Efficient Characteristic Set Algorithms for Equation Solving in Finite Fields and Application in Analysis of Stream Ciphers[1]

Xiao-Shan Gao and Zhenyu Huang
Key Laboratory of Mathematics Mechanization
Institute of Systems Science, AMSS, Chinese Academy of Sciences

Abstract. Efficient characteristic set methods for computing solutions of a polynomial equation system in a finite field are proposed. We introduce the concept of proper triangular sets and prove that proper triangular sets are square-free and have solutions. We present an improved algorithm which can be used to reduce the zero set of an equation system in general form to the union of zero sets of proper triangular sets. Bitsize complexity for the algorithm is given in the case of Boolean polynomials. We also give a characteristic set method for Boolean polynomials, where the size of the polynomials are effectively controlled. The methods are implemented and extensive experiments show that they are quite efficient for solving equations raised in analyzing certain classes of stream ciphers.

Keywords. Characteristic set, proper triangular set, finite field, Boolean function, stream cipher.

1. Introduction

Solving polynomial equations in finite fields plays a fundamental role in many important fields such as coding theory, cryptology, design and analysis of computer hardware. To find efficient algorithms to solve such equations is a central issue both in mathematics and in computer science (see Problem 3 in [39] and Section 8 of [13]). Efficient algebraic algorithms for solving equations in finite fields have been developed, such as the Gröbner basis methods [2, 6, 16, 17, 19, 25, 22, 38] and the XL algorithm and its improved versions [14].

The **characteristic set (CS)** method is a tool for studying polynomial, algebraic differential, and algebraic difference equation systems [1, 4, 5, 9, 10, 15, 20, 21, 23, 24, 26, 28, 29, 30, 34, 40, 41, 43]. The idea of the method is reducing equation systems in general form to equation systems in the form of triangular sets. With this method, solving an equation system can be reduced to solving univariate equations in cascaded form. In the case of finite fields, univariate equations can be solved with Berlekamp's algorithm [31]. The CS method can also be used to compute the dimension, the degree, and the order for an equation system, to solve the radical ideal membership problem, and to prove theorems from elementary and differential geometries [42].

[1] Partially supported by a National Key Basic Research Project of China and a grant from NSFC.

In most existing work on CS methods, the zeros of the equations are taken in an algebraically closed field which is infinite. These methods can also be used to solve equations in finite fields. But, they do not take into the account of the special properties of the finite fields and thus are not efficient for solving equations in finite fields. In this paper, we propose efficient CS methods to solve equations in the general finite field $\mathbb{F}_q$ with q elements. More precisely, we will develop efficient CS algorithms for polynomial systems in the ring

$$\mathbb{R}_q = \mathbb{F}_q[x_1, \dots, x_n]/(\mathbb{H})$$

where $\mathbb{H} = \{x_1^q - x_1, \dots, x_n^q - x_n\}$. Due to the special property of $\mathbb{R}_q$, the proposed CS methods are more efficient and have better properties than the general CS method.

A triangular set may have no solutions in a finite field. For instance, $x^2 + 1 = 0$ has no solution in the finite field $\mathbb{F}_3$. To avoid this problem, we introduce the concept of proper triangular sets and prove that proper triangular sets are square-free. We also give an explicit formula for the number of solutions of a proper triangular set.

We propose an improved zero decomposition algorithm which allows us to decompose the zero set of a polynomial equation system in $\mathbb{R}_q$ as the disjoint union of the zero sets of proper triangular sets. As a consequence, we can give an explicit formula for the number of solutions of the equation system. We also show that the improved zero decomposition algorithms have better complexity bounds than the general CS method. We prove that our elimination procedure to compute a triangular set needs a polynomial number of polynomial multiplications. In the general CS method, this procedure is exponential [20].

An element in $\mathbb{R}_2$ is called a Boolean polynomial. Solving Boolean polynomial systems is especially important and more methods are available. This paper will focus on CS methods. We show that for Boolean polynomial equations, the CS method proposed in this paper and that proposed in [8] for Boolean polynomials could be further improved. First, we give a bitsize complexity for the zero decomposition algorithm proposed in this paper. This is the first complexity analysis for the zero decomposition algorithm. The results in [20] are only for the procedure to compute one CS, which is called well-ordering procedure by Wu [41].

We also present a multiplication-free CS algorithm in $\mathbb{R}_2$, where the size of the polynomials occurring in the well-ordering procedure is bounded by the size of the input polynomial system and the worst case bitsize complexity of the algorithm is roughly $O(n^d)$. This result is surprising, because repeated additions of polynomials can also generate polynomials of exponential sizes. In the general CS method, the size of the polynomials is exponential [20]. Our result also means that for a small d, the well-ordering procedure is a polynomial-time algorithm in n. The bottle neck problem of intermediate expression swell is effectively avoided for certain classes of problems due to the low complexity of the well-ordering procedure and the usage of SZDD [33]. Our experimental results also support this observation.

We conduct extensive experiments of our methods for three kinds of polynomial systems. These systems are generated in totally different ways, but they all have the block triangular structure. By block triangular structure, we mean that the polynomial set can be divided into disjoint sets such that each set consists of polynomials with the same leading variable and different sets have different leading variables. Polynomial sets generated in many classes of stream ciphers are in triangular block form. The experiments show that our improved algorithm is very effective for solving these polynomial equations comparing to existing methods. We do not claim that our algorithm is faster in all cases. For instance, the first HFE Challenge, which was solved by the Gröbner basis algorithm [18, 35], can not be solved by our algorithm.

The rest of this paper is organized as follows. In Section 2, we introduce the notations. In Section 3, we prove properties for the proper triangular sets. In Section 4, we present the improved zero decomposition algorithm. In Section 5, we present a CS algorithm in $\mathbb{R}_2$. In Section 6, we present the experimental results. In section 7, conclusions are presented.

2. Notations and Preliminary Results

Let p be a prime number and $q = p^k$ for a positive integer k. $\mathbb{F}_q$ denotes the finite field with q elements. For an algebraic equation, we will consider the problem of finding its solutions in $\mathbb{F}_q$. Let $\mathbb{X} = \{x_1, \dots, x_n\}$ be a set of indeterminants. Since we only consider solutions in $\mathbb{F}_q$, we can work in the ring

$$\mathbb{R}_q = \mathbb{F}_q[\mathbb{X}]/(\mathbb{H})$$

where

$$\mathbb{H} = \{x_1^q - x_1, x_2^q - x_2, \dots, x_n^q - x_n\}. \tag{1}$$

When we want to emphasize the variables, we use the notation $\mathbb{R}_q[x_1, \dots, x_n]$ instead of $\mathbb{R}_q$. It is easy to see that $\mathbb{R}_q$ is not an integral domain. For any $\alpha \in \mathbb{F}_q$, $x_i - \alpha$ is a zero divisor in $\mathbb{R}_q$. An element P in $\mathbb{R}_q$ has the following canonical representation:

$$P = \alpha_s M_s + \cdots + \alpha_0 M_0, \quad \alpha_i \in \mathbb{F}_q, \tag{2}$$

where M_i is a monomial and $\deg(M_i, x_j) \le q - 1$ for any j . We still call an element in $\mathbb{R}_q$ a **polynomial**. In this paper, a polynomial is always in its canonical representation.

Let $\mathbb{P}$ be a set of polynomials in $\mathbb{R}_q$. We use $\text{Zero}_q(\mathbb{P})$ to denote the common zeros of the polynomials in $\mathbb{P}$ in the affine space $\mathbb{F}_q^n$, that is,

$$\text{Zero}_q(\mathbb{P}) = \{(a_1, \dots, a_n), a_i \in \mathbb{F}_q, s.t., \forall P \in \mathbb{P}, P(a_1, \dots, a_n) = 0\}.$$

In this paper, when we say a **variety** in $\mathbb{F}_q^n$, we mean $\text{Zero}_q(\mathbb{P})$ for some $\mathbb{P} \subseteq \mathbb{R}_q[x_1, \dots, x_n]$. Let D be a polynomial in $\mathbb{R}_q$. We define a **quasi variety** to be

$$\text{Zero}_q(\mathbb{P}/D) = \text{Zero}_q(\mathbb{P}) \setminus \text{Zero}_q(D).$$

Let $\mathbb{P}$ be a set of polynomials in $\mathbb{F}_q[\mathbb{X}]$. Denote the zeros of $\mathbb{P}$ in an algebraically closed extension of $\mathbb{F}_q$ as $\text{Zero}(\mathbb{P})$. We use $\overline{\mathbb{P}}$ to denote the image of $\mathbb{P}$ under the natural ring homomorphism:

$$\mathbb{F}_q[\mathbb{X}] \Rightarrow \mathbb{R}_q.$$

We will give some preliminary results about the polynomials in $\mathbb{R}_q$.

Lemma 2.1 *Use the notations just introduced. We have* $\text{Zero}(\mathbb{P} \cup \mathbb{H}) = \text{Zero}_q(\overline{\mathbb{P}})$*, where* $\mathbb{H}$ *is defined in (1).*

Proof: Let $P \in \mathbb{P}$. By the definition, we have $P = \overline{P} + \sum_i B_i(x_i^q - x_i)$, where B_i are some polynomials. Note that any zero in $\mathrm{Zero}_q(\overline{\mathbb{P}})$ is also a zero of $x_i^q - x_i$. Then the formula to be proved is a direct consequence of the above relation between P and $\overline{P}$. □

Lemma 2.2 *Let P be a polynomial in $\mathbb{R}_q$. We have $P^q = P$.*

Proof: Since $x_i^q = x_i$, for any monomial m in $\mathbb{R}_q$ we have $m^q = m$. Let $P = \sum_i \alpha_i m_i$ where m_i are monomials and $\alpha_i \in \mathbb{F}_q$. Then $P^q = (\sum_i \alpha_i m_i)^q = \sum_i \alpha_i^q m_i^q = \sum_i \alpha_i m_i = P$. □

Lemma 2.3 *Let* I *be a polynomial ideal in $\mathbb{R}_q$. Then* I *is a radical ideal.*

Proof: For any $f^s \in \mathrm{I}$ with s an integer, there exists an integer k such that $q + k(q-1) \geq s$. Then $f^s f^{q+k(q-1)-s} = f^{q+k(q-1)} \in I$. By Lemma 2.2, $f^{q+k(q-1)} = f^q f^{k(q-1)} = f^{k(q-1)+1} = f^{q+(k-1)(q-1)} = \cdots = f^q = f$. Thus, we have $f \in I$, which implies that I is a radical ideal. □

Lemma 2.4 *Let* I *be a polynomial ideal in $\mathbb{R}_q$.*

(1) $\mathrm{I} = (x_0 + a_0, \ldots, x_n + a_n)$ *if and only if* $(a_0, \ldots, a_n)$ *is the only solution of* I.

(2) $\mathrm{I} = (1)$ *if and only if* I *has no solutions.*

Proof: If $\mathrm{I} = (x_0 + a_0, \ldots, x_n + a_n)$, it is easy to see that $(a_0, \ldots, a_n)$ is the only solution of I. Conversely, let $(a_0, \ldots, a_n)$ be the only solution of I. By Lemma 2.1, we have $x_i + a_i = 0$ on $\mathrm{Zero}(\mathrm{I} \cup \mathbb{H})$ in $\mathbb{F}_q[\mathbb{X}]$, where $\mathbb{H}$ is defined in (1). By Hilbert's Nullstellensatz, there is an integer s such that $(x_i + a_i)^s$ is in the ideal generated by $\mathrm{I} \cup \mathbb{H}$ in $\mathbb{F}_q[\mathbb{X}]$. Considering $\mathbb{R}_q$, it means that $(x_i + a_i)^s$ is in I. By Lemma 2.3, I is a radical ideal in $\mathbb{R}_q$. Thus, $x_i + a_i$ is in I. This prove (1). For (2), if I has no solution, we have $\mathrm{Zero}(\mathrm{I} \cup \mathbb{H}) = \emptyset$. By Hilbert's Nullstellensatz, $1 \in (\mathrm{I} \cup \mathbb{H})$. That is, $1 \in \mathrm{I}$. □

Lemma 2.5 *Let $P \in \mathbb{R}_q$.* $\mathrm{Zero}_q(P) = \mathbb{F}_q^n$ *iff* $P \equiv 0$. $\mathrm{Zero}_q(P) = \emptyset$ *iff* $P^{q-1} - 1 \equiv 0$.

Proof: If $P \equiv 0$, then $\mathrm{Zero}_q(P) = \mathbb{F}_q^n$. Conversely, we prove the result by induction on n. If $n = 1$, we consider the univariate polynomial $P(x) \in \mathbb{R}_q$. Suppose that $P(x) \neq 0$. Since $\deg(P, x) \leq q - 1$, P has at most $q - 1$ solutions in $\mathbb{F}_q$, a contradiction. Now assume that the result has been proved for $n = k$. For $n = k + 1$, we have $P(x_1, \ldots, x_n) = f_0 x_n^{q-1} + f_1 x_n^{q-2} + \cdots + f_{q-1}$, where f_i is a k-variable polynomial. By the induction hypothesis, if some f_i is not 0, there exists an element $(a_1, a_2, \ldots, a_k)$ in $\mathbb{F}_q^k$ such that $f_i(a_1, \ldots, a_k) \neq 0$. Then $P(a_1, \ldots, a_k)$ is a nonzero polynomial whose degree in x_{k+1} is less than q. Supposing a_{k+1} is not the solution of $P(a_1, \ldots, a_k)$, $(a_1, \ldots, a_{k+1})$ is not the solution of P, a contradiction. Thus, we have $f_i = 0$ for all i. It means that $P \equiv 0$, and the first result is proved.

If $\mathrm{Zero}_q(P) = \emptyset$, then $P \neq 0$ for any element in $\mathbb{F}_q^n$, which implies that $P^{q-1} - 1 = 0$ for any element in $\mathbb{F}_q^n$. Then $P^{q-1} - 1 \equiv 0$. Conversely, suppose that there is an element $\alpha \in \mathbb{F}_q^n$ such that $P(\alpha) = 0$, which is impossible since $P^{q-1}(\alpha) - 1 \neq 0$. Thus, $\mathrm{Zero}_q(P) = \emptyset$. □

As a consequence of Lemma 2.5, we have

Corollary 2.6 *Let $q = 2$ and $P \in \mathbb{R}_2 \setminus \mathbb{F}_2$. Then* $\mathrm{Zero}_2(P) \neq \emptyset$.

But when $q > 2$, the corollary is not correct. For example, considering $\mathbb{R}_3$, it is easy to see that $\text{Zero}_3(x^2+1) = \emptyset$.

