Formal Verification of Sequential Galois Field Circuits using Word-Level FSM Traversal via Algebraic Geometry

Xiaojun Sun*, Priyank Kalla*, Florian Enescu†
*Electrical & Computer Engineering, University of Utah
†Mathematics & Statistics, Georgia State University

***Abstract*—Sequential circuits that implement Galois field arithmetic over $\mathbb{F}_{2^k}$ are prevalent in cryptography and error control coding, where the datapath size k is very large. Formal verification of sequential arithmetic circuits with large datapath size is beyond the capabilities of contemporary verification techniques. To address this problem, this paper introduces a new concept of implicit state enumeration of finite state machines (FSMs) performed at the word-level. Using algebraic geometry, we show that the state-space of a sequential circuit can be encoded, canonically, as solutions to a multivariate word-level polynomial $\mathcal{F}(A)$ in $\mathbb{F}_{2^k}$, where $A = a_0, \dots, a_{k-1}$ represents the k-bit state-register (word). Subsequently, the Groebner basis algorithm, along with elimination ideals and quotients of ideals can be employed for FSM traversal. We apply our approach to verify sequential Galois field multipliers with large datapath sizes, where contemporary techniques prove to be infeasible.**

I. Introduction

Verification for sequential circuits has been extensively discussed for decade. Binary Decision Diagram (BDD) is the first and most widely used canonical technique among those for sequential verification [1]. It is easy to manipulate any Boolean expressions with BDD and use them to enumerate implicit states. However, an issue that BDDs have to face is the size explosion when Boolean variables increase significantly. Several modifications have been made to improve this problem by optimizing original BDD [2] [3], or by reducing Boolean variables from circuits[4] [5]. Some variants of BDD representation have also been developed such as Linear Taylor Expansion Diagram (LTED) [6].

Another sort of approaches are based on satisfiability theory (SAT). SAT-solver is applied to state space traversal and work through eliminations on Boolean constrains [7]. By exploiting SAT-based algorithm, direct reachability induction is proposed [8]. Still SAT-based approaches need to struggle with size explosion problem.

II. Related Works

G. Avrunin [9] introduced algebraic geometry into formal verification. In his paper, the corresponding relation between varieties of ideals and circuit variables is discussed, and concepts such as intersect of varieties, ideal and its generators and algebraic closure are explored. Yet, a self-contained system of algebraic geometry representation theory has not been well-defined.

T. Pruss, el(which one should I cite?) developed a word-level abstraction method based on Gröbner basis theory...

J. Lv, el (should I cite this part) gave out a F4-style reduction algorithm which can help speed up Gröbner basis calculation...

III. Essential Theory

A. Polynomial Representation

A typical sequential circuit is composed as figure 1. The combinational logic has primary input x, state inputs $\{s_0, s_1, \dots, s_{k-1}\}$, primary output z and state outputs $t_0, t_1, \dots, t_{k-1}$. State outputs are latched into registers, which will be fed back to state inputs in next clock cycle.

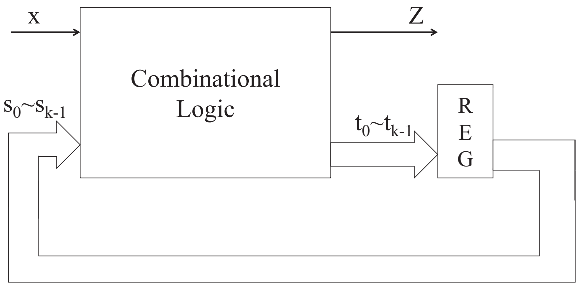

Fig. 1: A typical sequential circuit block view

A transition function Δ is a Boolean function about state outputs and all inputs, i.e.

$$t_i = \Delta_i(x, s_0, s_1, \dots, s_{k-1})$$

This constrain could also be written in the form of equation $polynomial = 0$:

$$t_i - \Delta_i(x, s_0, s_1, \dots, s_{k-1}) = 0$$

Solution to this equation is limited within $\{0,1\}$, which is set of all elements in Galois field $\mathbb{F}_2$. Furthermore, there exists a one-to-one corresponding relation from Boolean space to Galois field on higher dimension: $\mathbb{B}^k \rightarrow \mathbb{F}_{2^k}$, which means it is possible to find a set of elements in $\mathbb{F}_{2^k}$ representing all Boolean values every bit of a bit vector can take. In Galois field, a as I shows. A word-level representation on state

$a_3a_2a_1a_0$	Polynomial	$a_3a_2a_1a_0$	Polynomial
0000	0	1000	α^3
0001	1	1001	α^3+1
0010	α	1010	$\alpha^3+\alpha$
0011	$\alpha+1$	1011	$\alpha^3+\alpha+1$
0100	α^2	1100	$\alpha^3+\alpha^2$
0101	α^2+1	1101	$\alpha^3+\alpha^2+1$
0110	$\alpha^2+\alpha$	1110	$\alpha^3+\alpha^2+\alpha$
0111	$\alpha^2+\alpha+1$	1111	$\alpha^3+\alpha^2+\alpha+1$

TABLE I: Bit-vector, Exponential and Polynomial representation of elements in $\mathbb{F}_{2^4} = \mathbb{F}_2[x] \pmod{x^4+x^3+1}$

inputs/outputs can be attained through above correspondence. If the set of state inputs $\{s_0, s_1, \dots, s_{k-1}\}$ is denoted by S taking values from $\mathbb{F}_{2^k}$, and set of outputs $\{t_0, t_1, \dots, t_{k-1}\}$ is denoted by T, a word-level transition function can be written as

$$T - \Delta(x, S) = 0$$

If $T - \Delta(x, S)$ is taken as a polynomial f, then $f \in \mathbb{F}_{2^k}[x]$ *mod* $P(\alpha)$.

