draw a DAG depicting the process through which we obtain remainder "1" as shown in Fig.5(b). From leaf "1" we backtrace the graph to roots f_1, f_2, f_3, f_4. They constitute an UNSAT core for this problem as these polynomials are the "causes" of unsatisfiability of original set of clauses.*

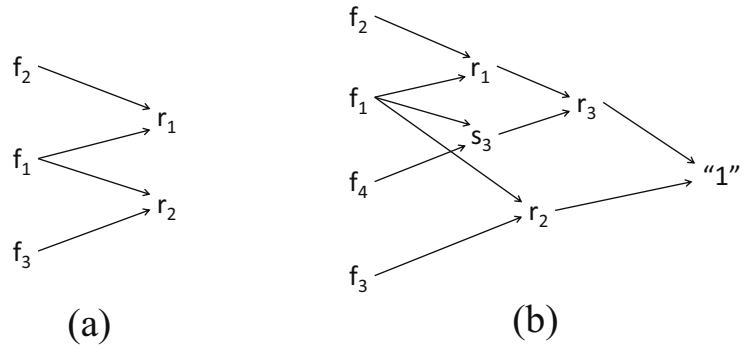

Fig. 5: DAG representing Spoly computations and multivariate divisions

We conclude our approach as a conjecture algorithm (Algorithm 5).

ALGORITHM 5: Extract UNSAT core using a variation of Buchberger's algorithm

Input: A set of polynomials $F = \{f_1, f_2, \ldots, f_s\}$
Output: An UNSAT core $\{f_{m_1}, f_{m_2}, \ldots, f_{m_t}\}$

repeat
  Pick a pair $f_i, f_j \in F$ that has never been computed S-poly;
  if $Spoly(f_i, f_j) \xrightarrow{F}_+ r_l \neq 0$ **then**
    $F = F \cup r_l$;
    Create a DAG G_l with f_i, f_j as roots, r_l as leaf, recording the S-poly, all intermediate remainders and $f_k \in F$ that cancel monomial terms in the S-poly;
  end
until $r_l == 1$;
Backward traverse the DAG for remainder "1", replace r_l with corresponding DAG G_l;
return *All roots*

D. Research objective

i) To prove that the algorithm is concrete and complete: it can always abstract an UNSAT core;

ii) To design an algorithm to find a minimal UNSAT core;

iii) Run the algorithm at word level, to find UNSAT proofs for word-level variables in $\mathbb{F}_{2^k}$, which can help word-level abstraction refinement procedure.

V. Proposed Objectives

- Explore an implementation of algebraic geometry based reachability analysis for sequential circuits verification;
- Currently our approach is implemented within the SINGULAR tool, which is not efficient enough to execute our proposed algorithms, because its data-structures are not efficient for circuit verification problems. We plan to design a standalone CAD tool implemented in C++, which specifically aims to solve word-level sequential verification problems. We will borrow techniques from [1], [2] and [21], and tailor them to sequential circuit analysis, to lower the computational complexity;
- Fine-tune the tool for sequential Galois field arithmetic circuits verification, because arithmetic circuit verification greatly benefits from the abstraction of word-level operands in the datapath;
- Explore a new abstraction-refinement paradigm, based on information from UNSAT cores, which is also attained using a Gröbner basis approach.

VI. Proposed Timetable

- Current status: Experiments have been performed to run implicit state enumeration on sequential circuits benchmarks such as ISCAS'89 circuits. Current problem is the algorithm spends too much time on multivariate division procedure;
- Spring 2015: Implement the tool which can efficiently do multivariate division and test the performance of our tool based on T. Pruss et al's [1] approach;
- Summer 2015: Integrate the refined multivariate division routine into our verification tool, test its performance on circuits with various sizes;
- Fall 2015: Evaluate data and write the dissertation.

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] Jinpeng Lv, Priyank Kalla, and Florian Enescu, "Efficient gröbner basis reductions for formal verification of galois field arithmetic circuits", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, pp. 1409–1420, 2013.

[3] Alexey Lvov, Luis Alfonso Lastras-Montano, Viresh Paruthi, Robert Shadowen, and Ali El-Zein, "Formal verification of error correcting circuits using computational algebraic geometry", *in Formal Methods in Computer-Aided Design (FMCAD), 2012*, pp. 141–148. IEEE, 2012.

[4] Michael Brickenstein and Alexander Dreyer, "Polybori: A framework for gröbner-basis computations with boolean polynomials", *Journal of Symbolic Computation*, vol. 44, pp. 1326–1345, 2009.

[5] Aaron R Bradley, "Sat-based model checking without unrolling", *in Verification, Model Checking, and Abstract Interpretation*, pp. 70–87. Springer, 2011.

[6] Aaron R Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang, "An incremental approach to model checking progress properties", *in Formal Methods in Computer-Aided Design (FMCAD), 2011*, pp. 144–153. IEEE, 2011.

[7] George S Avrunin, "Symbolic model checking using algebraic geometry", *in Computer Aided Verification*, pp. 26–37. Springer, 1996.

[8] Quocnam Tran and Moshe Y Vardi, "Groebner bases computation in boolean rings for symbolic model checking", *in Proceedings of IASTED 2007 Conference on Modeling and Simulation*, 2007.

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

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

[11] 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.

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

[13] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, "SINGULAR 3-1-6 — A computer algebra system for polynomial computations", http://www.singular.uni-kl.de, 2012.

[14] 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.

[15] 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.

[16] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith, "Counterexample-guided abstraction refinement", *in Computer aided verification*, pp. 154–169. Springer, 2000.

[17] Liang Zhang, *Design Verification for Sequential Systems at Various Abstraction Levels*, PhD thesis, Citeseer, 2005.

[18] Armin Biere, Alcssandro Cimatti, Edmund Clarke, and Yunshan Zhu, "Symbolic model checking without bdds", *in Tools and Algorithms for the Construction of Analysis of Systems: 5th International Conference, TACAS'99, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, number 1597, p. 193. Springer, 1999.

[19] Himanshu Jain, Daniel Kroening, Natasha Sharygina, and Edmund Clarke, "Word level predicate abstraction and refinement for verifying rtl verilog", *in Proceedings of the 42nd annual Design Automation Conference*, pp. 445–450. ACM, 2005.

[20] Kenneth L McMillan, "Interpolation and sat-based model checking", *in Computer Aided Verification*, pp. 1–13. Springer, 2003.

[21] Christian Eder and John Edward Perry, "Signature-based algorithms to compute gröbner bases", *in Proceedings of the 36th international symposium on Symbolic and algebraic computation*, pp. 99–106. ACM, 2011.