Lemma 2.7 *Let U, V, and D be polynomials in $\mathbb{R}_q$. We have*

$$(U^{q-1}V^{q-1} - 1) = (U^{q-1} - 1, V^{q-1} - 1). \tag{3}$$

$$(U^{q-1}V^{q-1} - U^{q-1} - V^{q-1}) = (U, V). \tag{4}$$

$$\text{Zero}_q(UV) = \text{Zero}_q(U) \cup \text{Zero}_q(V). \tag{5}$$

$$\text{Zero}_q(\emptyset/D) = \text{Zero}_q(D^{q-1} - 1). \tag{6}$$

$$\text{Zero}_q(\mathbb{P}) = \text{Zero}_q(\mathbb{P} \cup \{U\}) \cup \text{Zero}_q(\mathbb{P} \cup \{U^{q-1} - 1\}). \tag{7}$$

Proof: We have

$$\begin{aligned}(U^{q-1}V^{q-1} - 1) &= (U^{q-1}V^{q-1} - 1, U^{q-1}(U^{q-1}V^{q-1} - 1)) \\ &= (U^{q-1}V^{q-1} - 1, U^{q-1}V^{q-1} - U^{q-1}) \\ &= (U^{q-1}V^{q-1} - 1, U^{q-1} - 1) = (U^{q-1} - 1, V^{q-1} - 1).\end{aligned}$$

This proves (3). Equation (4) can be proved similarly:

$$\begin{aligned}(U^{q-1}V^{q-1} - U^{q-1} - V^{q-1}) &= (U^{q-1}V^{q-1} - U^{q-1} - V^{q-1}, U(U^{q-1}V^{q-1} - U^{q-1} - V^{q-1})) \\ &= (U^{q-1}V^{q-1} - U^{q-1} - V^{q-1}, U) = (U, V).\end{aligned}$$

Since $\mathbb{F}_q$ is a field, (5) is obvious. For any element $\alpha \in \mathbb{F}_q^n$, $D(\alpha) \neq 0$ means that $D^{q-1}(\alpha) - 1 = 0$. Conversely, for any element $\alpha \in \mathbb{F}_q^n$, if $D(\alpha) = 0$, we have $D^{q-1}(\alpha) - 1 \neq 0$. This proves (6). Since $U(U^{q-1} - 1) \equiv 0$, (7) is a consequence of (5). □

From (6) of Lemma 2.7, we can see that a quasi variety in $\mathbb{F}_q^n$ is also a variety.

3. Proper Triangular Sets in $\mathbb{R}_q$

In this section, we will introduce the concept of proper triangular sets for which we can give an explicit formula for its number of solutions.

3.1 Triangular Sets

Let $P \in \mathbb{R}_q$. The **class** of P, denoted by $\text{cls}(P)$, is the largest c such that x_c occurs in P. Then x_c is called the **leading variable** of P, denoted as $\text{lvar}(P)$. If $P \in \mathbb{F}_q$, we set $\text{cls}(P) = 0$. If $\text{cls}(P) = c$, let us regard P as a univariate polynomial in x_c. We call $\deg(P, x_c)$ the **degree** of P, denoted as $\deg(P)$. The coefficient of P wrt x_c^d is called the **initial** of P, and is denoted by $\text{init}(P)$. Then P can be represented uniquely as the following form:

$$P = I x_c^d + U \tag{8}$$

where $I = \text{init}(P)$ and U is a polynomial with $\deg(U, x_c) < d$. A polynomial P_1 has **higher ordering** than a polynomial P_2, denoted as $P_2 \prec P_1$, if $\text{cls}(P_1) > \text{cls}(P_2)$ or $\text{cls}(P_1) = \text{cls}(P_2)$ and $\deg(P_1) > \deg(P_2)$. If neither $P_1 \prec P_2$ nor $P_2 \prec P_1$, they are said to have the same ordering, denoted as $P_1 \sim P_2$. It is easy to see that $\prec$ is a partial order on the polynomials in $\mathbb{R}_q$.

A sequence of nonzero polynomials

$$\mathcal{A}: \quad A_1, A_2, \ldots, A_r \tag{9}$$

is a **triangular set** if either $r = 1$ and $A_1 \neq 0$ or $0 < \text{cls}(A_1) < \cdots < \text{cls}(A_r)$. A **trivial** triangulated set is a polynomial set consisting of a nonzero element in $\mathbb{F}_q$. For a triangular set $\mathcal{A}$, we denote $\mathbf{I}_{\mathcal{A}}$ to be the product of the initials of the polynomials in $\mathcal{A}$.

Let $\mathcal{A}' : A'_1, A'_2, \ldots, A'_{r'}$ and $\mathcal{A}'' : A''_1, A''_2, \ldots, A''_{r''}$ be two triangular sets. $\mathcal{A}'$ is said to be of **lower ordering** than $\mathcal{A}''$, denoted as $\mathcal{A}' \prec \mathcal{A}''$, if either there is some k such that $A'_1 \sim A''_1, \ldots, A'_{k-1} \sim A''_{k-1}$, while $A'_k \prec A''_k$; or $r' > r''$ and $A'_1 \sim A''_1, \ldots, A'_{r''} \sim A''_{r''}$. We have the following basic property for triangular sets.

Lemma 3.1 *A sequence of triangular sets steadily lower in ordering is finite. More precisely, let $\mathcal{A}_1 \succ \mathcal{A}_2 \succ \cdots \succ \mathcal{A}_m$ be a strictly decreasing sequence of triangular sets in $\mathbb{R}_q$. Then $m \leq q^n$.*

Proof: Let P be a polynomial in $\mathbb{R}_q$. If $\text{cls}(P) = c$ and $\deg(P) = d$, P and x_c^d have the same ordering. Since we only consider the ordering of the triangular sets, we may assume that the triangular sets consist of powers of variables. In this case, two distinct triangular sets can not have the same ordering. To form a triangular set of this kind, we can choose one polynomial M_i from $\{0, x_i, x_i^2, \ldots, x_i^{q-1}\}$ for each i, and the triangular set is $M_1, M_2, \ldots, M_n$. Note that when $M_i = 0$, we will remove it from the triangular set. Thus, there are $q^n - 1$ nontrivial triangular sets consist of powers of variables. Adding the trivial triangular set consist of 1, we have a sequence of triangular sets $\mathcal{C}_1 \succ \mathcal{C}_2 \succ \cdots \succ \mathcal{C}_{q^n}$. Let $\mathcal{A}_1 \succ \mathcal{A}_2 \succ \cdots \succ \mathcal{A}_m$ be a strictly decreasing sequence of triangular sets. If $\mathcal{A}_i$ is nontrivial, for $P \in \mathcal{A}_i$, replace it by $\text{lvar}(P)^{\deg(P)}$. If $\mathcal{A}_i$ is trivial, replace it by 1. Then we get a strictly decreasing sequence of triangular sets $\mathcal{B}_1 \succ \mathcal{B}_2 \succ \cdots \succ \mathcal{B}_m$. This sequence must be a sub-sequence of $\mathcal{C}_1 \succ \mathcal{C}_2 \succ \cdots \succ \mathcal{C}_{q^n}$. Hence, $m \leq q^n$. □

For two polynomials P and Q, we use $\text{prem}(Q, P)$ to denote the pseudo-remainder of Q with respect to P. For a triangular set $\mathcal{A}$ defined in (9), the **pseudo-remainder** of Q wrt $\mathcal{A}$ is defined recursively as

$$\text{prem}(Q, \mathcal{A}) = \text{prem}(\text{prem}(Q, A_r), A_1, \ldots, A_{r-1}) \text{ and } \text{prem}(Q, \emptyset) = Q.$$

Let $R = \text{prem}(Q, \mathcal{A})$. Then we have

$$I_1^{s_1} I_2^{s_2} \cdots I_r^{s_r} Q = \sum_i Q_i A_i + R \tag{10}$$

where $I_i = \text{init}(A_i)$ and Q_i are some polynomials. The above formula is called the **remainder formula**. Let $\mathbb{P}$ be a set of polynomials and $\mathcal{A}$ a triangular set. We use $\text{prem}(\mathbb{P}, \mathcal{A})$ to denote the set of nonzero $\text{prem}(P, \mathcal{A})$ for $P \in \mathbb{P}$.

A polynomial Q is **reduced** wrt $P \neq 0$ if $\text{cls}(P) = c > 0$ and $\deg(Q, x_c) < \deg(P)$. A polynomial Q is **reduced** wrt a triangular set $\mathcal{A}$ if P is reduced wrt to all the polynomials in $\mathcal{A}$. It is clear that the pseudo-remainder of any polynomial wrt $\mathcal{A}$ is reduced wrt $\mathcal{A}$.

The **saturation ideal** of a triangular set $\mathcal{A}$ is defined as follows

$$\mathrm{sat}(\mathcal{A}) = \{P \in \mathbb{R}_q |\ JP \in (\mathcal{A})\}$$

where J is a product of certain powers of the initials of the polynomials in $\mathcal{A}$. We have

Lemma 3.2 *Let $\mathcal{A} = A_1, \ldots, A_r$ be a triangular set. Then* $\mathrm{sat}(\mathcal{A}) = (A_1, \ldots, A_r, \mathbf{I}_{\mathcal{A}}^{q-1} - 1)$

Proof: Denote $\mathrm{I} = (A_1, \ldots, A_r, A_0)$ and $A_0 = \mathbf{I}_{\mathcal{A}}^{q-1} - 1$. If $P \in \mathrm{sat}(\mathcal{A})$, then $\mathbf{I}_{\mathcal{A}}^{q-1} P \in A$. There exist polynomials B_i such that $\mathbf{I}_{\mathcal{A}}^{q-1} P = \sum_{i=1}^{r} B_i A_i$. Hence, $P = \sum_{i=1}^{r} B_i A_i - P A_0 \in \mathrm{I}$. Conversely, let $P \in \mathrm{I}$. Then there exist polynomials C_i such that $P = \sum_{i=1}^{r} A_i + C_0 A_0$. Multiply $\mathbf{I}_{\mathcal{A}}$ to both sides of the equation. Since $\mathbf{I}_{\mathcal{A}}(\mathbf{I}_{\mathcal{A}}^{q-1} - 1) = 0$, we have $\mathbf{I}_{\mathcal{A}} P = \sum_{i=1}^{r} \mathbf{I}_{\mathcal{A}} C_i A_i$. Thus, $P \in \mathrm{sat}(\mathcal{A})$. □

As shown by the following example, saturation ideals have different properties comparing with that in the usual polynomial ring.

Example 3.3 *Let $\mathcal{A} = A_1, A_2$, $A_1 = (x_1 - 1)x_2, A_2 = (x_1 + 1)x_3$. Then* $\mathrm{sat}(\mathcal{A}) = (A_1, A_2, (x_1^2 - 1)^2 - 1) = (x_2, x_3, x_1)$.

3.2 Proper Triangular Sets

As we mentioned before, a triangular set could have no zero. For example, $\mathrm{Zero}_3(x^2+1) = \emptyset$. To avoid this problem, we introduce the concept of proper triangular sets.

A triangular set $\mathcal{A} = A_1, A_2, \ldots, A_r$ is called **proper**, if the following condition holds: if $\mathrm{cls}(A_i) = c_i$ and $\deg(A_i) = d_i$, then $\mathrm{prem}(x_{c_i}^{q-d_i} A_i, \mathcal{A}) = 0$.

The following lemmas show that proper triangular sets always have solutions.

Lemma 3.4 *Let $P(x)$ be a univariate polynomial in $\mathbb{R}_q$, and suppose that* $\deg(P(x)) = d$. *If prem$(x^{q-d}P(x), P(x)) = 0$, then $P(x) = 0$ has d distinct solutions in $\mathbb{F}_q$.*

Proof: Since $P(x)$ is a univariate polynomial, $\mathrm{init}(P) \in \mathbb{F}_q$. If $\mathrm{prem}(x^{q-d}P(x), P(x)) = 0$ in $\mathbb{R}_q$, we have $x^{q-d}P(x) = Q(x)P(x)$, where $Q(x)$ is a polynomial and $\deg(Q(x)) < q - d$. Considering the above equation in $\mathbb{F}_q[x]$, there is a polynomial C such that $x^{q-d}P(x) + C(x^q - x) = Q(x)P(x)$ in $\mathbb{F}_q[x]$, where $x^{q-d}P(x) + C(x^q - x)$ is equal to the canonical representation of $\overline{x^{q-d}P(x)}$ in $\mathbb{R}_q$. Thus, we have $(x^{q-d} - Q(x))P(x) = -C(x^q - x)$. Since all the elements of $\mathbb{F}_q$ are solutions of $x^q - x$, the q distinct elements of $\mathbb{F}_q$ are solutions of $(x^{q-d} - Q(x))P(x)$. Note that $\deg(Q(x)) < q - d$. Then $\deg(x^{q-d} - Q(x)) = q - d$. Thus, $x^{q-d} - Q(x)$ has at most $q - d$ solutions in $\mathbb{F}_q$, which means that $P(x)$ has at least d distinct solutions in $\mathbb{F}_q$. However, $\deg(P(x)) = d$ implies $P(x)$ has at most d solutions in $\mathbb{F}_q$. Hence, we can conclude $P(x)$ has d distinct solutions in $\mathbb{F}_q$. □

A triangular set $\mathcal{A}$ is called **monic** if the initial of each polynomial in $\mathcal{A}$ is 1. A monic triangular set is of the following form:

$$A_1 = x_{c_1}^{d_1} + U_1, A_2 = x_{c_2}^{d_2} + U_2, \cdots, A_r = x_{c_r}^{d_r} + U_r$$

where U_i is a polynomial in $x_1, \ldots, x_{c_i}$ such that $\deg(U_i, x_{c_i}) < d_i$.

For a monic triangular set $\mathcal{A} : A_1, \ldots, A_r$, we call $\deg(A_1)\deg(A_2)\cdots\deg(A_r)$ the **degree** of $\mathcal{A}$, denoted as $\deg(\mathcal{A})$. Let $\mathbb{Y}$ be the set $\{x_i \in \mathbb{X} | \, x_i$ is the leading variable of some $A_j \in \mathcal{A}\}$. We use $\mathbb{U}$ to denote $\mathbb{X} \setminus \mathbb{Y}$ and call the variables in $\mathbb{U}$ **parameters** of $\mathcal{A}$. Then we call $|\mathbb{U}|$ the **dimension** of $\mathcal{A}$, denoted as $\dim(\mathcal{A})$.

The following result shows that a monic proper triangular set has nice properties by giving an explicit formula for the number of solutions. The result is useful because we will prove later that the zero set for any polynomial system can be decomposed as the union of the zero sets of monic proper triangular sets.

Theorem 3.5 *Let $\mathcal{A}$ be a monic triangular set. Then $\mathcal{A}$ is proper if and only if* $|\mathrm{Zero}_q(\mathcal{A})| = \deg(\mathcal{A}) \cdot q^{\dim(\mathcal{A})}$.

Proof: Assume that $\mathcal{A}$ is proper. For the parameters in $\mathbb{U}$, we can substitute them by any element of $\mathbb{F}_q$. Since $|\mathbb{U}| = \dim(\mathcal{A})$, there are $q^{\dim(\mathcal{A})}$ parametric values for $\mathbb{U}$. For a parametric value U_0 of $\mathbb{U}$ and a polynomial $P \in \mathbb{R}_q$, let P' denote $P(U_0)$. After the substitution, we obtain a new monic triangular set $\mathcal{A}' : A'_1, \ldots, A'_r$, where $\mathrm{cls}(A'_i) = \mathrm{cls}(A_i)$ and $\deg(A'_i) = \deg(A_i)$. Let $c_i = \mathrm{cls}(A_i)$ and $d_i = \deg(A_i)$. Since $\mathcal{A}$ is a proper triangular set, we have $x_{c_1}^{q-d_1} A_1 = P A_1$. Then $x_{c_1}^{q-d_1} A'_1 = P'_1 A'_1$. By Lemma 3.4, A'_1 has d_1 distinct solutions. For a solution α of A'_1, consider $A'_2(\alpha)$. Since $\mathcal{A}$ is proper, we have $x_{c_2}^{q-d_2} A_2 = Q_1 A_1 + Q_2 A_2$ and hence $x_{c_2}^{q-d_2} A'_2(\alpha) = Q'_1(\alpha) A'_1(\alpha) + Q'_2(\alpha) A'_2(\alpha)$. Since $A'_1(\alpha) = 0$, we have $x_{c_2}^{q-d_2} A'_2(\alpha) = Q'_2(\alpha) A'_2(\alpha)$. By Lemma 3.4, $A'_2(\alpha)$ has d_2 distinct solutions. By repeating the process, we can prove that $\mathcal{A}'$ has $d_1 d_2 \cdots d_r = \deg(\mathcal{A})$ distinct solutions. Hence, $|\mathrm{Zero}_q(\mathcal{A})| = \deg(\mathcal{A}) \cdot q^{\dim(\mathcal{A})}$.