B. Gröbner Basis Theory

Gröbner basis is used to calculate desired results out of a set of polynomials. If there exists multiple polynomials $\{f_1, f_2, \dots, f_r\}$ representing constrains on T, x and S, then all polynomials should simultaneously equal to 0.

$$\begin{cases} f_1 & = 0 \\ f_2 & = 0 \\ \dots & \\ f_r & = 0 \end{cases}$$

Solution to these equations is set of values of (T, x, S), which is called **Variety** of **ideal** $I = \langle f_1, f_2, \dots, f_r \rangle$.

Calculate Reduced Gröbner Basis (RGB) from ideal I with **elimination term order**, there exists a polynomial contains only T, if the values of states input S and primary input x are given in the form of polynomials and included by I, the values of T can be computed.

Otherwise, if RGB calculation is manipulated under **abstraction term order**, there exists a polynomial in the form of $T - \mathcal{F}(x, S)$ representing transition function. In next clock cycle, assign new state input S' with T, the next state can be computed again, result is polynomial $T' - \mathcal{F}(x', S')$.

C. Normal Basis Representation

Let β be an element in the Galois field F_{2^n} constructed by primitive element α and minimal polynomial $P(\alpha)$. Then a basis in the form $\{\beta, \beta^2, \beta^4, \beta^8, ..., \beta^{2^{n-1}}\}$ is a *Normal Basis*; here β is called *Normal Element*.

For arithmetic operations in Galois fields, squaring and multiplication (with modulus) can be greatly simplified if normal basis representation is adopted to represent operands.

Example III.1. *Element squaring: In F_{2^n}, all coefficients which can be divided by 2 are reduced, so following equality holds for all field elements a and b:*

$$(a+b)^2 = a^2 + b^2$$

Apply this rule on element squaring of standard/polynomial basis:

$$\begin{aligned} &(b_0\beta + b_1\beta^2 + b_2\beta^4 + \cdots + b_{n-1}\beta^{2^{n-1}})^2 \\ &= b_0^2\beta^2 + b_1^2\beta^4 + b_2^2\beta^8 + \cdots + b_{n-1}^2\beta^{2^n} \\ &= b_{n-1}^2\beta + b_0\beta^2 + b_1\beta^4 + \cdots + b_{n-2}\beta^{2^{n-1}} \end{aligned}$$

using Fermat's little theorem $\beta^{2^n} = \beta$. This shows that squaring of field elements represented by normal bases can be easily completed by operating a simple right-cyclic rotation.

Example III.2. *Normal basis multiplication: There are 2 binary vectors representing operands of multiplication in normal basis:*

$$A = (a_0, a_1, \dots, a_{n-1}),\ B = (b_0, b_1, \dots, b_{n-1})$$

the product is also written in normal basis representation:

$$C = A * B = (c_0, c_1, \dots, c_{n-1})$$

Describe calculation procedure for the most significant digit of product with a function:

$$c_{n-1} = f(a_0, a_1, \dots, a_{n-1}; b_0, b_1, \dots, b_{n-1})$$

Square both side: $C^2 = A^2 * B^2$, i.e. the second significant digit

$$c_{n-2} = f(a_{n-1}, a_0, a_1, \dots, a_{n-2}; b_{n-1}, b_0, b_1, \dots, b_{n-2})$$

By this method it is easy to get all digits of product C.

IV. Algebraic Geometry Approach

A. State Space Traversal

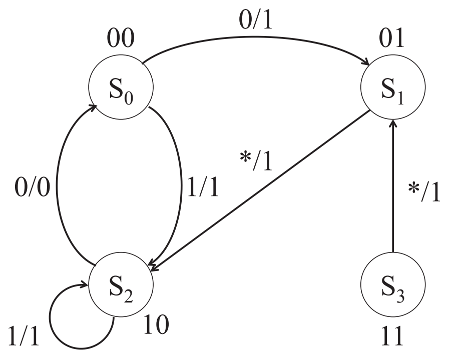

Fig. 2: State transition graph of sample FSM

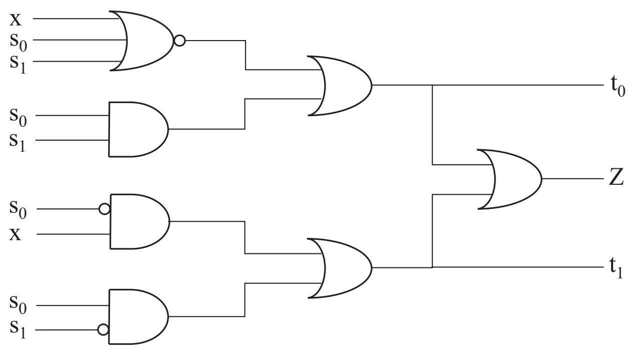

Fig. 3: Gate-level circuit of combinational logic block of sample FSM

The first example is a mealy finite state machine (FSM). Gate-level circuits of combinational logic is figure 3, x is primary input, $\{s_0, s_1\}$ are state inputs, Z is primary output and $\{t_0, t_1\}$ are state outputs. Word-level state input is $S = s_0 + s_1 \cdot \alpha$, output is $T = t_0 + t_1 \cdot \alpha$. In Galois field $\mathbb{F}_{2^2}$, bit-level variables x, s_0, s_1, Z, t_0, t_1 can take values from $\{0, 1\}$, while word-level variables S, T may take values from $\{0, 1, \alpha, 1+\alpha\}$. State transition graph (STG) showed in figure 2 uses 2-bit Boolean vector to represent 4 states $\{S_0, S_1, S_2, S_3\}$, which could be converted to elements in $\mathbb{F}_{2^2}$ similarly as table I shows.