Conversely, let us assume that $\mathcal{A}$ has $N = \deg(\mathcal{A}) \cdot q^{\dim(\mathcal{A})}$ solutions. Since $\mathcal{A}$ is monic, it means that for any parametric value U_0 of $\mathbb{U}$ and any point x in $\mathrm{Zero}_q(A_1(U_0), \ldots, A_{i-1}(U_0))$, $A_i(U_0, x)$ has $\deg(A_i)$ distinct solutions. Let $A_i = I_i x_{c_i}^{d_i} + V_i$ for any i. For A_1, suppose $\mathrm{prem}(x_{c_1}^{q-d_1} A_1, \mathcal{A}) = R_1 \neq 0$. Then we have $(x_{c_1}^{q-d_1} - P_1) A_1 = R_1$, where P_1 is a polynomial. Choose a parametric value U_0 of $\mathbb{U}$ such that $R_1(U_0) \neq 0$. Then $A_1(U_0)$ has d_1 distinct solutions, this is contradicts to $0 < \deg(R_1(U_0), x_{c_1}) < d_1$. Thus, $R_1 = 0$. Now we consider A_2. Suppose $\mathrm{prem}(x_{c_2}^{q-d_2} A_2, \mathcal{A}) = R_2 \neq 0$. Then we have two polynomials Q_1 and Q_2 such that $x_{c_2}^{q-d_2} A_2 = Q_1 A_1 + Q_2 A_2 + R_2$. Choose a parametric value U_1 of $\mathbb{U}$ such that $R_2(U_1) \neq 0$. Since $\deg(R_2, x_{c_1}) < d_1$, there is a solution x of $A_1(U_1)$ such that $R_2(U_1, x) \neq 0$. Then we have $(x_{c_2}^{q-d_2} - Q_1(U_1, x)) A_2(U_1, x) = R_2(U_1, x)$. $A_2(U_1, x)$ has d_2 distinct solutions which contradicts to $0 < \deg(R_2(U_1, x_{c_2})) < d_2$. Thus, $R_2 = 0$. Similarly, we have $\mathrm{prem}(x_{c_i}^{q-d_i} A_i, \mathcal{A}) = 0$. Hence, $\mathcal{A}$ is proper. □

As a consequence of Theorem 3.5, a monic proper triangular set is square-free.

4. An Efficient Zero Decomposition Algorithm in $\mathbb{R}_q$

In this section, we will give an improved algorithm which can be used to decompose the zero set of a polynomial system into the union of zero sets of monic triangular sets. Due to the special property of $\mathbb{R}_q$, this algorithm has lower complexities than the general zero decomposition algorithm and the output is stronger.

First, note that the following zero decomposition theorem [10, 24, 28, 30, 40, 41] is still valid and the proof is also quite similar.

Theorem 4.1 *There is an algorithm which permits to determine for a given polynomial set $\mathbb{P}$ in a finite number of steps triangular sets $\mathcal{A}_j, j = 1, \ldots, s$ such that*

$$\mathrm{Zero}_q(\mathbb{P}) = \cup_{j=1}^{s}\mathrm{Zero}_q(\mathcal{A}_j/\mathbf{I}_{\mathcal{A}_j}) = \cup_{j=1}^{s}\mathrm{Zero}_q(\mathrm{sat}(\mathcal{A}_j))$$

where $\mathrm{sat}(\mathcal{A}_j)$ *is the saturation ideal of* $\mathcal{A}_j$*.*

In $\mathbb{R}_q$, we can give the following improved zero decomposition theorem which allows us to compute the number of solutions for a finite set of polynomials.

Theorem 4.2 *For a finite polynomial set $\mathbb{P}$, we can compute monic proper triangular sets $\mathcal{A}_j, j = 1, \ldots, s$ such that*

$$\mathrm{Zero}_q(\mathbb{P}) = \cup_{i=1}^{s}\mathrm{Zero}_q(\mathcal{A}_i)$$

such that $\mathrm{Zero}_q(\mathcal{A}_i) \cap \mathrm{Zero}_q(\mathcal{A}_j) = \emptyset$ *for $i \neq j$. As a consequence, we have*

$$|\mathrm{Zero}_q(\mathbb{P})| = \sum_{i=1}^{s}\deg(\mathcal{A}_i)\cdot q^{\dim(\mathcal{A}_i)}.$$

4.1 A Top-Down Characteristic Set Algorithm

In this section, we will give a **top-down characteristic set algorithm TDCS** that allows us to compute a decomposition which has the properties mentioned in Theorem 4.2.

Before giving the zero decomposition algorithm, we first give an algorithm to compute a triangular set. The algorithm works from the polynomials with the largest class and hence is a top-down zero decomposition algorithm. The idea of top-down elimination is explored in [26, 40]. The key idea of the algorithm is as follows. Let $Q = Ix_c^d + U$ be a polynomial with largest class and smallest degree in x_c in a polynomial set $\mathbb{Q}$. If $I = 1$, we can reduce the degrees of the polynomials in $\mathbb{Q}$ by taking $\mathbb{R} = \mathrm{prem}(\mathbb{Q}, Q)$. Since $I = 1$, we have

$$\mathrm{Zero}_q(\mathbb{Q}) = \mathrm{Zero}_q(\mathbb{R} \cup \{Q\}).$$

If $I \neq 1$, by (7), we split the zero set into two parts:

$$\mathrm{Zero}_q(\mathbb{Q}) = \mathrm{Zero}_q(\mathbb{Q} \cup \{I^{q-1} - 1\}) \cup \mathrm{Zero}_q(\mathbb{Q} \setminus \{Q\} \cup \{I, U\}). \qquad (11)$$

In the first part, since $I \neq 0$ and $I^{q-1} - 1 = 0$, Q can be replaced by $Q_1 = x_c^d + I^{q-2}U$ and we can treat this part as in the first case. The second part is simpler than $\mathbb{Q}$ and can be treated recursively. The following **well ordering procedure** is based on the above idea.

Algorithm 4.3 —TDTriSet($\mathbb{P}$)

Input: *A finite set of polynomials $\mathbb{P}$.*

Output: *A monic triangular set $\mathcal{A}$ and a set of polynomial systems $\mathbb{P}^*$ such that* $\mathrm{Zero}(\mathbb{P}) = \mathrm{Zero}(\mathcal{A}) \cup_{\mathbb{Q}\in\mathbb{P}^*} \mathrm{Zero}(\mathbb{Q})$, $\mathrm{Zero}(\mathcal{A}) \cap \mathrm{Zero}(\mathbb{Q}_1) = \emptyset$, *and* $\mathrm{Zero}(\mathbb{Q}_1) \cap \mathrm{Zero}(\mathbb{Q}_2) = \emptyset$ *for all* $\mathbb{Q}_1, \mathbb{Q}_2 \in \mathbb{P}^*$.

```
1 Set $\mathcal{A}=\emptyset$ and $\mathbb{P}^*=\emptyset$.
2 While $\mathbb{P}\neq\emptyset$ do
    2.1 If some nonzero element $\alpha$ of $\mathbb{F}_q$ is in $\mathbb{P}$, $\mathrm{Zero}_q(\mathbb{P})=\emptyset$. Return $\mathcal{A}=\emptyset$ and $\mathbb{P}^*$.
    2.2 Let $\mathbb{P}_1\subset\mathbb{P}$ be the polynomials with the highest class.
    2.3 Let $Q\in\mathbb{P}_1$ be a polynomial with lowest degree.
    2.4 Let $Q=Ix_c^d+U$ such that $\mathrm{cls}(Q)=c$, $\deg(Q)=d$ and $\mathrm{init}(Q)=I$.
    2.5 If $I=1$ do
        2.5.1 Set $\mathbb{R}=\mathrm{prem}(\mathbb{P}_1,Q)$.
        2.5.2 If the classes of polynomials in $\mathbb{R}$ are lower than $c$
                (this situation will always happen when $q=2$), do
                $\mathcal{A}=\mathcal{A}\cup\{Q\}$.
                $\mathbb{P}=\mathbb{R}\cup\{\mathbb{P}\setminus\mathbb{P}_1\}$.
        2.5.3 Else, do
                $\mathbb{P}=\mathbb{R}\cup\{Q\}\cup\{\mathbb{P}\setminus\mathbb{P}_1\}$ and goto 2.2.
    2.6 Else do
        2.6.1 Set $Q_1=x_c^d+I^{q-2}U$ and $\mathbb{P}_2=\mathbb{P}_1\setminus\{Q\}$.
        2.6.2 $\mathbb{P}=\mathrm{prem}(\mathbb{P}_2,Q_1)\cup\{I^{q-1}-1\}\cup\{\mathbb{P}\setminus\mathbb{P}_1\}$.
        2.6.3 $\mathbb{P}_1=\{\mathbb{P}\setminus\{Q\}\}\cup\mathcal{A}\cup\{I,U\}$.
        2.6.4 $\mathbb{P}^*=\mathbb{P}^*\cup\{\mathbb{P}_1\}$.
        2.6.5 Set $\mathbb{R}=\mathrm{prem}(\mathbb{P}_2,Q_1)$.
        2.6.6 If the classes of polynomials in $\mathbb{R}$ are lower than $c$, do
                $\mathcal{A}=\mathcal{A}\cup\{Q_1\}$.
        2.6.7 Else, do
                $\mathbb{P}=\mathbb{P}\cup\{Q_1\}$. and goto 2.2.
3 Return $\mathcal{A}$ and $\mathbb{P}^*$.
```

The following theorem shows that to compute a monic triangular set in $\mathbb{R}_q$, we need only a polynomial number of polynomial arithmetic operations. Note that if the zero set is in an algebraically closed field, the process to compute a triangular set is exponential [20].

Theorem 4.4 *Algorithm* **TDTriSet** *is correct and in the whole algorithm we need $O(n^2q^2+nlq)$ polynomial multiplications where $l=|\mathbb{P}|$. In particular, we need $O(nl)$ polynomial multiplications when $q=2$.*

Proof: Let $\mathbb{P}_1\subset\mathbb{P}$ be the set of polynomials with the highest class c and $Q\in\mathbb{P}_1$ a polynomial with lowest degree in x_c. Let $c=\mathrm{cls}(Q)$, $d=\deg(Q)$ and $I=\mathrm{init}(Q)$. If $I=1$, then for $P\in\mathbb{P}_1$, as a consequence of remainder formula (10), $\mathrm{Zero}_q(\{Q,P\})=\mathrm{Zero}_q(\{Q,\mathrm{prem}(P,Q)\})$. Therefore, we have

$$\mathrm{Zero}_q(\mathbb{P})=\mathrm{Zero}_q((\mathbb{P}\setminus\mathbb{P}_1)\cup\{Q\}\cup\{\mathrm{prem}(P,Q)\neq 0\,|\,P\in\mathbb{P}_1\}).$$

If $I\neq 1$, by (7), we can split $\mathrm{Zero}_q(\mathbb{P})$ as the following two parts:

$$\begin{aligned}\mathrm{Zero}_q(\mathbb{P})&=\mathrm{Zero}_q(\mathbb{P}\cup\{I^{q-1}-1\})\cup\mathrm{Zero}_q(\mathbb{P}\cup\{I\}) &(12)\\ &=\mathrm{Zero}_q((\mathbb{P}\setminus\{Q\})\cup\{Q_1\}\cup\{I^{q-1}-1\})\cup\mathrm{Zero}_q((\mathbb{P}\setminus\{Q\})\cup\{I,U\}) &(13)\end{aligned}$$

where $Q_1 = x_c + I^{q-2}U$. The first part of (13) can be treated similarly to the case of $I = 1$, and the second part of (13) will be a polynomial set in the output. This proves that if we have the output it must be correct.

Now let us prove the termination of the algorithm. After each iteration of the loop, the lowest degree of the polynomials with highest class in $\mathbb{P}$ will decrease. Then the highest class of the polynomials in $\mathbb{P}$ will be reduced and the polynomial Q will be added to $\mathcal{A}$. Hence, the loop will end and give a triangular set $\mathcal{A}$ and some polynomial sets $\mathbb{P}^*$.

Finally, we will analyze the complexity of the algorithm. Let $l = |\mathbb{P}|$. After each iteration, the lowest degree of the highest class of the polynomials in $\mathbb{P}$ will be reduced at least by one. Then, this loop will execute at most $n(q-1)$ times. After each iteration, if $I = 1$, then the new $\mathbb{P}$ has at most l polynomials. If $I \neq 1$, after this iteration there are two cases:

(a) Except Q we still have some polynomials with this class. Then, the new $\mathbb{P}$ contains at most $l + 1$ polynomials;

(b) The highest class is eliminated by Q. Then, the new $\mathbb{P}$ contains at most l polynomials.

Therefore, in the whole algorithm there are at most $n(q-2) + l$ polynomials (The number is l when $q = 2$) .

In an iteration, suppose we use $Q = Ix_c^d + U$ to eliminate other polynomials. First we should set Q to be monic. It means that we should compute $Q_1 = x_c^d + I^{q-2}U$ and $I^{q-1} - 1$, so we need $2(q-2)$ polynomial multiplications. Thus, in the whole algorithm we need at most $2n(q-1)(q-2)$ polynomial multiplications in order to obtain the monic polynomials. Then we want to get $\mathrm{prem}(P, Q_1)$. Since Q_1 is monic, it takes at most one polynomial multiplication when we reduce the degree of P by one. Let D be the sum of the degrees of polynomials with highest class. Then D decreases by one after one polynomial multiplication. Therefore, we need at most $(n(q-2)+l)(q-1)-1$ multiplications to reduce D from $(n(q-2)+l)(q-1)$ to 1. At the same time, we eliminate the highest class. Thus, in the whole algorithm, we need at most $n^2(q-2)(q-1)+nl(q-1)-n$ polynomial multiplications to get the pseudo-remainders. In all, the algorithm needs $O(n^2q^2 + nlq)$ polynomial multiplications, and when $q = 2$ the number is $O(nl)$. □

Lemma 4.5 *Let* $\mathbb{P}$ *be an input of* **TDTriSet**. *Assume that there is a polynomial* P *in* $\mathbb{P}$ *such that* $\mathrm{cls}(P) = c$ *and* $\mathrm{init}(P) = 1$. *Let* $\mathcal{A}$ *be the monic triangular set in the output. Then, there is a polynomial* $P' \in \mathcal{A}$ *such that* $\mathrm{cls}(P') = c$ *and* $\deg(P') \leq \deg(P)$.

Proof: Since there is a P with class c, we need to deal with this class. And we will eliminate this class by P or by a Q with class c and lower degree. This polynomial is the P'. □

By using **TDTriSet**, we have the following **zero decomposition algorithm**.

Algorithm 4.6 — TDCS($\mathbb{P}$)

Input: *A finite set of polynomials* $\mathbb{P}$.
Output: *Monic proper triangular sets satisfying the properties in Theorem 4.2.*

```
1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$ and $\mathcal{C}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$ do
    2.1 Choose a polynomial set $\mathbb{Q}$ from $\mathbb{P}^*$.
    2.2 Let $\mathbb{Q}$ be the input of TDTriSet. Let $\mathcal{A}$ and $\mathbb{Q}^*$ be the output.
    2.3 if $\mathcal{A} \neq \emptyset$, set $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$.
    2.4 $\mathbb{P}^* = \mathbb{P}^* \cup \mathbb{Q}^*$
3 Suppose $\mathcal{A}^* = \{\mathcal{A}_1, \ldots, \mathcal{A}_q\}$ and $\mathcal{A}_i = \{A_{i1}, \ldots, A_{ip_i}\}$.
4 For i from 1 to q do
    4.1 Set $\mathcal{B} = \emptyset$.
    4.2 For j from 1 to $p_i$ do
        4.2.1 Let $\mathrm{cls}(A_{ij}) = c_{ij}$ and $\deg(A_{ij}) = d_{ij}$.
        4.2.2 $\mathcal{B} = \mathcal{B} \cup \{\mathrm{prem}(x_{c_{ij}}^{q-d_{ij}} A_{ij}, \mathcal{A}_i)\} \neq 0$.
    4.3 If $\mathcal{B} \neq \emptyset$, do $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathcal{A}_i \cup \mathcal{B}\}$.
    4.4 Else, do $\mathcal{C}^* = \mathcal{C}^* \cup \{\mathcal{C}\}$
5 If $\mathbb{P}^* \neq \emptyset$, do
    5.1 Set $\mathcal{A}^* = \emptyset$, goto 2.
6 Return $\mathcal{C}^*$
```

Theorem 4.7 *Algorithm* **TDCS** *is correct.*

Proof: By Theorem 4.4, if the loop in step 2 ends, we can obtain $\mathcal{A}_1, \ldots, \mathcal{A}_q$ such that $\mathrm{Zero}(\mathbb{P}) = \cup_i \mathrm{Zero}(\mathcal{A}_i)$. In step 4, we check whether $\mathcal{A}_i$ is a proper triangular set. If it is proper, we save it in the output list $\mathcal{C}^*$. If $\mathcal{A}_i$ is not proper, suppose $\mathcal{A}_i = A_{i1}, \ldots, A_{ip_i}$. we add $\mathrm{prem}(x_{c_{ij}}^{q-d_{ij}} A_{ij}, \mathcal{A}_i) \neq 0$ to $\mathcal{A}_i$, and obtain a new polynomials set $\mathcal{B}_i$. We have $\mathrm{Zero}_q(\mathcal{A}_i) = \mathrm{Zero}_q(\mathcal{A}_i, x_{c_{ij}}^{q-d_{ij}} A_{ij}) = \mathrm{Zero}_q(\mathcal{A}_i, \mathrm{prem}(x_{c_{ij}}^{q-d_{ij}} A_{ij}, \mathcal{A}_i))$. Thus, $\mathrm{Zero}_q(\mathcal{A}_i) = \mathrm{Zero}_q(\mathcal{B}_i)$. Then we treated $\mathcal{B}_i$ recursively by step 2. Hence, if $\mathcal{A}'_1, \ldots, \mathcal{A}'_s$ is the output of the algorithm, we have $\mathrm{Zero}_q(\mathbb{P}) = \cup_i \mathrm{Zero}_q(A'_i)$.

Now we need to show the termination for the algorithm. First, we prove the termination of step 2. For a polynomial set $\mathbb{P}$, we assign an index $(c_{n,q-1}, c_{n,q-2}, \ldots, c_{n,1}, \ldots, c_{1,q-1}, \ldots, c_{1,1})$ where $c_{i,j}$ is the number of polynomials in $\mathbb{P}$ and with class i and degree j. In step 2, we add $\mathbb{Q}' = (\mathbb{Q} \setminus \{Q\}) \cup \{I, U\}$ which is in the output of **TDTriSet** to $\mathbb{P}^*$, where $Q = Ix_c + U$. It is clear that the index of $\mathbb{Q}'$ is less than the index of $\mathbb{Q}$ in the lexicographical ordering. It is easy to show that a strictly decreasing sequence of indexes must be finite. This proves the termination of the step 2.

Suppose we obtain $\mathcal{A}^* = \mathcal{A}_1, \ldots, \mathcal{A}_q$ after step 2. If all $\mathcal{A}_i$ are proper, the algorithm will terminate. If $\mathcal{A}_i = A_{i1}, \ldots, A_{ip_i}$ is not proper, similar as above, we obtain a polynomial set $\mathcal{B}_i$ such that there exist polynomials in $\mathcal{B}_i$, which are reduced wrt $\mathcal{A}_i$. To prove the termination of the whole algorithm, it is sufficient to show that the new monic triangular sets we obtain from $\mathcal{B}_i$ in step 2 is of lower ordering than that of $\mathcal{A}_i$. Note that $\mathcal{B}_i \setminus \mathcal{A}_i$ is the set of polynomials in $\mathcal{B}_i$ which are reduced wrt $\mathcal{A}_i$.

Now let $\mathbb{Q}_1$ be the set of polynomials with highest class in $\mathcal{B}_i \setminus \mathcal{A}_i$ and Q be the one of lowest degree in $\mathbb{Q}_1$. Let $Q = Ix_c^d + U$. Then in **TDTriSet**, we splits $\mathrm{Zero}_q(\mathcal{B}_i)$ into two parts:

$$\mathrm{Zero}_q(\mathcal{B}_i) = \mathrm{Zero}_q(\{\mathcal{B}_i \setminus \{Q\}\} \cup \{x_c^d + I^{q-2}U\} \cup \{I^{q-1}-1\}) \cup \mathrm{Zero}_q(\{\mathcal{B}_i \setminus \{Q\}\} \cup \{I, U\}).$$

Note that $\mathcal{A}_i \subseteq \mathcal{B}_i$ and if there is a polynomial A' in $\mathcal{A}_i$ with class c then $deg(A') > deg(x_c^d + I^{q-2}U)$. Thus, by Lemma 4.5, we can conclude that the monic triangular sets we obtain from $\{\mathcal{B}_i \setminus \{Q\}\} \cup \{x_c^d + I^{q-2}U\} \cup \{I^{q-1}-1\}$ is of lower ordering than $\mathcal{A}_i$. For $\{\mathcal{B}_i \setminus \{Q\}\} \cup \{I, U\}$, it can be recursively treated as $\mathcal{B}_i$. Hence, we prove the termination of the algorithm. □

We use the following simple example to illustrate how the algorithm works.

Example 4.8 *In $\mathbb{R}_3$, let $\mathbb{P} = \{x_1x_2x_3^2 - 1\}$.*

In Algorithm **TDTriSet***, we have* $\mathrm{Zero}_3(\mathbb{P}) = \mathrm{Zero}_3(x_3^2 - x_1x_2, x_1^2x_2^2 - 1) \cup \mathrm{Zero}_3(x_1x_2, 1)$*. Obviously,* $\mathrm{Zero}_3(x_1x_2, 1) = \emptyset$*. Then,* $\mathrm{Zero}_3(\mathbb{P}) = \mathrm{Zero}_3(x_3^2 - x_1x_2, x_1^2x_2^2 - 1) = \mathrm{Zero}_3(x_3^2 - x_1x_2, x_2^2 - 1, x_1^2 - 1) \cup \mathrm{Zero}_3(x_1^2, 1)$*. The algorithm returns $\mathcal{A} = \{x_1^2 - 1, x_2^2 - 1, x_3^2 - x_1x_2\}$ and $\emptyset$.*

In Algorithm **TDCS***, we check whether $\mathcal{A}$ is proper:* $\mathrm{prem}(x_3(x_3^2 - x_1x_2), \mathcal{A}) = (1 - x_1x_2)x_3$*,* $\mathrm{prem}(x_2(x_2^2 - 1), \mathcal{A}) = \mathrm{prem}(x_1(x_1^2 - 1), \mathcal{A}) = 0$*. We obtain a new $\mathbb{P}' = \{\mathcal{A}, (x_1x_2 - 1)x_3\}$ such that* $\mathrm{Zero}_3(\mathbb{P}) = \mathrm{Zero}_3(\mathbb{P}')$*.*

Execute Algorithm **TDTriSet** *with input $\mathbb{P}'$. Choose $(x_1x_2 - 1)x_3$ to eliminate x_3. Then* $\mathrm{Zero}_3(\mathbb{P}') = \mathrm{Zero}_3(x_3, x_3^2 - x_1x_2, x_2^2 - 1, x_1x_2 + 1, x_1^2 - 1) \cup \mathrm{Zero}_3(x_3^2 - x_1x_2, x_1x_2 - 1, x_2^2 - 1, x_1^2 - 1)$*. For the first part, we have* $\mathrm{Zero}_3(x_3, x_3^2 - x_1x_2, x_2^2 - 1, x_1x_2 + 1, x_1^2 - 1) = \mathrm{Zero}_3(x_3, x_1x_2, x_2^2 - 1, x_1x_2 + 1, x_1^2 - 1) = \emptyset$*. For the second part, we execute Algorithm* **TDTriSet** *again and have* $\mathrm{Zero}_3(x_3^2 - x_1x_2, x_1x_2 - 1, x_2^2 - 1, x_1^2 - 1) = \mathrm{Zero}_3(x_3^2 - x_1x_2, x_2 - x_1, x_2^2 - 1, x_1^2 - 1) \cup \mathrm{Zero}_3(x_3^2 - x_1x_2, x_2^2 - 1, x_1^2 - 1, x_1, 1) = \mathrm{Zero}_3(x_3^2 - x_1x_2, x_2 - x_1, x_1^2 - 1)$*. Let $\mathcal{A}' = \{x_3^2 - x_1x_2, x_2 - x_1, x_1^2 - 1\}$. Thus,* $\mathrm{Zero}_3(\mathbb{P}) = \mathrm{Zero}_3(\mathcal{A}')$*.*

Returning to Algorithm **TDCS***, it is easy to check that $\mathcal{A}'$ is proper. Then we have* $\mathrm{Zero}_3(\mathbb{P}) = \mathrm{Zero}_3(x_3^2 - 1, x_2 - x_1, x_1^2 - 1)$*, and* $|\mathrm{Zero}_3(\mathbb{P})| = 3^0(2 \times 1 \times 2) = 4$.

4.2 Complexity Analysis of TDCS in $\mathbb{R}_2$

As we mentioned in Section 1, a complexity analysis for the zero decomposition algorithm is never given. Although, **TDCS** is much simpler than the zero decomposition algorithm over the field of complex numbers, it is still too difficult to give a complexity analysis. However, we are able to give a worst case complexity analysis for algorithm **TDCS** in the very important case of $\mathbb{R}_2$.

In $\mathbb{R}_2$, it is easy to prove that a monic triangular set is always proper. Therefore, we do not need to check whether a triangular set is proper in Algorithm **TDCS**. Moreover, by (4), we can modify the Step 2.6.3 of **TDTriSet** as

$$\mathbb{P}_1 = \{\mathbb{P} \setminus \{Q\}\} \cup \mathcal{A} \cup \{U, I\} = \{\mathbb{P} \setminus \{Q\}\} \cup \mathcal{A} \cup \{IU + I + U\},$$

and call the new algorithm **TDTriSet**$_2$. After this modification, the number of polynomials in the new component $\mathbb{P}_1$ will not be bigger than $|\mathbb{P}|$. From the proof of Theorem 4.4, we know that in the whole algorithm **TDTriSet**$_2$ with input $\mathbb{P}$ the number of polynomials is also at most $|\mathbb{P}|$. Then we obtain the following algorithm:

Algorithm 4.9 — $\mathbf{TDCS}_2(\mathbb{P})$

Input: *A finite set of Boolean polynomials $\mathbb{P}$.*
Output: *A sequence of monic triangular sets satisfying Theorem 4.2.*

```
1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$ and $\mathcal{C}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$ do
    2.1 Choose a polynomial set $\mathbb{Q}$ from $\mathbb{P}^*$.
    2.2 Let $\mathbb{Q}$ be the input of TDTriSet$_2$. Let $\mathcal{A}$ and $\mathbb{Q}^*$ be the output.
    2.3 if $\mathcal{A} \neq \emptyset$, set $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$.
    2.4 $\mathbb{P}^* = \mathbb{P}^* \cup \mathbb{Q}^*$
3 Return $\mathcal{A}^*$
```

Theorem 4.10 *The bitsize complexity of Algorithm* $\mathbf{TDCS}_2$ *is $O(l^n) = O(2^{n \log l})$, where l is the number of polynomials in $\mathbb{P}$.*

Remark. It is interesting to note that the complexity for the exhaust search algorithm is $O(\|\mathbb{P}\| \cdot 2^n)$, where $\|\mathbb{P}\|$ is the bitsize of the polynomials in $\mathbb{P}$ as defined in Section 5.2. The complexity of the exhaust search is generally better than our algorithm. But on the other hand, our algorithm can solve nontrivial problems with $n \geq 128$ as shown in Section 6.2 and Section 6.3, while it is clear that the exhaust search algorithm cannot do that. The complexity to compute a Gröbner basis of $\mathbb{P} \cup \mathbb{H}$ ($\mathbb{H}$ is defined in (1)) is known to be a polynomial in d^n where d is the degree of the polynomials in $\mathbb{P}$ [27]. Recently, Bardet, Faugere, Salvy gave better complexity bounds under the assumption of semi-regularity [2]. It is an interesting problem that whether there exists a deterministic algorithm to find all the solutions of a Boolean polynomial system with complexity less than $O(2^n)$.

In order to estimate the complexity of algorithm $\mathbf{TDCS}_2$, we need to consider the worst case in the algorithm. We call the zero decomposition process in the worst case **W-Decomposition**.

In the worst case, we consider a set $\mathbb{P}$ of l Boolean polynomials which are with the highest class n and the initials of all these l polynomials are not 1. Then we need to choose one polynomial $Q = Ix_n + U \in \mathbb{P}$ and add $I + 1$ to $\mathbb{P}$. Let $Q_1 = x_n + U$. Then we have:

$$\mathrm{Zero}_q(\mathbb{P}) = \mathrm{Zero}_q(\mathrm{prem}(\mathbb{P} \setminus \{Q\}, Q_1), \cup\{Q_1, I+1\})) \cup \mathrm{Zero}_q(\mathbb{P} \setminus \{Q\} \cup \{IU + I + U\}) \quad (14)$$

In the worst case, we assume that the class of $I + 1$ is $n - 1$ and $\mathrm{prem}(\mathbb{P} \setminus \{Q\}, Q_1)$ contains $l - 1$ non-zero polynomials with class $n - 1$. Moreover, in the second component in (14), we have a new polynomial $IU + I + U$ which is also of class $n - 1$. When we repeat the above procedure for the two components in (14), the above situations always happen. In other words, in the worst case,when we eliminate a variable x_c, the newly generated non-zero polynomials are always of class $c - 1$.