One important technique to check sequential equivalence is **state space traversal**. In this example, Breath-First-Search (BFS) is employed for traversal, which can be described with following algorithm:

ALGORITHM 1: Breadth-first Traversal Algorithm

```
Input: Transition functions $\Delta$, initial state $S^0$
$from^0 = reached = S^0$;
repeat
    $i \leftarrow i + 1$;
    $to^i \leftarrow \mathrm{Img}(\Delta, \overline{from^{i-1}})$;
    $new^i \leftarrow to^i \cap \overline{reached}$;
    $reached \leftarrow reached \cup new^i$;
    $from^i \leftarrow new^i$;
until $new^i == 0$;
return reached
```

Our approach is implementing state space traversal by refined BFS algorithm combined with polynomial representation and Gröbner basis theory. In practice, this example will show how to apply concepts and techniques from algebraic geometry on each line of above BFS algorithm.

B. States and Varieties of Ideal

Theorem IV.1. *State variables S, T and sets of states such as $from^i, to^i$ can always be represented as varieties of ideals.*

For example in Line 1 of BFS algorithm, assume initial state is S_0 in figure 2, then corresponding polynomial $f = \mathcal{F}(S^0) = S^0 - 0$. Consider an ideal I with only one generator f, its variety $V(I) = \{\gamma \mid \gamma \in \mathbb{F}_{2^2}, \gamma = 0\}$, which equals to $\{0\}$, the only valid value S^0 can take.

If an ideal contains multiple polynomial specifications, it is necessary to compute Gröbner basis with elimination term order to get one polynomial only on desired variable. In first iteration of Line 4, to^i has multiple specifications, some of them are transition functions on bit-level:

$$\begin{aligned} f_1: &\quad t_0 - (\overline{x} \text{ and } \overline{s_0} \text{ and } \overline{s_1}) \text{ or } (s_0 \text{ and } s_1) \\ f_2: &\quad t_1 - (\overline{s_0} \text{ and } x) \text{ or } (s_0 \text{ and } \overline{s_1}) \end{aligned}$$

And some are bits-to-word definitions fitting polynomial representation of elements in $\mathbb{F}_{2^2}$:

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

And an polynomial about initial state as mentioned above:

$$f_5: \; S$$

And the rests are vanishing polynomials for every variable, bit-level and word level: $f_6: x^2 - x$; $f_7: t_0^2 - t_0$; $f_8: t_1^2 - t_1$; $f_9: S^4 - S$; $f_{10}: s_0^2 - s_0$; $f_{11}: s_1^2 - s_1$; $f_{12}: T^4 - T$

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

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

Transition equations here contains some Boolean operators, they can be re-written in terms of operations in Galois fields. In $\mathbb{F}_2$, let $TRUE = 1, FALSE = 0$, for either element a in this field, considering $0 + 0 = 1 + 1 \equiv 0 \ (mod\ 2)$ and $0 + 1 = 1 + 0 \equiv 1 \ (mod\ 2)$, the inverse of a is: $\overline{a} = 1 + a$. Similarly all Boolean operators can be converted within $\mathbb{F}_2$, table II gives part of them and their corresponding operations in $\mathbb{F}_2$: Also note that there is no specification on initial primary input x, in Gröbner basis approach this means x is smoothed by reversely using Shannon's expansion. Using elimination term order: *intermediate bit-level signals* $>$ *bit-level primary inputs/outputs* $> S > T$, compute Gröbner basis for ideal $J = \langle f_1, f_2, \ldots, f_{12} \rangle$, the result will include one polynomial f_T contains only variable T. In this example, $f_T = T^2 + (\alpha+1)T + \alpha$. This polynomial equals to $T^2 - (\alpha + 1)T + \alpha$ since coefficients of polynomial representation in $\mathbb{F}_{2^2}$ are limited within $\mathbb{F}_2$, where $x \equiv -x \ (mod\ 2)$ for any element x in the field. Solution to $f_T = 0$ is $T = 1$ *or* $T = \alpha$, which shows that next state the machine just reached is $\{S_1(01), S_2(10)\}$.

Continue to run this traversal algorithm, it will terminates with final reachable states $f_T = T^3 + (\alpha+1)T^2 + \alpha T$, its solution is $T = 0$ *or* $T = 1$ *or* $T = \alpha$, which shows within total 4 states, state S_3 is unreachable from initial state S_0.

C. Application of Algebraic Geometry

There are some difficulties with polynomial representation when executing Line 5 and Line 6, it is necessary to explore how **Union**, **Intersection** and **Complement Set** works in Galois field $\mathbb{F}_{2^k}$. Since state set variables $from^i, to^i, reached$ all have single polynomial, consider an ideal I with single generator $I = \langle f \rangle$. Example: assume $I = \langle f \rangle = \langle T^2 + (1+\alpha) \cdot T + \alpha \rangle$, its variety $V(I) = \{a \mid a \in \mathbb{F}_{2^2} \text{ and } f(a) = 0\} = \{1, \alpha\}$.