We can illustrate the W-decomposition by the following figure:

$$\begin{tikzcd}
(l,k,\ldots,\ldots) \arrow[r, Rightarrow] \arrow[d] & (l-1,k+1,\ldots) \arrow[r, Rightarrow] \arrow[d] & (l-2,k+2,\ldots) \arrow[r, Rightarrow] \arrow[d] & \cdots \\
(0,l+k,\ldots) \Rightarrow \cdots \arrow[d] & (0,l+k,\ldots) \Rightarrow \cdots \arrow[d] & \vdots & \\
\vdots & \vdots & &
\end{tikzcd}$$

In this figure and the rest of this section, $(l_n, l_{n-1}, \cdots, l_1)$ represents a polynomial set which contains l_i polynomials with class i. The right arrows point to the second component in (14), while the down arrows point to the first component in (14) or more precisely, to $\mathrm{prem}(\mathbb{P} \setminus \{Q\}, Q_1) \cup \{I+1\}$.

To solve a polynomial set $\mathbb{P}$ with l elements, we will obtain a lot of components. We can sort these components into n groups by the variables involved in them. For any $i = 1, 2, \ldots, n$, the i-th group consists of the components where the variables to be eliminated are $\{x_1, x_2, \ldots, x_i\}$. Suppose there are k_i elements in the i-th group. We define the **time-polynomial** of $\mathbb{P}$ to be

$$B(\mathbb{P}) = k_n T_n + k_{n-1} T_{n-1} + \cdots + k_1 T_1 \tag{15}$$

where T_i is a quantity to measure the complexity for executing **TDTriSet**$_2$ whose input is a polynomial set consisting of l polynomials in i variables $\{x_1, x_2, \ldots, x_i\}$. T_i could be the bitsize of the involving polynomials or the number of arithmetic operations needed in the algorithm. Obviously, $B(\mathbb{P})$ gives the corresponding worst case complexity when the meaning of T_i is fixed.

For two polynomial sets $\mathbb{P}_1$ and $\mathbb{P}_2$, let $B(\mathbb{P}_1) = k_n T_n + \cdots + k_1 T_1$ and $B(\mathbb{P}_2) = k'_n T_n + \cdots + k'_1 T_1$. If $k_i > k'_i$ for all i, we say that $B(\mathbb{P}_1)$ is of **higher ordering** than $B(\mathbb{P}_2)$, denoted by $B(\mathbb{P}_1) > B(\mathbb{P}_2)$. We define

$$S(\mathbb{P}) = B(\mathbb{P}) - T_c$$

where c is the highest class of the polynomials in $\mathbb{P}$. Thus, $S(\mathbb{P})$ is the complexity for solving all the components which are originated from the second component in (14). The order of $S(\mathbb{P})$ can also be defined as $B(\mathbb{P})$. Therefore, we can use equation (15) as the recursive formula to compute the worst case complexity of the algorithm.

The following result shows that the problems solved with w-decomposition is indeed the worst case in terms of complexity.

Lemma 4.11 *Let $\mathbb{Q}$ be a polynomial set of the form $(l, 0, \ldots, 0)$, which need to be solved with w-decomposition. Let $B(\mathbb{P})$ be the time-polynomial of any other problem with $|\mathbb{P}| \leq l$. We have $B(\mathbb{Q}) \geq B(\mathbb{P})$ and $S(\mathbb{Q}) \geq S(\mathbb{P})$.*

Proof: We proof the lemma by induction. If $n = 1$, no components are generated, so we have $B(\mathbb{P}) = T_1$ and $S(\mathbb{P}) = 0$ for any problem, and the lemma holds for $n = 1$. Now suppose we have proved the lemma for $n = k$. If $n = k+1$, we have the following figure for the w-decomposition of problem $(l, 0, \ldots, 0)$:

$$\begin{tikzcd}
(l,0,\ldots,0) \arrow[r, Rightarrow] \arrow[d] & (l-1,1,\ldots,0) \arrow[r, Rightarrow] \arrow[d] & \cdots \arrow[r, Rightarrow] & (1,l-1,0,\ldots,0) \arrow[r, Rightarrow] \arrow[d] & (0,l,0,\ldots,0) \\
(0,l,0,\ldots,0) & (0,l,0,\ldots,0) & \cdots & (0,l,0,\ldots,0) &
\end{tikzcd}$$

We can get the following recursive formula for the time-polynomial of $(l, 0, \ldots, 0)$:

$$B(l, 0, \ldots) = lT_n + B(0, l, 0, \ldots) + lS(0, l, 0, \ldots, 0) \tag{16}$$

where $(0, l, 0, \ldots)$ represents a w-decomposition problem with l input polynomials in variable $\{x_1, \ldots, x_{n-1}\}$

For any other polynomial set $\mathbb{P}$ with no more than l input polynomials, we can write it as $(l_n, l_{n-1}, \ldots, l_1)$. If $l_n = 0$ the lemma can be proved easily from equation (16). Now we assume $l_n > 0$. For the l_n polynomials with class n, if there is a polynomial with initial 1, we will not generate any component when we eliminate class n, then $B(\mathbb{P}) = T_n + S(\mathbb{P}')$. Note that $|\mathbb{P}'| \le l$ and the elements of $\mathbb{P}'$ are all have $n-1$ variables $\{x_1, \ldots, x_{n-1}\}$. Thus $B(l, 0, \ldots) \ge B(\mathbb{P})$ and $S(l, 0, \ldots) \ge S(\mathbb{P})$ by the hypothesis.

If there exist no polynomials with initial 1 in these l_n polynomials. we have the the following decomposition figure:

$$\begin{tikzcd}
(l_n,\ldots) \arrow[r, Rightarrow] \arrow[d] & (l_n-1,\ldots) \arrow[r, Rightarrow] \arrow[d] & \cdots \arrow[r, Rightarrow] & (1,\ldots) \arrow[r, Rightarrow] \arrow[d] & \mathbb{P}_0 \\
\mathbb{P}_1 & \mathbb{P}_2 & \cdots & \mathbb{P}_{l_n} &
\end{tikzcd}$$

Thus, we have

$$B(\mathbb{P}) = l_n T_n + B(\mathbb{P}_0) + \sum_{i=1}^{l_n} S(\mathbb{P}_i).$$

Note that $\mathbb{P}_i$ has at most $n-1$ variables $\{x_1, \ldots, x_n\}$ and $|\mathbb{P}_i| \le l$, for any $i = 0, 1, \ldots, l_n$. By the hypothesis we have $S(\mathbb{P}_i) \le S(0, l, 0, \ldots, 0)$ and $B(\mathbb{P}_0) \le B(0, l, 0, \ldots, 0)$. Since $l \ge l_n$ we can conclude that $B(l, 0, \ldots) \ge B(\mathbb{P})$ and $S(l, 0, \ldots) \ge S(\mathbb{P})$. Consequently, the lemma holds in any case for $n = k + 1$. □

Proof of Theorem 4.10. From equation (16), we can obtain the value of $B(l, 0, \ldots, 0)$. Write $B(0, \ldots, 0, l, 0, \ldots, 0)$ as B_i and $S(0, \ldots, 0, l, 0, \ldots, 0)$ as S_i, where l is in the i-th coordinate. Then we have $B_n = l(T_n - T_{n-1}) + (l+1)B_{n-1}$. It is easy to check that for $n \ge 3$ we have

$$B_n = lT_n + l^2 T_{n-1} + l^2(l+1)T_{n-2} + \cdots + l^2(l+1)^{n-3}T_2 + (l+1)^{n-2}T_1.$$

If the variables of input polynomials are $\{x_1, \ldots, x_k\}$, the number of monomials occuring in **TDTriSet**$_2$ are at most 2^k, and therefore the bitsize complexity of multiplication is $2 \cdot 4^k$. By Theorem 4.4, we can substitute T_k with $(2 \cdot 4^k)k(l-1)$ for any $k \ge 2$ and T_1 can be set to 0. We have $B_n \approx 2(4^3 l^{n+1} - 4^{n+1} l^3)/(l-4)^2 + 4^3 l(l^n - 2nl4^{n-2})/(l-4)$. Since $l >> 4$, we have proved Theorem 4.10.

5. A Multiplication Free Zero Decomposition Algorithm in R_2

It is known that a major difficulty in computing a zero decomposition is the occurrence of large polynomials. In order to overcome this difficulty, we introduce a zero decomposition algorithm in $\mathbb{R}_2$, where the procedure to compute a triangular set has nice complexity bounds.

5.1 The Algorithm

The key idea of the algorithm is to avoid polynomial multiplication. Before doing the pseudo remainders, we reduce the initials of the polynomials in $\mathbb{P}_1$ in step 2.2 of the Algorithm **TDTriSet** to 1 by repeatedly using (11). For such polynomials, we have the following result.

Lemma 5.1 *Let $P = x_c + U_1$ and $Q = x_c + U_2$ be polynomials with class c and initial 1. Then, we have* $\deg(\mathrm{prem}(Q, P)) \leq \max\{\deg(P), \deg(Q)\}$.

Proof: In that case, the pseudo-remainder needs additions only: $\mathrm{prem}(Q, P) = U_1 + U_2$. The lemma follows from this formula directly. □

Based on the above idea, Algorithm **TDTriSet** can be modified to the following multiplication free (MF) **well ordering procedure** to compute a triangular set.

Algorithm 5.2 — MFTriSet($\mathbb{P}$)

Input:*A finite set of polynomials $\mathbb{P}$.*
Output: *A monic triangular set $\mathcal{A}$ and a set of polynomial systems $\mathbb{P}^*$ such that* $\mathrm{Zero}_2(\mathbb{P}) = \mathrm{Zero}_2(\mathcal{A}) \cup_{\mathbb{Q}\in\mathbb{P}^*} \mathrm{Zero}_2(\mathbb{Q})$, $\mathrm{Zero}_2(\mathcal{A}) \cap \mathrm{Zero}_2(\mathbb{Q}_1) = \emptyset$, *and* $\mathrm{Zero}_2(\mathbb{Q}_1) \cap \mathrm{Zero}_2(\mathbb{Q}_2) = \emptyset$ *for all* $\mathbb{Q}_1, \mathbb{Q}_2 \in \mathbb{P}^*$.

```
1 Set $\mathbb{P}^* = \{\}$, $\mathcal{A} = \emptyset$.
2 While $\mathbb{P} \neq \emptyset$ do
    2.1 If $1 \in \mathbb{P}$, $\mathrm{Zero}_2(\mathbb{P}) = \emptyset$. Set $\mathcal{A} = \emptyset$ and return $\mathcal{A}$ and $\mathbb{P}^*$.
    2.2 Let $\mathbb{P}_1 \subset \mathbb{P}$ be the polynomials with the highest class.
    2.3 Let $\mathbb{P}_2 = \emptyset$, $\mathbb{Q}_1 = \mathbb{P} \setminus \mathbb{P}_1$.
    2.4 While $\mathbb{P}_1 \neq \emptyset$ do
            Let $P = Ix_c + U \in \mathbb{P}_1$, $\mathbb{P}_1 = \mathbb{P}_1 \setminus \{P\}$.
            $\mathbb{Q}_2 = \mathbb{P}_1 \cup \mathbb{Q}_1 \cup \mathbb{P}_2 \cup \{I, U\}$.
            $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathbb{Q}_2\}$.
            $\mathbb{P}_2 = \mathbb{P}_2 \cup \{x_c + U\}$, $\mathbb{Q}_1 = \mathbb{Q}_1 \cup \{I + 1\}$.
    2.5 Let $Q = x_c + U$ be a polynomial with lowest degree in $\mathbb{P}_2$.
    2.6 $\mathcal{A} = \mathcal{A} \cup \{Q\}$.
    2.7 $\mathbb{P} = \mathbb{Q}_1 \cup \mathrm{prem}(\mathbb{P}_2, Q)$.
3 Return $\mathcal{A}$ and $\mathbb{P}^*$.
```

In Step 2.4, we use formula (11) in $\mathbb{R}_2$, that is,

$$\mathrm{Zero}_2(P = Ix_c + U) = \mathrm{Zero}_2(\{x_c + U, I + 1\}) \cup \mathrm{Zero}_2(\{I, U\})$$

to split the polynomial set.

With Algorithm **MFTriSet**, we can easily give a multiplication-free zero decomposition algorithm: we just need to replace Algorithm **TDTriSet** by Algorithm **MFTriSet** in Algorithm **TDCS**. We call this algorithm **MFCS** and omit the details.

Algorithm 5.3 — MFCS($\mathbb{P}$)

Input: *A finite set of polynomials $\mathbb{P}$.*
Output: *Monic proper triangular sets satisfying the properties in Theorem 4.2.*

```
1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$ and $\mathcal{C}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$ do
    2.1 Choose a polynomial set $\mathbb{Q}$ from $\mathbb{P}^*$.
    2.2 Let $\mathbb{Q}$ be the input of MFTriSet. Let $\mathcal{A}$ and $\mathbb{Q}^*$ be the output.
    2.3 if $\mathcal{A} \neq \emptyset$, set $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$.
    2.4 $\mathbb{P}^* = \mathbb{P}^* \cup \mathbb{Q}^*$
3 Return $\mathcal{A}^*$
```

Remark. In the following, we will analyze the complexity of Algorithm **MFTriSet**. Basically, we will show that the size of the polynomials in bounded by the size of the input polynomials and the worst case complexity of this algorithm is roughly $O(n^d)$. The second result implies that for a fixed d, say $d = 2$, Algorithm **MFTriSet** is a polynomial time algorithm. Note that solving quadratic Boolean equations is NP complete. In Algorithm **MFCS**, the number branches could be exponential. We will discuss this in Section 6.

5.2 Bitsize Bounds of the Polynomials in MFTriSet

In order to estimate the size of the polynomials, we introduce a **bitsize measure** for a polynomial in $\mathbb{R}_2$. Let $M = x_{i_1}x_{i_2}\cdots x_{i_k}$ be a monomial. The length of M, denoted by $\|M\|$, is defined to be k. Specially, the length of 1 is defined as 1. For a polynomial $P = M_1 + \cdots + M_t$ where M_i are monomials, $\|P\| = \sum_{i=1}^{t} \|M_i\|$ is called the **length** of P.

We first note that since Algorithm **MFCS** is multiplication free, the degrees of the polynomials occurring in the algorithm will be bounded by $d = \max_{P\in\mathbb{P}}\{\deg(P)\}$. As a consequence, the size of the polynomials occurring in the algorithm will be bounded by $O(n^d)$. Then, the size of the polynomials is effectively controlled if d is small. For all the examples in Section 6, we have $d \leq 4$ and n ranges from 40 to 128. For such examples, the polynomials have size $O(n^4)$, while the largest possible polynomials in n variables has size $O(2^n)$.

In the following theorem, we will further show that the size of the polynomials in Algorithm **MFTriSet** are effectively controlled in all cases.

Theorem 5.4 *Let n be the number of variables and $\mathbb{P}$ the input of Algorithm* **MFTriSet**. *Then, for any polynomial T occurring in Algorithm* **MFTriSet**, *we have $\|T\| \leq \sum_{P\in\mathbb{P}} \|P\|$. If $|\mathbb{P}| > n$, then there exist n polynomials $P_1, \ldots, P_n$ in $\mathbb{P}$ such that $\|T\| \leq \|P_1\| + \|P_2\| + \cdots + \|P_n\|$.*

This result is nontrivial, because repeated additions of polynomials can increase the size of the polynomials by an exponential factor. The proof of this result is quite complicated. Intuitively, we want to show that a polynomial P used in early steps of the algorithm will be "canceled" in later steps by addition of two polynomials both containing P, that is, $(P_1 + P) + (P_2 + P) = P_1 + P_2$.

In order to prove Theorem 5.4, we need to prove several lemmas first. Let k be an integer and P be a polynomial. Write $P = Ix_k + R$ as a univariate polynomial in x_k. We define two operators $\mathcal{R}_k$ and $\mathcal{J}_k$ as follows:

$$\mathcal{R}_k(P) = U, \mathcal{J}_k(P) = I + 1 \text{ if } \mathrm{cls}(P) = k. \quad \mathcal{R}_k(P) = P, \mathcal{J}_k(P) = 0 \text{ if } \mathrm{cls}(P) < k. \tag{17}$$

Then, we have the following lemma

Lemma 5.5 *Let P and Q be polynomials with* $\mathrm{cls}(P) \le k$ *and* $\mathrm{cls}(Q) \le k$. *Then*

(1) $\mathcal{R}_k(P + Q) = \mathcal{R}_k(P) + \mathcal{R}_k(Q)$;

(2) $\mathcal{R}_k(P + 1) = \mathcal{R}_k(P) + 1$;

(3) If $\mathrm{cls}(P) = \mathrm{cls}(Q) = k$ *then* $\mathcal{J}_k(P + Q) = \mathcal{J}_k(P) + \mathcal{J}_k(Q) + 1$; *otherwise* $\mathcal{J}_k(P + Q) = \mathcal{J}_k(P) + \mathcal{J}_k(Q)$.

Proof: It is easy to check. □

Note that we can define the composition of $\mathcal{R}$ and $\mathcal{J}$ naturally. Let $\mathcal{S}_{j,k} = \{\mathcal{O}_j\mathcal{O}_{j+1}\dots\mathcal{O}_k | \mathcal{O}_i = \mathcal{R}_i \text{ or } \mathcal{J}_i,\ i = j, \dots, k\}$, where $1 \le j \le k \le n$.

Lemma 5.6 *Let P be a polynomial with* $\mathrm{cls}(P) = k$. *Then* $\sum_{L_{j,i}\in\mathcal{S}_{j,k}} \|L_{j,i}P\| \le \|P\|$ *for any fixed* $j = 1, 2, \dots, k$.

Proof: For a polynomial $Q = Ix_c+U$ with $I \ne 1$, we have $\|Q\| \ge \|I\|+|U\|+1$. $\mathcal{J}_cQ = I+1$ and $\mathcal{R}_cQ = U$. Therefore, $\|\mathcal{J}_cQ\|+\|\mathcal{R}_cQ\| = \|I+1\|+\|U\| \le \|I\|+\|U\|+1 \le \|Q\|$. If $I = 1$, we have $\|\mathcal{J}_cQ\| + \|\mathcal{R}_cQ\| = 0 + \|U\| < \|Q\|$. For $i > c$, we have $\mathcal{J}_iQ = 0$ and $\mathcal{R}_iQ = Q$. Then $\|\mathcal{J}_iQ\| + \|\mathcal{R}_iQ\| = \|Q\|$. Hence, in any case, we have $|\mathcal{J}_iQ\| + \|\mathcal{R}_iQ\| \le \|Q\|$.

For any j, we have $\sum_{L_{j,i}\in\mathcal{S}_{j,k}} \|L_{j,i}P\| = \sum_{L_{j+1,i}\in\mathcal{S}_{j+1,k}}(\|\mathcal{J}_jL_{j+1,i}P\| + \|\mathcal{R}_jL_{j+1,i}P\|) \le \sum_{L_{j+1,i}\in\mathcal{S}_{j+1,k}} \|L_{j+1,i}P\| \le \dots \le \|\mathcal{J}_kP\| + \|\mathcal{R}_kP\| \le \|P\|$. □

Proof of Theorem 5.4: For any $k = 1, \dots, n$, we assume that in the k-th round of **MFTriSet** we deal with the polynomials of class k. In algorithm **MFTriSet**, when we compute the pseudo-remainder of two polynomials P and Q in the k-th round, we set their initials to 1 at first, and then compute a new polynomial $\mathcal{R}_kP + \mathcal{R}_kQ$. Thus, a polynomial $P^{(k)}$ in k-th round can be obtained in three ways:

(1) $P^{(k)}$ is an input polynomial;

(2) $P^{(k)} = \mathrm{init}(Q^{(k+i)})+1$ for some $Q^{(k+i)}$ of round $k+i$. $P^{(k)} = \mathcal{R}_{k+1}\cdots\mathcal{R}_{k+i-1}\mathcal{J}_{k+i}Q^{(k+i)}$.

(3) $P^{(k)} = \mathcal{R}_{k+j}(Q_1^{(k+j)}+Q_2^{(k+j)}) = \mathcal{R}_{k+1}\cdots\mathcal{R}_{k+j}(Q_1^{(k+j)}+Q_2^{(k+j)}) = \mathcal{R}_{k+1}\cdots\mathcal{R}_{k+j}Q_1^{(k+j)}+ \mathcal{R}_{k+1}\cdots\mathcal{R}_{k+j}Q_2^{(k+j)}$, where $Q_1^{(k+j)}$ and $Q_2^{(k+j)}$ are polynomials of round $k + j$.

In the cases 2 and 3, if i and j are bigger than 1, we still regard $\mathcal{R}_{k+2}\cdots\mathcal{R}_{k+i-1}\mathcal{J}_{k+i}Q^{(k+i)}$, $\mathcal{R}_{k+2}\cdots\mathcal{R}_{k+j}Q_1^{(k+j)}$ and $\mathcal{R}_{k+2}\cdots\mathcal{R}_{k+j}Q_2^{(k+j)}$ as polynomials of round $k+1$. In this way, we can represent $P^{(k)}$ by operators and polynomials of round $k+1$. We call it the **backtracking** representation of $P^{(k)}$. Now we can consider these polynomials of round $k+1$ and get the backtracking representation of them. By Lemma 5.5, we can get a representation of $P^{(k)}$ by composite operators and polynomials in round $k+2$. Then, we can do the process recursively. In the process of computing the backtracking representation, when meet an input polynomial, we stop representing this polynomial by the ones of higher round. At last, we backtrack to the round n, and eliminate the terms composed of the same operators and polynomials. Note that the polynomials of round n are all from the input. Then we have

$$P^{(k)} = \sum_{i=1}^{r_n} \sum_{L_j \in T_{n,i}} L_j Q_i^{(n)} + \sum_{i=1}^{r_{n-1}} \sum_{L_j \in T_{n-1,i}} L_j Q_i^{(n-1)} + \cdots + \sum_{i=1}^{r_{k+1}} \sum_{L_j \in T_{k+1,i}} L_j Q_i^{(k+1)} \tag{18}$$

or

$$P^{(k)} = \sum_{i=1}^{r_n} \sum_{L_j \in T_{n,i}} L_j Q_i^{(n)} + \sum_{i=1}^{r_{n-1}} \sum_{L_j \in T_{n-1,i}} L_j Q_i^{(n-1)} + \cdots + \sum_{i=1}^{r_{k+1}} \sum_{L_j \in T_{k+1,i}} L_j Q_i^{(k+1)} + 1 \tag{19}$$

where $T_{m,i} \subseteq \mathcal{S}_{k+1,m}$ is a set of composite operators and $Q_i^{(m)}$ is an input polynomial with class m $(m = k+1, \ldots, n, i = 1, \ldots, r_m)$. The appearance of 1 is due to the equation (3) of Lemma 5.5. The number of different polynomials in the above equation, denoted by N, is $r_{k+1} + r_{k+2} + \cdots + r_n$.

Now we will give an upper bound for N. It is easy to see that, when we backtrack to the round $k+1$, there exist at most two different polynomials. Suppose that now we backtrack to the round $k+i$, and there are t different polynomials in the representation. Then, t_1 of them are the form of $\mathcal{R}_{k+i+1} f$, where f is a polynomial with $\text{cls}(f) < k+i+1$; t_2 of them are the form of $\mathcal{J}_{k+i+1} g$, where $\text{cls}(g) = k+i+1$; t_3 of them are input polynomials. Thus, the others can be represented as $\mathcal{R}_{k+i+1} h + \mathcal{R}_{k+i+1} h_i$, where h is a fixed polynomial with $\text{cls}(h) = k+i+1$ and h_i is some polynomial with $\text{cls}(h_i) = k+i+1$. Therefore, the number of different polynomials in the representation of round $k+i+1$ is at most $2(t - t_1 - t_2 - t_3) - (t - t_1 - t_2 - t_3 - 1) + t_1 + t_2 + t_3 = t+1$. Hence, when we backtrack to the round n, we have $N \le n - k + 1$.

For any $m = k+1, \ldots, n,\ i = 1, \ldots, r_m$, since $T_{m,i} \subseteq \mathcal{S}_{k+1,m}$, by Lemma 5.6, we have $\sum_{L_j \in T_{m,i}} \|L_j Q_i^{(m)}\| \le \sum_{L_j \in \mathcal{S}_{k+1,m}} \|L_j Q_i^{(m)}\| \le \|Q_i^{(m)}\|$.

(a) Suppose that $P^{(k)}$ is of form (18). We have $\|P^{(k)}\| \le \sum_{m=k+1}^{n} \sum_{i=1}^{r_m} \|Q_i^{(m)}\|$ where $r_{k+1} + \cdots + r_n \le n - k + 1 \le n$.

(b) Suppose the representation of $P^{(k)}$ is equation (19). It is easy to see that there exists a term of the form $\mathcal{R}_{k+1} \cdots \mathcal{R}_{k+i-1} \mathcal{J}_{k+i} L Q^{(k+j)}$, where $Q^{(k+j)}$ is an input polynomial with class $k+j$, $L \in \mathcal{S}_{k+i+1,k+j}$ and $\text{cls}(LQ^{(k+j)}) = k+i$. If $\text{init}(LQ^{(k+j)}) = W + 1$ where W is a polynomial without a constant term, we have $\mathcal{J}_{k+i} LQ^{(k+j)} = W$. Therefore $\|\mathcal{J}_{k+i} LQ^{(k+j)}\| + \|\mathcal{R}_{k+i} LQ^{(k+j)}\| < \|LQ^{(k+j)}\|$. Hence, $\|P^{(k)}\| < \sum_{m=k+1}^{n} \sum_{i=1}^{r_m} \|Q_i^{(m)}\| + 1$ which means $\|P^{(k)}\| \le \sum_{m=k+1}^{n} \sum_{i=1}^{r_m} \|Q_i^{(m)}\|$. If $\text{init}(LQ^{(k+j)})$ $= W$ where W is a polynomial without a constant term, we have $\mathcal{J}_{k+i} LQ^{(k+j)} = W + 1$. Thus, $P^{(k)} = \mathcal{R}_{k+1} \cdots \mathcal{R}_{k+i-1} \mathcal{J}_{k+i} LQ^{(k+j)} + 1 + E = \mathcal{R}_{k+1} \cdots \mathcal{R}_{k+i-1} W + E$ where

E is the sum of other terms in equation (19). Obviously, $\|\mathcal{R}_{k+1}\cdots\mathcal{R}_{k+i-1}W\| < \|\mathcal{R}_{k+1}\cdots\mathcal{R}_{k+i-1}(W+1)\| = \|\mathcal{R}_{k+1}\cdots\mathcal{R}_{k+i-1}\mathcal{J}_{k+i}LQ^{(k+j)}\|$. Then $\|P\| < \|\mathcal{R}_{k+1}\cdots\mathcal{R}_{k+i-1}\mathcal{J}_{k+i}LQ^{(k+j)}\| + \|E\| \leq \sum_{m=k+1}^{n}\sum_{i=1}^{r_m}\|Q_i^{(m)}\|$.

In summary, we always have $\|P^{(k)}\| \leq \sum_{m=k+1}^{n}\sum_{i=1}^{r_m}\|Q_i^{(m)}\|$ where $r_{k+1}+\cdots+r_n \leq n-k+1 \leq n$. □

The following result shows that even the size of the monomials occurring in the algorithms is nicely bounded.

Corollary 5.7 *Let M be the set of distinct monomials which are contained in some polynomial occurring in Algorithm* **MFTriSet** *and $H = \sum_{m\in M}\|m\|$. Then, $H \leq \sum_{P\in\mathbb{P}} \mathrm{cls}(P)\|P\| + 1$ where $\mathbb{P}$ is the input of the algorithm.*

Proof: From the proof of Theorem 5.4, a polynomial P occurring in the Algorithm **MFTriSet** must have form (18) or (19). Then, a monomials m of P must be either 1 or contained in some $LQ^{(k)}$, where $Q^{(k)}$ is an input polynomial with class k and $L \in \mathcal{S}_{k-i,k}$. Thus, H is not bigger than the sum of the length of all such LQ and 1. From Lemma 5.6, $\sum_{L_{i_2}\in\mathcal{S}_{2,k}}\|L_{i_2}Q^{(k)}\| + \cdots + \sum_{L_{i_k}\in\mathcal{S}_{k,k}}\|L_{i_k}Q^{(k)}\| + \|Q^{(k)}\| \leq k\|Q^{(k)}\|$. Considering all input polynomials P and 1, we get the corollary. □

5.3 Complexity Analysis of MFTriSet

For a polynomial set $\mathbb{P}$, we define $\deg(\mathbb{P})$ to be the highest degree of the elements in $\mathbb{P}$. In this section, we will always consider a Boolean polynomial set $\mathbb{P}$ with l polynomials and $\deg(\mathbb{P}) = d$.

Theorem 5.8 *For an input polynomial set $\mathbb{P}$ with $|\mathbb{P}| = l$ and $\deg(\mathbb{P}) = d$, the bitsize complexity of* **MFTriSet** *is $O(ln^{d+1}\sum_{P\in\mathbb{P}}\mathrm{term}(P))$. If $l \geq n$, the bitsize complexity of* **MFTriSet** *is $O(ln^{d+2}M)$ where $M = \max_{P\in\mathbb{P}}\mathrm{term}(P)$.*

As a consequence, Algorithm **MFTriSet** is a polynomial-time algorithm for a small d. For all the examples in Section 6, we have $d \leq 4$ and n ranges from 40 to 128. For such examples, the complexity is $O(n^8M)$ since l is roughly $O(n^2)$.

We will prove Theorem 5.8 in the rest of this section. As in Section 5.2, we assume that in the k-th round of **MFTriSet** started as step 2, we deal with the polynomials of class k, which is the worst case. Suppose that we have l_k polynomials with class k in the k-th round. Since the complexity of computing $I+1$ is smaller than that of doing the polynomial additions, we only consider the addition of two polynomials. Then we need to do $l_k - 1$ polynomial additions in order to eliminate x_k. Thus, if we can estimate the number of the polynomials in $\mathbb{P}$ in every round, then we can obtain the complexity bound of **MFTriSet**. Note that, in Step 2.5 of **MFTriSet**, we choose a Q with the lowest degree, which is important for the complexity analysis.