So it is reasonable to specify: union, intersection and complement set mentioned in this paper are all functions on **varieties**. To better discuss this problem, introduce following concepts from algebraic geometry:

Definition IV.1. *(***Sum of Ideals***) If I and J are ideals in $k[x_1, \ldots, x_n]$, then the* **sum** *of I and J, denoted by $I + J$, is the set*

$$I + J = \{f + g \mid f \in I \text{ and } g \in J\}. \qquad (1)$$

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

Definition IV.2. *(***Product of Ideals***) If I and J are ideals in $k[x_1, \ldots, x_n]$, then the* **product** *of I and J, denoted by $I \cdot J$, is defined to be the ideal generated by all polynomials $f \cdot g$ where $f \in I$ and $g \in J$. Furthermore, let $I = \langle f_1, \ldots, f_r \rangle$ and $J = \langle g_1, \ldots, g_s \rangle$, then*

$$I \cdot J = \langle f_i g_j \mid 1 \le i \le r, 1 \le j \le s \rangle. \qquad (2)$$

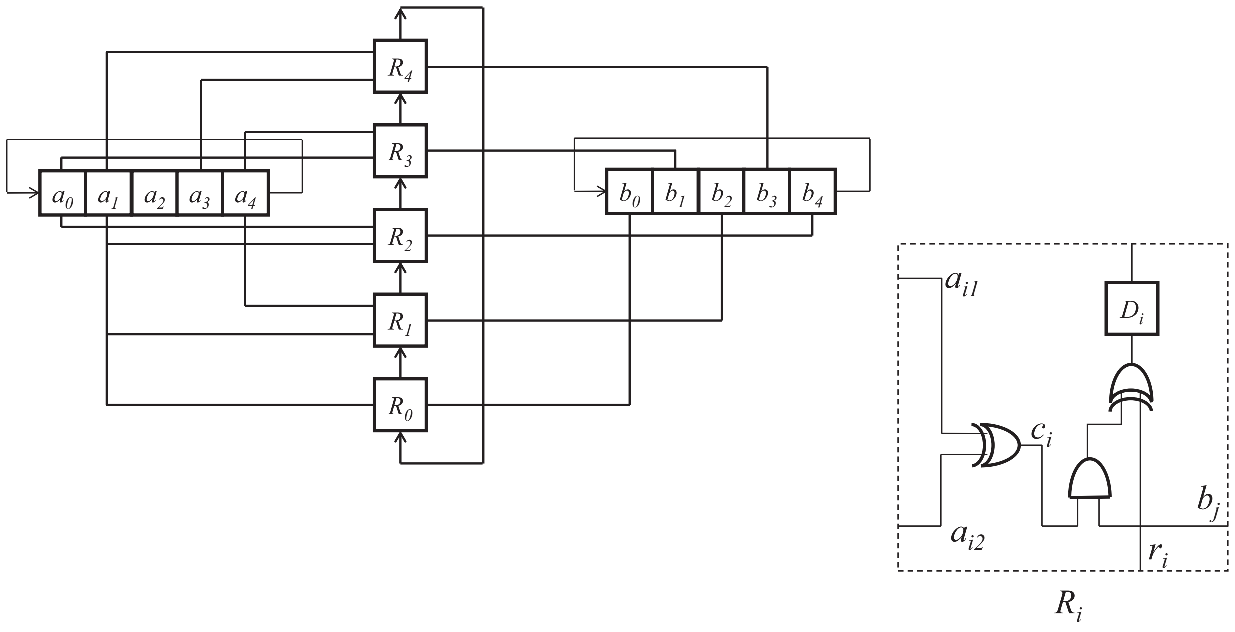

Fig. 4: 5-bit SMPO

Definition IV.3. *(***Quotient of Ideals***) If I and J are ideals in $k[x_1,\ldots,x_n]$, then $I:J$ is the set*

$$\{f \in k[x_1,\ldots,x_n] \mid f\cdot g \in I, \forall g \in J\} \quad (3)$$

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

These concepts can lead the way to solution by adopting following theorems:

Theorem IV.2. *If I and J are ideals in $k[x_1,\ldots,x_n]$, then $\mathbf{V}(I+J)=\mathbf{V}(I)\bigcap\mathbf{V}(J)$ and $\mathbf{V}(I\cdot J)=\mathbf{V}(I)\bigcup\mathbf{V}(J)$.*

Theorem IV.3. *If I,J are ideals with only one generator, then $\mathbf{V}(I:J)=\mathbf{V}(I)-\mathbf{V}(J)$.*

Proof to these theorems are referred to (that yellow book?) and (my write-up?).

For varieties intersection $\{1\}\bigcap\{1,\alpha\}$, calculate ideal sum $\langle T+1, T^2+(1+\alpha)\cdot T+\alpha\rangle = \langle T+1\rangle$, its variety is $\{1\}$; for varieties union $\{1,\alpha\}\bigcup\{1+\alpha\}$, calculate ideal product $\langle (T+1+\alpha)\cdot(T^2+(1+\alpha)\cdot T+\alpha)\rangle = \langle T^3+1\rangle$, its variety is $\{1,\alpha,1+\alpha\}$; 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 $\{0,1+\alpha\}$.

From discussion above, the BFS traversal is completely implemented in Galois field. If the initial input is S_3: $S+1+\alpha$, the final return value will be set of reachable states: T^4+T, i.e. the universal set, $\{S_0,S_1,S_2,S_3\}$. Modified algorithm could be written in following interpretation:

ALGORITHM 2: Algebraic Geometry based Traversal Algorithm

```
Input: Input-output circuit characteristic polynomial ideal $I_{ckt}$,
       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$< I_{ckt}, from^{i-1} >$;
5     $new^i \leftarrow$generator of $< to^i > + (< T^4 - T >:< reached >)$;
6     $reached \leftarrow$generator of $< reached > \cdot < new^i >$;
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 polynomial about S or T

V. Functional Verification

A. Sequential Galois Arithmetic Multiplier

Our experiment is to verify the function of a Galois field multiplier after certain clock cycles. The following design is Sequential Multiplier with Parallel Output (SMPO), a Normal Basis multiplier based on Massey-Omura algorithm. Inputs and outputs are all 5-bit word taking value from $\mathbb{F}_{2^5}$. After loading operands A and B, setting all output latches to 0 and running for 5 iterations, the output should be $R = A\cdot B$ (mod x^5+x^2+1).

Similarly, build elimination ideal for all gates/operations and induce initial states of latches. However, instead of eliminating all variables to one, this example adopts abstraction term order and keeps the polynomial contains function between output and inputs. Here the lex ordering is $\{all\ bit\text{-}level\ variables\} > R > \{A,B\}$, and objective polynomial is $R+\mathcal{F}(A,B)$.

Apply this approach to calculate image in BFS traversal, but modify algorithm 1 to make it adapt 5 steps reachable states enumeration. The result is $R+AB$, which validates the function of this circuit.

B. Gröbner basis based Approach

For 5-bit normal basis multiplication, the i-th digit of product can be written as

$$\begin{aligned}R_i = b_i a_{i+1} + b_{i+1}(a_i + a_{i+3}) + b_{i+2}(a_{i+3} + a_{i+4})\\ + b_{i+3}(a_{i+1} + a_{i+2}) + b_{i+4}(a_{i+2} + a_{i+4}),\ 0 \le i \le 4\end{aligned}$$

It is possible to calculate every conjunctive term in the product simultaneously within one clock cycle, on distinct bits. Use cyclic similarity of above function, following executions are operated:

Example V.1. *Sequential Multiplier Protocol:*

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

In BFS algorithm, the **Image** function is implemented by constructing an elimination ideal then compute its Gröbner basis.

Example V.2. *For 5-bit SMPO, the ideal consists of (for the first clock cycle):*

(a) **Gate descriptions:** $a_1 + a_4 + c_1, a_1 + a_0 + c_2, a_0 + a_4 + c_3, a_1 + a_3 + c_4, a_1 b_0 + r_4 + R_0, c_1 b_2 + r_0 + R_1, c_2 b_4 + r_1 + R_2, c_3 b_1 + r_2 + R_3, c_4 b_3 + r_3 + R_4$;

(b) **Word-level variables:** $A + a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18}, B + b_0\alpha^5 + b_1\alpha^{10} + b_2\alpha^{20} + b_3\alpha^9 + b_4\alpha^{18}, r + r_0\alpha^5 + r_1\alpha^{10} + r_2\alpha^{20} + r_3\alpha^9 + r_4\alpha^{18}, R + R_0\alpha^5 + R_1\alpha^{10} + R_2\alpha^{20} + R_3\alpha^9 + R_4\alpha^{18}$;

(c) **Vanishing polynomials:** $a_0^2 + a_0, a_1^2 + a_1, a_2^2 + a_2, a_3^2 + a_3, a_4^2 + a_4, b_0^2 + b_0, b_1^2 + b_1, b_2^2 + b_2, b_3^2 + b_3, b_4^2 + b_4, r_0^2 + r_0, r_1^2 + r_1, r_2^2 + r_2, r_3^2 + r_3, r_4^2 + r_4, R_0^2 + R_0, R_1^2 + R_1, R_2^2 + R_2, R_3^2 + R_3, R_4^2 + R_4, c_1^2 + c_1, c_2^2 + c_2, c_3^2 + c_3, c_4^2 + c_4, A^{32} + A, B^{32} + B, r^{32} + r, R^{32} + R$;

(d) **Feedback input:** r_{in}.

Polynomial r_{in} equals to $from^{i-1}$ in Line 4, BFS algorithm. Under abstraction term ordering, polynomial to^i is assigned with a polynomial in result Gröbner basis which has the form $R+\mathcal{F}(A,B)$. Since all outputs are connected to feedback inputs in SMPO, Line 5 and 6 will be neglected. Line 7 is finished by replace current output R with previous output r. Initially $r_{in} = 0$.

In next clock cycle, $r_{in} = r + \mathcal{F}(A,B)$ updated from latest result; simultaneously operands A and B have been cyclically shifted, so gate descriptions in (a) should be modified accordingly. The loop is operated for 5 times, result from the last step's Gröbner basis should be polynomial $R+AB$.

This experiment can be repeated on n-bits SMPO after running for n clock cycles.

VI. Fast Gröbner basis computation

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 VI.1. *r will only contain primary inputs (bit-level and word-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. $\square$

Example VI.1. *The elimination ideal for 5-bit SMPO (Ex.V.2) could be rewritten under RATO:*

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

Thus the candidate pair is (f_w, f_g), $f_w = R_0 + r_4 + b_0 \cdot a_1$, $f_g = R_0\alpha^5 + R_1\alpha^{10} + R_2\alpha^{20} + R_3\alpha^9 + R_4\alpha^{18} + R$. Result after reduction is:

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

The remainder satisfies Lemma VI.1.

B. Bit-level Variable Substitution (BLVS)

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