Suppose that we have a polynomial set $\mathbb{S} = \{P_1, \ldots, P_l\}$ with class n, which is the worst case. After eliminating x_n, we obtain two sets of polynomials:

$$\mathbb{S}_J = \{\mathcal{J}_nP|P\in\mathbb{S}\}, \mathbb{S}_R = \{\mathcal{R}_n(P_s+P)|P\in\mathbb{S}\}$$

where P_s is a fixed polynomial with lowest degree in $\mathbb{S}$ and $\{\mathcal{J}_n,\mathcal{R}_n\}$ are the operators defined in (17). Note that $\deg(\mathbb{S}_J) \le d-1$ and $\deg(\mathbb{S}_R) \le d$. Moreover, $|\mathbb{S}_J| \le l$ and $|\mathbb{S}_R| \le l$. After eliminating x_{n-1}, we have four polynomial sets:

$$\begin{aligned}&\mathbb{S}_{JJ} = \{\mathcal{J}_{n-1}P|P \in \mathbb{S}_J\}, \mathbb{S}_{JR} = \{\mathcal{J}_{n-1}P|P \in \mathbb{S}_R\},\\ &\mathbb{S}_{RJ} = \{\mathcal{R}_{n-1}(P_s+P)|P \in \mathbb{S}_J\}, \mathbb{S}_{RR} = \{\mathcal{R}_{n-1}(P_s+P)|P \in \mathbb{S}_R\}.\end{aligned}$$

Similarly, $|\mathbb{S}_{JJ}|, |\mathbb{S}_{RJ}| \le |\mathbb{S}_J| \le l$ and $|\mathbb{S}_{JR}|, |\mathbb{S}_{RR}| \le |\mathbb{S}_R| \le l$. Since P_s is a polynomial with the lowest degree, we have $\deg(\mathcal{R}_{n-1}(P_s+P)) \le \deg(P)$ which means that $\deg(\mathbb{S}_{RR}) \le \deg(\mathbb{S}_R)$ and $\deg(\mathbb{S}_{RJ}) \le \deg(\mathbb{S}_J)$. For the other two sets, we can conclude $\deg(\mathbb{S}_{JJ}) \le \deg(\mathbb{S}_J) - 1 \le d-2$ and $\deg(\mathbb{S}_{JR}) \le \deg(\mathbb{S}_R) - 1 \le d-1$.

Recursively, we have the following sequence

$$(\mathbb{S}) \to (\mathbb{S}_J, \mathbb{S}_R) \to (\mathbb{S}_{JJ}, \mathbb{S}_{JR}, \mathbb{S}_{RR}, \mathbb{S}_{RJ}) \to \cdots \tag{20}$$

For a set $\mathbb{S}_{O_1O_2\cdots O_k}$ where O_i is J or R, we have $|\mathbb{S}_{O_1O_2\cdots O_k}| \le l$. We can deduce that $\deg(\mathbb{S}_{O_1O_2\cdots O_k}) \le d-s$ where s is the number of O_i which is J. Therefore, the number of J occurring in the subscript of $\mathbb{S}$ can be $d-1$ at most. As a consequence, in round $n-k$ corresponding to the $(k+1)$-th part of the sequence (20), the number of $\mathbb{S}_i$ is at most $\binom{k}{0}+\binom{k}{1}+\cdots+\binom{k}{d-1}$. Thus, the number of polynomials in round $n-k$ is at most $l(\sum_{i=0}^{d-1}\binom{k}{i})$. It implies that we need at most $l(\sum_{k=0}^{n-1}\sum_{i=0}^{d-1}\binom{k}{i}) = l(\sum_{i=1}^{d}\binom{n}{i})$ polynomial additions in the algorithm. It is easy to prove that in other simpler cases, the times of additions are still bounded by $l(\sum_{i=1}^{d}\binom{n}{i})$ or $O(ln^d)$.

Now let us estimate the complexity of polynomial additions in **MFTriSet**. We can define an operator $\mathcal{I}_k$ as follows: If $\mathrm{cls}(P) = k$, $\mathcal{I}_k(P) = \mathrm{init}(P)$; if $\mathrm{cls}(P) < k$, $\mathcal{I}_k(P) = 0$. It is easy to prove that if we substitute $\mathcal{J}_i$ with $\mathcal{I}_i$ in equation (18) and equation (19) of Section 5.2, any of the two equations will either be unchanged or become itself plus one. Now we use $\mathrm{term}(P)$ to denote the number of monomials occurring in P. Then we have $\mathrm{term}(\mathcal{I}P) + \mathrm{term}(\mathcal{R}P) \le \mathrm{term}(P)$. Similar to the proof of Theorem 5.4, we can prove the following lemma

Lemma 5.9 *Let n be the number of variables and $\mathbb{P}$ the input of Algorithm* **MFTriSet***. Then, for any polynomial T occurring in* **MFTriSet***, we have* $\mathrm{term}(T) \le \sum_{P\in\mathbb{P}}\mathrm{term}(P)+1$. *If $|\mathbb{P}| > n$, then there exist n polynomials $P_1,\ldots,P_n$ in $\mathbb{P}$ such that* $\mathrm{term}(T) \le \mathrm{term}(P_1) + \mathrm{term}(P_2) + \cdots + \mathrm{term}(P_n) + 1$.

Note that the bitsize complexity of computing the sum of P_1 and P_2 is $O(n(\mathrm{term}(P_1) + \mathrm{term}(P_2)))$. Then the complexity of Algorithm **MFTriSet** is $O(ln^{d+1}(\sum_{P\in\mathbb{P}}\mathrm{term}(P)))$. We have proved Theorem 5.8.

6. Experimental Results

We have implemented algorithms **TDCS** and **MFCS** in $\mathbb{R}_2$ with the C language and tested them with a large number of polynomial systems. In order to save storage space, we use the SZDD to store the polynomials in our implementation [33].

For comparison, we also use the Gröbner basis algorithm (F4) in Magma with Degree Reverse Lexicographic order, denoted by **GB**, to solve these polynomial systems. The experiments are done on a PC with a 3.19GHz CPU, 2G memory, and a Linux OS. The running times in the tables are all given in seconds.

6.1 Boolean Matrix Multiplication Problem

For two $n \times n$ Boolean matrices A and B, if $AB = I$, by the linear algebra we can deduce that $BA = I$, where I is the $n \times n$ identity matrix. However, if we want to check the conclusion by reasoning, it will become an extremely difficult problem. This challenge problem was proposed by Stephen Cook in his invited talk at SAT 2004 [11, 12]. The best known result was that the problem of $n = 5$ can be solved by SAT-solvers in about 800-2000 seconds. The problem of $n = 6$ were still unsolved [3].

Now we test our software for this problem by converting the problem into the solving of a Boolean polynomial system. By setting the entries of A and B to be $2n^2$ distinct variables, we can obtain n^2 quadratic polynomials from $AB = I$. Then we compute the Gröbner basis or the zero decomposition of this polynomials, and check wether the polynomials generated by $BA = I$ can be reduced to 0 by the Gröbner basis or by every characteristic set in the zero decomposition. In this way, we can prove the conclusion.

We use the CS method to illustrate the above procedure. Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be the polynomial sets generated by $AB = I$ and $BA = I$ respectively. With the CS method, we have

$$\mathrm{Zero}_q(\mathbb{P}_1) = \cup_i \mathrm{Zero}_q(\mathcal{A}_i)$$

where $\mathcal{A}_i$ are triangular sets. If $\mathrm{prem}(P, \mathcal{A}_i) = 0$ for all possible i and $P \in \mathbb{P}_2$, then we have solved the problem. It is clear that the major difficulty here is to compute the decomposition.

For $n = 4, 5, 6$, the numbers of variables are $32, 50, 72$ respectively. Therefore, computing the Gröbner basis or the zero decomposition of this polynomials will be a hard work. We used **GB** and our **MFCS** algorithm to solve the problem with $n = 4, 5, 6$. The running time given in Table 1 includes solving the equations generated by $AB = I$ and checking the conclusion $BA = I$. Notation • means memory overflow.

	n=4	n=5	n=6
MFCS	0.11	41	196440
GB	2363	•	•

Table 1. Running times for Boolean matrix multiplication problems

6.2 Equations from Stream Ciphers Based on Nonlinear Filter Generators

In this section we generate our equations from stream ciphers based on LFSRs. We first show how these polynomial systems are generated. A linear feedback shift register (LFSR) of length L can be simply considered as a sequence of L numbers $(c_1, c_2, \ldots, c_L)$ from $\mathbb{F}_2$ such that $c_L \neq 0$ [31]. For an **initial state** $S_0 = (s_0, s_1, \ldots, s_{L-1}) \in \mathbb{F}_2^L$, we can use the given LFSR to produce an infinite sequence satisfying

$$s_i = c_1 s_{i-1} + c_2 s_{i-2} + \cdots c_L s_{i-L}, i = L, L+1, \cdots . \tag{21}$$

A key property of an LFSR is that if the related **feedback polynomial** $P(x) = c_L x^L + c_{L-1}x^{L-1} + \cdots + c_1 x - 1$ is primitive, then the sequence (21) has period $2^L - 1$ [31]. The number of non-zero coefficients in P is called the **weight** of P, denoted by w_P.

An often used technique in stream ciphers to enhance the security of an LFSR is to add a **nonlinear filter** to the LFSR. Let $f(x_1, \ldots, x_m)$ be a Boolean polynomial with m variables. We assume that $m \leq L$. Then we can use f and the sequence (21) to generate a new sequence as follows

$$z_t = f(s_{t+k_1}, s_{t+k_2} \ldots, s_{t+k_m}), t = 0, 1, \ldots \tag{22}$$

where $\{k_i\}_{1\leq i\leq m}$ is called the **tapping sequence**. A combination of an LFSR and a nonlinear polynomial f is called a **nonlinear filter generator** (NFG).

The filter functions used in this paper are due to Canteaut and Filiol [7]:

- CanFil 1, $x_1x_2x_3 + x_1x_4 + x_2x_5 + x_3$
- CanFil 2, $x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 + x_1x_4 + x_2x_5 + x_3 + x_4 + x_5$
- CanFil 3, $x_2x_3x_4x_5 + x_1x_2x_3 + x_2x_4 + x_3x_5 + x_4 + x_5$
- CanFil 4, $x_1x_2x_3 + x_1x_4x_5 + x_2x_3 + x_1$
- CanFil 5, $x_2x_3x_4x_5 + x_2x_3 + x_1$
- CanFil 6, $x_1x_2x_3x_5 + x_2x_3 + x_4$
- CanFil 7, $x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_5 + x_1 + x_2 + x_3$
- CanFil 8, $x_1x_2x_3 + x_2x_3x_6 + x_1x_2 + x_3x_4 + x_5x_6 + x_4 + x_5$
- CanFil 9, $x_2x_4x_5x_7 + x_2x_5x_6x_7 + x_3x_4x_6x_7 + x_1x_2x_4x_7 + x_1x_3x_4x_7 + x_1x_3x_6x_7 + x_1x_4x_5x_7 + x_1x_2x_5x_7 + x_1x_2x_6x_7 + x_1x_4x_6x_7 + x_3x_4x_5x_7 + x_2x_4x_6x_7 + x_3x_5x_6x_7 + x_1x_3x_5x_7 + x_1x_2x_3x_7 + x_3x_4x_5 + x_3x_4x_7 + x_3x_6x_7 + x_5x_6x_7 + x_2x_6x_7 + x_1x_4x_6 + x_1x_5x_7 + x_2x_4x_5 + x_2x_3x_7 + x_1x_2x_7 + x_1x_4x_5 + x_6x_7 + x_4x_6 + x_4x_7 + x_5x_7 + x_2x_5 + x_3x_4 + x_3x_5 + x_1x_4 + x_2x_7 + x_6 + x_5 + x_2 + x_1$
- CanFil 10, $x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_5 + x_6x_7 + x_3 + x_2 + x_1$.

In the experiments, we use our algorithms to find $S_0 = (s_0, s_1, \ldots, s_{L-1})$ by solving the following equations for given c_i, z_i, and f

$$z_t = f(s_{t+k_1}, s_{t+k_2} \ldots, s_{t+k_m}), t = 0, 1, \ldots, k \tag{23}$$

where k is a positive integer, s_i satisfy (21), and $k_1, \ldots, k_m$ is a tapping sequence.

We compare four different algorithms for solving these equations. Two of them are the **MFCS** and **GB**. Faugère and Perret suggested to us that an incremental version of the Gröbner basis algorithm is faster than **GB** for the equations generated by the LFSR[1]. Therefore, we also compare the incremental Gröbner basis algorithm and the incremental **TDCS**, denoted **IGB** and **ITDCS** respectively. Note that the F5 method [17] and the CS method presented in [30] also use the incremental technique.

[1] By an incremental **GB** for a polynomial set $\{P_1, \ldots, P_s\}$, we mean to compute the Gröbner basis G_1 of $\{P_1\}$ first and then to compute the Gröbner basis G_2 of $G_1 \cup \{P_2\}$, etc.

We did three sets of experiments with increasing difficulties. The test problems are similar to those in [8] but are more difficult. We also compare our method with one of the benchmark implementations of the Gröbner basis method on the same computer, which are not given in [8].

In the first set of experiments, we choose a simple tapping sequence $\{0, 1, 2, 3, 4, 5, 6\}$ and feedback polynomials with small weights. The results are given in Table 2, where L is the number of variables, k is the number of equations (see (23)). k is the smallest number such that the system has a unique solution, w_P is the weight of the feedback polynomial P, and $\bullet$ means memory overflow.

In the second set of experiments, we generate more difficult equations in the cases of $L = 40$ and $k = 60$ by changing the weight of the feedback polynomial w_P to 11. The results are given in Table 3.

In the third set of experiments, we generate more dense polynomial systems by changing the tapping sequence. The results are given in Table 4, in which $L = 40$, $w = 7$, $k = 55$, and the tapping sequence is $\{0, 6, 11, 18, 25, 31, 37\}$. And $*$ means we have computed over 2 hours and did not obtain the solutions.

From the experiments, we have the following observations.

- From Table 2, we can see that for these "simple" examples, **ITDCS** is the fastest method. **IGB** and **MFCS** are also very efficient with **MFCS** better than **IGB** in most cases. **GB** tends to generate large polynomials and causes memory overflow.
- From Table 3, we can see that for "moderately difficult" polynomial systems, **ITDCS** is still the fastest method. Now, **IGB** performs better than **MFCS**.
- From Table 4, we can see that for the "most difficult" polynomial systems, **MFCS** is the only algorithm that can find the solutions on our computer. **IGB** and **GB** quickly use all the memory and cause memory overflow. **ITDCS** has been run for two hours without giving a result. The reason is that, in this case, **ITDCS** and **IGB** need to deal with some high degree and dense polynomials. On the other hand, due to Theorems 5.4 and 5.8, the polynomials occurring in Algorithm **MFCS** are much smaller.

In summary, Algorithm **MFCS** seems to be the most efficient and stable approach to deal with these kinds of polynomial systems. The main reason is that the size of the polynomials in this algorithm is effectively controlled due to Theorems 5.4 and 5.8. To use SZDD [33] to represent polynomials is another key factor in memory saving. Note that SZDD suits the CS method very well. The CS method will generate a large number of components and the polynomial sets representing different components differ only for a very few number of polynomials due to the way of generating new components (see Step 2.6.3 of Algorithm 4.3). Then different polynomial sets will share memory for their common polynomials, and as a consequence, the total memory consumption is well contained.