Example VI.2. Objective*: Abstract polynomial $a_i + \mathcal{G}_i(A)$ from $f_0 : a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18} + A$.*

First, compute $f_0^2 : a_0\alpha^{10} + a_1\alpha^{20} + a_2\alpha^9 + a_3\alpha^{18} + a_4\alpha^5 + A^2$. Apparently variable a_0 can be eliminated by operation

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

Recursively eliminate a_1 from f_1, a_2 from f_2, etc. The final polynomial f_4 has the form $a_4 + \mathcal{G}_4(A) = a_4 + (\alpha^4+\alpha^3+\alpha)A^{16} + (\alpha^3+\alpha^2)A^8 + (\alpha^4+1)A^4 + (\alpha^2+1)A^2 + (\alpha+1)A$. Recursively substitute $\mathcal{G}_4(A)$ back to f_3, etc. The result is a set of polynomials

$$\{g_i \mid g_i : a_i + \mathcal{G}_i(A)\}$$

In this approach it is easy to get word-level variable representation for each bit-level primary inputs. By substitution, a new polynomial in the form $R+\mathcal{F}(A,B)$ could be attained.

Discussion (a) This approach won't conduct to a result reducing to 0, because *Spoly*'s remainder contains information from the whole system, the substitution will result something new, i.e. an abstraction of the system; (b) The Gaussian-like approach is a pre-processing of a variable, and could be immediately reused to other words with the same length; (c) For n-bits SMPO, the fast GB is very efficient: for each cycle, first compute a *Spoly* in $O(1)$, then do multi-division by $O(n)$ polynomials (unknown complexity, maybe lower if using F4-style reduction?), the substitution cost at most $O(n^3)$ time (assuming $O(1)$ for each term). In total repeat for n cycles. The complexity should be much lower compared to Buchberger's algorithm.

VII. Experiment Results

Our experiment on different size of SMPO designs is performed with Singular symbolic algebra computation system. The SMPO designs are given as gate-level netlists with registers, then translated to polynomials to compose elimination ideal in Singular for Gröbner basis calculation. The experiment is conducted on desktop with 3.5GHz Intel Core^{TM} i7 Quad-core CPU, 16 GB RAM and running 64-bit Linux OS.

A. Computing Gröbner basis in Singular

The Singular tool can read in scripts written in its own format similar to ANSI-C. For SMPO experiment, the main loop of our script file performs the same function as algorithm 2 describes, while Gröbner basis computation in main loop can be divided into 4 different function parts:

(i) Pre-process:

This step is executed only once before main loop starts. The function of pre-process is to compute following system of equations for bit-level inputs $a_0 \sim a_{k-1}$:

$$\begin{cases} a_0 &= f_0(A) \\ a_1 &= f_1(A) \\ \vdots & \\ a_{k-1} &= f_{k-1}(A) \end{cases}$$

The methodology has been discussed in section VI-B. For 5-bit SMPO example, we start from word-level expression polynomial

$$A + a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18}$$

and the result is

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

By replacing bit-level variable a_i with b_i, r_i or R_i, and word-level variable A with B, r, R respectively, we can directly get bit-word relation functions for another operand input, pseudo input and pseudo output.

One limitation to Singular tool is the exponential cannot exceed 2^{63}, so when doing experiments for SMPO larger than 62 bits, we use a little trick (the feasibility of this trick can also be verified in following steps). Since the BLVS method only requires squaring of equations each time, the exponential of word A can only be in the form 2^{i-1}, i.e. $A^{2^0}, A^{2^1}, \ldots, A^{2^{k-1}}$. To minimize the exponential presenting in Singular tool, we rewrite 2^{i-1} to i, i.e. $(A^{2^0}, A^{2^1}, \ldots, A^{2^{k-1}}) \to (A, A^2, \ldots, A^k)$. In this way result is rewritten to be

$$a_0 = (\alpha+1)A^5 + (\alpha^4+\alpha^3+\alpha)A^4 + (\alpha^3+\alpha^2)A^3 + (\alpha^4+1)A^2 + (\alpha^2+1)A$$

Thus the exponential will not exceed the Singular data size limit.

This step requires limited substitution operations in Singular, so although we use the naive Gaussian elimination method (whose time complexity is $O(k^3)$), the time cost is trivial comparing to following steps. For 33 bits experiment, pre-process execution time is 2.7 sec; while for 100 bits experiment time cost is 36 sec.

(ii) Spoly reduction:

First, Spoly is calculated based on RATO, then reduced with the ideal composed by circuit description polynomials (J). For already finished experiments, naive reduction (multi-division) is adopted, and this step takes largest portion of total time consumption.

For SMPO experiments, reduced Spoly has following generic form (all coefficients are omitted):

$$redSpoly = \sum r_i + \sum a_i b_i + \sum a_i B + \sum b_i A + R + r$$

Note there is no cross-term for bit-level or word-level variables from same side such as $a_i a_j, a_i A$, etc. Consider the necessary condition of our trick, this property of reduced Spoly guarantees the word level variable can only exist in the form $A^{2^{i-1}}$, after substituting bit-level variables with corresponding word-level variable.

(iii) Substitute bit-level variables in reduced Spoly:

Use the result from pre-process, get rid of r_i, a_i and b_i through substitution. This step yields following polynomial (consider the trick we used):

$$R + \sum r^i + \sum A^i B^j$$

all coefficients omitted.

(iv) Substitute present state word-level variable r with inputs A and B:

According to section V-B, there is still a polynomial r_{in} in the ideal we want to compute Gröbner basis. This polynomial has form $r + f'(A,B)$, which is last clock cycle's output ($R + f'(A,B)$) with only leading term replaced in step "$from^i \leftarrow to^i$" in algorithm 2. Basically this step has nothing different from last one, however, it must be taken good care of when using our trick. There is power of r, r^m is originally $r^{2^{m-1}}$; so if $r + f'(A,B)$ contains terms $A^i B^j$, the correct result after doing power is

$$(A^{2^{i-1}} B^{2^{j-1}})^{2^{m-1}} = A^{2^{((i+m-2) \bmod k)+1}} B^{2^{((j+m-2) \bmod k)+1}}$$

So the correct exponential for A and B in $(A^i B^j)^m$ should be $((i+m-2) \bmod k)+1$ and $((j+m-2) \bmod k)+1$, respectively.

Within one main loop, after finishing steps (ii) to (iv), the output should be intermediate multiplication result $R + f(A,B)$. After k loops, the output is $R + A \cdot B$ when SMPO is bug-free.

B. Discussion

I have concerns on this SMPO experiment because we use abstraction polynomial $R + f(A,B)$ instead of eliminating to univariate polynomial $g(R)$. The only point of inconsistency to the FSM traversal we proposed is, currently the ideal quotient theory's correctness is only guaranteed under univariate assumption. I think we have some options here: 1, expand the ideal quotient theory to multivariate; 2, argue that we use abstraction only to make it easier to check the correctness of our experiment and let the output make sense to people; 3, abandon SMPO experiment in this paper focusing on FSM traversal, try to do some new experiments on reachability checking, leave this SMPO experiment for some new paper concerning functional verification.

VIII. Experimental Results

Using our approach described above, we have performed experiments to verify SMPO circuits over $\mathbb{F}_{2^k}$ with SINGULAR symbolic algebra computation system [v. 3-1-6][**?**]. Bugs are also introduced into SMPO designs for runtime comparison. To show the advantages of our approach, experiments using traditional methods such as SAT, BDD are also conducted. Our experiments run on a desktop with 3.5GHz Intel Core$^{\text{TM}}$ i7 Quad-core CPU, 16 GB RAM and 64-bit Linux.

A. Evaluation of SAT/BDD based methods

For every SMPO design, there is an equation in following form (this is a 5-bit example):

$$R_i = b_i a_{i+1} + b_{i+1}(a_i + a_{i+3}) + b_{i+2}(a_{i+3} + a_{i+4}) \\ + b_{i+3}(a_{i+1} + a_{i+2}) + b_{i+4}(a_{i+2} + a_{i+4}),\ 0 \le i \le 4$$

This equation can be used as specification of SMPO circuit. For SAT/BDD based methods, we first unroll our SMPO design, then create a *miter* with the unrolled design and circuit specification to check if it is unsatisfiable.

The SAT solver we use is *Lingeling*[**?**]. Single-output miter circuit is written in DIMACS CNF[**?**] format and fed to Lingeling. For BDD based approach, we use the BDD module integrated with VIS tool. The miter is described by a BLIF file and read by VIS[**?**]. After BDD is built, if there is no path to leaf "1", then the miter is unsatisfiable.

The runtime results given in seconds include experiments for 11, 18, 23, 33 and 50 bits unrolled SMPO and their specifications. They show that both SAT and BDD based approach cannot verify SMPO beyond 33 bits.

	Word size of the operands k-bits				
Solver	11	18	23	33	50
Lingeling	0.5	12.9	157.4	5931.2	*TO*
BDD	0.02	5.7	237.9	??	*TO*

TABLE III: Runtime for verification of bug-free SMPO circuits over $\mathbb{F}_{2^k}$ for SAT and BDD based methods. *TO* = timeout of 4 hrs

B. Evaluation of Our Approach

Our approach requires only one time reduction for each clock cycle using RATO. If we use built-in function in SINGULAR to directly compute Gröbner basis, the high-degree word definition polynomial will exceed SINGULAR's capacity, even for 11 bits SMPO.

We also introduce bugs to our SMPO designs; these bugs are simply swapping an arbitrary pair of gate output signals, they will make the final result be a far more complicated polynomial instead of $R + A \cdot B$, so the runtime will be slightly longer than bug-free one.

Appendix

A. Normal Basis Multiplication and λ-Matrix

λ-Matrix is defined with cross-product terms from multiplication. That is

$$Product\ C = (\sum_{i=0}^{n-1} a_i \beta^{2^i})(\sum_{j=0}^{n-1} b_j \beta^{2^j}) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \beta^{2^i}\beta^{2^j}$$

The expressions $\beta^{2^i}\beta^{2^j}$ are referred to as cross-product terms, and can be represented by normal basis, i.e.

$$\beta^{2^i}\beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ij}^{(k)} \beta^{2^k}, \quad \lambda_{ij}^{(k)} \in F_2.$$

Substitution yields, get the expression for k-th digit of product:

$$c_k = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \lambda_{ij}^{(k)} a_i b_j$$

$\lambda_{ij}^{(k)}$ is the entry with coordinate (i,j) in k-th λ-Matrix.