For Algorithm **MFCS**, the bottle neck problem is how to control the number of components (that is, the number of polynomial sets in $\mathbb{P}^*$ in the output of Algorithm **MFTriSet**). Theoretically, this number is exponential in the worst case. Practically, this number could

Filters	L(w_f)=	40 (5)	60 (3)	81 (3)	100 (3)	128 (5)
CanFil1	**MFCS**	0.10	0.02	0.07	0.37	0.49
	ITDCS	0.10	0.04	0.05	0.21	0.37
	IGB	0.42	0.99	2.29	3.26	8.32
	GB	0.91	0.43	8.12	3.61	1997.2
	k	52	114	154	140	230
CanFil2	**MFCS**	0.17	0.03	0.07	0.59	1.11
	ITDCS	0.04	0.02	0.06	0.19	0.53
	IGB	0.43	0.65	1.61	3.17	7.13
	GB	0.92	30.65	0.02	55.09	●
	k	44	72	138	140	217
CanFil3	**MFCS**	0.17	0.03	0.07	0.59	1.11
	ITDCS	0.14	0.03	0.23	1.10	0.72
	IGB	0.16	0.96	2.51	6.04	16.08
	GB	178.57	1.68	●	●	●
	k	64	114	162	120	128
CanFil4	**MFCS**	0.09	0.05	0.07	0.83	2.70
	ITDCS	0.14	0.09	0.09	2.91	2.01
	IGB	0.17	0.89	1.99	2.13	10.26
	GB	0.65	2.24	0.39	●	●
	k	60	168	154	150	180
CanFil5	**MFCS**	0.03	0.01	0.03	0.08	0.12
	ITDCS	0.04	0.05	0.11	0.18	0.59
	IGB	0.14	0.37	0.80	1.59	3.46
	GB	0.10	0.06	0.10	0.50	0.85
	k	40	60	81	100	128
CanFil6	**MFCS**	0.05	0.04	0.08	0.11	0.35
	ITDCS	0.09	0.04	0.10	0.29	1.07
	IGB	0.08	0.35	0.80	1.70	5.28
	GB	0.24	0.09	0.01	0.65	●
	k	52	108	146	160	230
CanFil7	**MFCS**	0.05	0.02	0.08	0.38	0.70
	ITDCS	0.03	0.03	0.08	0.24	0.42
	IGB	0.10	0.81	1.86	3.32	9.78
	GB	0.27	0.40	0.01	831.89	●
	k	40	120	154	150	218
CanFil8	**MFCS**	0.32	0.08	0.21	0.61	1.31
	ITDCS	0.09	0.06	0.14	0.25	0.66
	IGB	0.13	0.30	1.26	2.09	6.11
	GB	0.88	0.56	92.51	20.03	●
	k	44	60	154	140	218
CanFil9	**MFCS**	2.94	0.30	0.64	0.79	15.31
	ITDCS	0.45	0.06	0.24	1.22	1.28
	IGB	4.39	5.13	13.15	17.78	47.62
	GB	●	90.49	●	●	●
	k	48	102	113	110	218
CanFil10	**MFCS**	0.39	0.06	0.12	1.40	3.43
	ITDCS	0.12	0.04	0.12	0.57	0.49
	IGB	4.48	28.16	50.87	63.63	100.39
	GB	28.72	2.21	492.16	●	●
	k	44	90	122	140	205

Table 2. Examples with simple feedback polynomials and tapping sequences

Filter	**ITDCS**	**MFCS**	**IGB**	**GB**
Canfil1	0.78	3.56	0.89	55.73
Canfil2	0.47	2.72	0.66	49.33
Canfil3	1.01	10.81	3.16	•
Canfil4	0.99	2.88	0.62	26.10
Canfil5	0.58	3.73	3.00	•
Canfil6	0.58	3.18	2.81	•
Canfil7	0.16	0.50	0.27	16.64
Canfil8	0.26	17.05	0.34	33.35
Canfil9	6.83	73.18	8.54	•
Canfil10	0.70	4.12	4.87	•

Table 3. Examples with larger feedback polynomials

Filter	**MFCS**	**ITDCS**	**IGB**
Canfil1	145.04	*	• after 10m
Canfil2	241.05	*	• after 8m
Canfil3	200.40	*	• after 28m
Canfil4	17.44	*	• after 60m
Canfil5	54.86	*	• after 4m
Canfil6	135.26	*	• after 6m
Canfil7	19.42	*	• after 37m
Canfil8	5132.84	*	• after 60m

Table 4. Examples with larger feedback polynomials and nontrivial tapping sequences

	Canfil1	Canfil2	Canfil3	Canfil4	Canfil5	Canfil6	Canfil7	Canfil8
N_C	13749	23881	7251	1657	1086	3331	1551	180710
$R \approx$	2^{-26}	2^{-25}	2^{-27}	2^{-29}	2^{-30}	2^{-28}	2^{-29}	2^{-19}

Table 5. The number of components for the examples in Table 4

also be very large. But, comparing to the number 2^n of exhaust search, the number of components generated in **MFTriSet** is still very small. In Table 5, we give the numbers of components for each example in Table 4. In this table, N_C is the number of components and $R = \frac{N_C}{2^n}$ could be considered as a measure of effectiveness of Algorithm **MFTriSet**. We can see that R is very small for all examples.

6.3 Attack on Bivium-A

Bivium is a simple version of the eStream stream cipher candidate Trivium [44] . It is built on the same design principles of Trivium. The intention is to reduce the complexity of Trivum, and to extend the attacks on Bivium to Trivium. Bivium has two versions Bivium-A and Bivium-B. Here we focus on attacking Bivium-A. There have been several successful attacks on Bivium-A, and we want to show that our algorithm is comparable with these algorithms.

The Bivium-A is given by the following pseudo-code:

```
for $i = 1$ to $N$ do
    $t_1 \leftarrow s_{66} + s_{93}$
    $t_2 \leftarrow s_{162} + s_{177}$
    $z_i \leftarrow t_2$
    $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
    $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{69}$
    $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots, s_{92})$
    $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
```

We want to recover the initial state $(s_1, \ldots, s_{177})$ from the given N output bits $(z_1, \ldots, z_N)$. Note that the degree of the equations will increase after several clocks. In order to avoid this problem, we can introduce two new variables and two equations for each clock:

$$s_{178} = s_{66} + s_{93} + s_{91} \cdot s_{92} + s_{171} \tag{24}$$

$$s_{179} = s_{162} + s_{177} + s_{175} \cdot s_{176} + s_{69} \tag{25}$$

Then we can obtain a boolean polynomial system with $2N + 177$ variables and $3N$ equations.

The results of the successful attacks on Bivium-A [32, 36, 37][2)] is given in the following table.

Method	Graph for sparse system	SatSolver	Gröbner Basis
Time	"about a day"	21 sec	400 sec
Output Bits	177	177	2000

Table 6. The known results for Bivium-A

In our experiments, we use the algorithm **MFCS** and the equations are generated by adding two new variables for each clock. We run **MFCS** on a sample of 100 different random initial states. We observed that the different initial keys make a great difference to the results. For every initial state, we can find a number M. When the number of output bits N is not less than M, the equations can be solved within one minute. When N becomes much bigger, the running time will increase slowly. However, if N is less than M, the running time will be much longer than one minute. From our experiment results, the value of M is from 200 to 700. In our experiments, we set $N = 700$.

The average time for solving the problem by **MFCS** with 700 output bits is 49.3 seconds. We also tried to use **GB** to solve the same sample by the same computer. The equations are also generated by adding two variables for each clock. In order to solve the equations, we need 1700 output bits. If the output is less than 1700 bits, the memory will be exhausted. For $N = 1700$, the average time for solving the problem by **GB** is 303.3 seconds. If we set $N = 2000$ as in [37], the average time is 521.6 seconds. From the results, we can see that our algorithm is comparable with the known successful algorithms in this problem.

[2)] In [37], they give four different results by solving in different ways. Here we only list the result by adding new variables but without guessing any variables.

7. Conclusions

In this paper, we present two algorithms to solve nonlinear equation systems in finite fields based on the idea of characteristic set. Due to the special property of finite fields, the given algorithms have better properties than the general characteristic set method. In particular, we obtain an explicit formula for the number of solutions of an equation system, and give the bitsize complexity of the algorithm for Boolean polynomials. We also prove that the size of the polynomials in **MFCS** can be effectively controlled, which allows us to avoid the expression swell problem effectively.

We test our methods by solving polynomial systems generated by the Boolean matrix problem, stream cipher Bivium-A and stream ciphers based on nonlinear filter generators. All these equations have block triangular structure. Extensive experiments show that our methods are efficient for solving this kind of equations and Algorithm **MFCS** seems to be the most efficient and stable approach for these problems.

References

[1] Aubry, P., Lazard, D., Maza, M.M., On the Theory of Triangular Sets, *Journal of Symbolic Computation*, 25, 105-124, 1999.

[2] Bardet, M., Faugere, J.C., B.Salvy, Complexity of Gröbner Basis Computation for Semi-regular Overdetermined sequences over F2 with Solutions in F2, *INRIA report RR-5049*, 2003.

[3] Biere, A., Linear Algebra, Boolean Rings and Resolution, *ACA'08*, July, Austria, 2008.

[4] Boulier, F., Lazard, D., Ollivier, F., Petitiot, M., Representation for the Radical of a Finitely Generated Differential Ideal, *Proc. of ISSAC'95*, 158-166, ACM Press, New York, 1995.

[5] Bouziane, D., Kandri Rody, A., Maârouf, H., Unmixed-dimensional Decomposition of a Finitely Generated Perfect Differential Ideal, *Journal of Symbolic Computation*, **31**, 631-649, 2001.

[6] Brickenstein, M. and Dreyer, A., PolyBoRi: A Framework for Gröbner Basis Computations with Boolean Polynomials, *MEGA 2007*, July, 2007, Austria.

[7] Canteaut, A. and Filiol, E., Ciphertext only Reconstruction of Stream Ciphers Based on Combination Generators, *Fast Software Encryption*, LNCS 1978, 165-180, Springer, 2000.

[8] Chai, F., Gao, X.S., Yuan C., A Characteristic Set Method for Solving Boolean Equations and Applications in Cryptanalysis of Stream Ciphers, *Journal of Systems Science and Complexity*, 21(2), 191-208, 2008.

[9] Chou, S.C., *Mechanical Geometry Theorem Proving,* D. Reidel, Dordrecht, 1988.

[10] Chou, S.C. and Gao, X.S., Ritt-Wu's Decomposition Algorithm and Geometry Theorem Proving, *Proc. of CADE-10*, LNAI 449, 207-220, Springer, 1990.

[11] Cook, S., From Satisfiability to Proof Complexity and Bounded Arithmetic, *SAT 2004*, Invited Talk, 10-13 May, 2004, Vancouver, Canada.

[12] Cook, S. and Nguyen, P., *Logical Foundations of Proof Complexity*, Cambridge University Press, 2010.

[13] Coron, J.S. and de Weger, B., ECRYPT: Hardness of the Main Computational Problems Used in Cryptography, *European Network of Excellence in Cryptology*, 2007.

[14] Courtois, N., Klimov, A., Patarin, J., and Shamir, A., Efficient Algorithms for Solving Overdetermined Systems of Multivariate Polynomial Equations, *EUROCRYPT 2000*, LNCS 1807, 392-407, 2000.

[15] Dahan, X., Maza, M.M., Schost, E., Wu, W., Xie, Y., Lifting Techniques for Triangular Decompositions, *Proc. ISSAC'05*, 108-115, ACM Press, New York, 2005.

[16] Faugère, J.C., A New Efficient Algorithm for Computing Gröbner Bases (F4), *Journal of Pure and Applied Algebra*, 139(1–3), 61–88, 1999.

[17] Faugère, J.C., A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5), *Proc. ISSAC 2002*, 75-83, 2002.

[18] Faugère, J.C., and Joux, A., Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In Dan Boneh, editor, Advances in Cryptology - CRYPTO 2003, volume 2729 of LNCS, pages 44–60. Springer, 2003.

[19] Faugère, J.C. and Ars, G., An Algebraic Cryptanalysis of Nonlinear Filter Generators Using Gröbner Bases, TR No. 4739, INRIA, 2003.

[20] Gallo, G. and Mishra, B., Efficient Algorithms and Bounds for Wu-Ritt Characteristic Sets, in *Effective Methods in Algebraic Geometry*, 119-142, Birkhauser, Boston, 1991.

[21] Gao, X.S., Luo, L, Yuan, C., A Characteristic Set Method for Difference Polynomial Systems, *Journal of Symbolic Computation*, 44(3), 242-260, 2009.

[22] Gerdt, V. and Zinin, M., A Pommaret Division Algorithm for Computing Gröbner Bases in Boolean Rings, *Proc. ISSAC 2008*, ACM Press, 2008.

[23] Hubert, E., Factorization-free Decomposition Algorithms in Differential Algebra, *Journal of Symbolic Computation*, 29, 641-662, 2000.

[24] Kalkbrener, M., A Generalized Euclidean Algorithm for Computing Triangular Representations of Algebraic Varieties, *Journal of Symbolic Computation*, 15, 143-167, 1993.

[25] Kapur, D. and P. Narendran, An Equational Approach to Theorem Proving in First-Order Predicate Calculus, Proc. *IJCAI-8)*, Los Angeles, Calif., 1985, 1146-1153.

[26] Kapur, D. and Wan, H.K., Refutational Proofs of Geometry Theorems via Characteristic Sets, *Proc. ISSAC'90*, 277-284, ACM Press New York, 1990.

[27] Lazard, D., Gröbner bases, Gaussian Elimination and Resolution of Systems of Algebraic Equations, LNCS 162, Springer Berlin, 1983.

[28] Lazard, D., A New Method for Solving Algebraic Systems of Positive Dimension, *Discrete Appl. Math.*, **33**, 147-160, 1991.

[29] Lin, D. and Liu, Z., Some Results on Theorem Proving in Geometry over Finite Fields, *Proc. ISSAC'93*, 292-300, ACM Press, New York, 1993.

[30] Maza, M.M., On Triangular Decompositions of Algebraic Varieties. Technical Report 4/99, NAG, UK, Presented at the MEGA-2000 Conference, Bath, UK.

[31] Menezes, A., van Ooschot, P., Vanstone, S., *Hanndbook of Applied Cryptography*, CRC Press, 1996.

[32] Mcdonald, C., Charnes, C. and Pieprzyk, J., Attacking Bivium with MiniSat. http://eprint.iacr.org/2007/129,2007.

[33] Minto, S., Zero-Sppressed BDDs for Set Manipulation in Combinatorial Problems, *Proc. ACM/IEEE Design Automation*, 272-277, ACM Press, 1993.

[34] Möller, H.M., On Decomposing Systems of Polynomial Equations with Finitely Many Solutions, *J. AAECC*, 4, 217–230, 1993.

[35] Patarin, J., Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. Extended version, 1996.

[36] Cryptanalytic results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, 2006. http://www.ecrypt.eu.org/stream

[37] Simonetti, I., Faugère, J.C. and Perret, L., Algebraic attack against trivium. In First International Conference on Symbolic Computation and Cryptography, SCC 08, pages 95–102.LMIB, Beijing, China, April 2008.

[38] Sato, Y. and Inoue, S., On the Construction of Comprehensive Boolean Gröbner Bases. *Proc. ASCM 2005*, 145-148, 2005.

[39] Smale, S, Mathematical Problems for The Next Century, *Math. Intelligencer*, 20, 7-15, 1998.

[40] Wang, D. An Elimination Method for Polynomial Systems. *Journal of Symbolic Computation*, 16, 83-114, 1993.

[41] Wu, W.T., Basic Principles of Mechanical Theorem-proving in Elementary Geometries, *Journal Automated Reasoning*, 2, 221-252, 1986.

[42] Wu, W.T., *Mathematics Machenization*, Sience Press/Kluwer, Beijing, 2001.

[43] Yang, L., Zhang, J.Z., and Hou, X.R., *Non-linear Algebraic Equations and Automated Theorem Proving* (in Chinese), ShangHai Science and Education Pub., Shanghai, 1996.

[44] eSTREAM: ECRYPT Stream Cipher Project http://www.ecrypt.eu.org/stream/