Example A.1. *λ-Matrix: A binary $n \times n$ matrix M could be employed to describe the "function" mentioned in Ex.III.2: $c_{n-1} = f(A,B) = A \cdot M \cdot B^T$, B^T denotes vector transposition. More specifically, denote the matrix by k-th λ-Matrix: $c_k = A \cdot M^{(k)} \cdot B^T$. Then $c_{k-1} = A \cdot M^{(k-1)} \cdot B^T = rotate(A) \cdot M^{(k)} \cdot rotate(B)^T$, which means $M^{(k)}$ is generated by right and down cyclic shifting $M^{(k-1)}$.*

In $\mathbb{F}_{2^3}$ constructed by $\alpha^3+\alpha+1$, let $\beta = \alpha^3$, $N = \{\beta, \beta^2, \beta^4\}$ is a normal basis. 0-th λ-Matrix

$$M^{(0)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

i.e.,

$$c_0 = (a_0\ a_1\ a_2)\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}.$$

B. Optimal Normal Basis

Definition A.1. *The number of non-zero entries in λ-Matrix is known as* Complexity(C_N).

Theorem A.1. *If N is a normal basis for $\mathbb{F}_{p^n}$ with λ-matrix $M^{(k)}$, then non-zero entries in matrix $C_N \geq 2n-1$.*

Proof omitted.

Definition A.2. *If there exists a set of normal basis satisfying $C_N = 2n-1$, this normal basis is named as* Optimal Normal Basis (ONB).

There are 2 types of optimal normal basis existing. Rules for creating Type-I ONB over $\mathbb{F}_{2^n}$ are:

- n+1 must be prime.
- 2 must be primitive in $\mathbb{Z}_{n+1}$.

Rules for creating Type-II ONB over $\mathbb{F}_{2^n}$ are:

- 2n+1 must be prime. And either
- 2 must be primitive in $\mathbb{Z}_{2n+1}$, OR
- $2n+1 \equiv 3 \quad mod \quad 4$ AND 2 generates the quadratic residues in $\mathbb{Z}_{2n+1}$

There are corresponding criteria for simply creating λ-Matrix for either type of ONB:

Lemma A.1. *Type-I ONB implies a λ-Matrix that each nonzero entry $M_{i,j}$ satisfies*

$$\begin{cases} 2^i + 2^j = 1 \bmod n+1 \\ 2^i + 2^j = 0 \bmod n+1 \end{cases}$$

Type-II ONB implies a λ-Matrix that each nonzero entry $M_{i,j}$ satisfies

$$\begin{cases} 2^i + 2^j = 1 \bmod\ 2n+1 \\ 2^i + 2^j = -1 \bmod 2n+1 \\ 2^i - 2^j = 1 \bmod 2n+1 \\ 2^i - 2^j = -1 \bmod 2n+1 \end{cases}$$

Proof omitted. (Issues here: We knew type-I & II definition can deduce corresponding λ-Matrix, how about reverse implication? i.e. can we prove the equivalence?)

C. Design a Normal Basis Multiplier using λ-Matrix

Imitating the structure of SMPO, just specify the gates' connections for the first clock cycle, then following cycles are automatically completed by cyclic shifting operands A and B. The whole design procedure is based on a $n \times n$ k-th λ-Matrix.

First figure out the first row (*row* 0) in λ-Matrix, for any nonzero entry $M_{0,j}$ (there will be only 1 if taking 0-th λ-Matrix), place an AND gate $a_j \wedge b_0$. Connect different a_j with a XOR gate before place the AND gate if there are 2 or more nonzero entries in this row;

Secondly, repeat this for each row. Note for nonzero entry $M_{i,j}$, corresponding index of a_u and b_v is incremented by i, i.e. $u = j+i \pmod n$, $v = i+i \pmod n$;

Finally, connect the output of AND gate for row i to one input of a XOR gate, then connect the output of XOR gate to a register/flip-flop. The output of register/flip-flop is connected to the other input of XOR gate, but at next row. All these gates and connections for row i is called unit R_i. (Fig.4)

D. Find Optimal Normal Basis

This step is done by directly constructing the field using special primitive polynomial $P(x)$. Primitive element α which is root of that primitive polynomial($P(\alpha) = 0$), is also optimal normal element. i.e. $\beta = \alpha$.

The special polynomial for type-I and type-II optimal normal basis can be computed from algorithms showed in page 108, IEEE standard 1363-2000 document.

References

[1] Herve J Touati, Hamid Savoj, Bill Lin, Robert K Brayton, and Alberto Sangiovanni-Vincentelli, "Implicit state enumeration of finite state machines using bdd's", *in Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pp. 130–133. IEEE, 1990.

[2] Wolfgang Günther, Andreas Hett, and Bernd Becker, "Application of linearly transformed bdds in sequential verification", *in Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pp. 91–96. ACM, 2001.

[3] Amit Narayan, Adrian J Isles, Jawahar Jain, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli, "Reachability analysis using partitioned-robdds", *in Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pp. 388–393. IEEE Computer Society, 1997.

[4] Jerry R Burch, Edmund M Clarke, and David E Long, "Representing circuits more efficiently in symbolic model checking", *in Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 403–407. ACM, 1991.

[5] J-HR Jiang and Robert K Brayton, "On the verification of sequential equivalence", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, pp. 686–697, 2003.

[6] Bijan Alizadeh and Masahiro Fujita, "Sequential equivalence checking using a hybrid boolean-word level decision diagram", *in Advances in Computer Science and Engineering*, pp. 697–704. Springer, 2009.

[7] Olivier Coudert, Christian Berthet, and Jean Christophe Madre, "Verification of synchronous sequential machines based on symbolic execution", *in Automatic verification methods for finite state systems*, pp. 365–373. Springer, 1990.

[8] Per Bjesse and Koen Claessen, "Sat-based verification without state space traversal", *in Formal Methods in Computer-Aided Design*, pp. 409–426. Springer, 2000.

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