A Characteristic Set Method for Solving Boolean Equations and Applications in Cryptanalysis of Stream Ciphers[1]

Fengjuan Chai, Xiao-Shan Gao[2], and Chunming Yuan
Key Laboratory of Mathematics Mechanization
Institute of Systems Science, AMSS, Academia Sinica, Beijing, 100080, China

Abstract. We present a characteristic set method for solving Boolean equations, which is more efficient and has better properties than the general characteristic set method. In particular, we give a disjoint and monic zero decomposition algorithm for the zero set of a Boolean equation system and an explicit formula for the number of solutions of a Boolean equation system. We also prove that a characteristic set can be computed with a polynomial number of multiplications of Boolean polynomials in terms of the number of variables. As experiments, we use our method to solve equations from cryptanalysis of a class of stream ciphers based on nonlinear filter generators. Extensive experiments show that the method is quite effective.

Keywords. Characteristic set method, Boolean equation, finite field $\mathbb{F}_2$, cryptanalysis, stream ciphers.

1. Introduction

The characteristic set (CS) method is an effective tool for studying systems of polynomial equations, algebraic differential equations, and algebraic difference equations [28, 23, 12]. The idea of the method is to reduce an equation system in general form to equation systems in a special "triangular form", also called ascending chains. The zero-set of any finitely generated equations can be decomposed into the union of the zero-sets of ascending chains. As a consequence, solving an equation system can be reduced to solving cascaded univariate equations.

Boolean equation solving is a fundamental problem in computer science and has many applications such as hardware design and verification [1, 19], cryptanalysis of ciphers [6, 7, 10], and SAT problem solving [14]. The problem of deciding whether a Boolean equation system has a solution is NP-complete [26]. There exist many approaches to solve Boolean equations such as the classic algebraic methods [24], the

[1)]Partially supported by a National Key Basic Research Project of China (2004CB318000).
[2)]Corresponding author. Email: xgao@mmrc.iss.ac.cn

logic methods such as the Davis-Putnam procedure [8], the graph based methods such as the BDD method [2], and the Gröbner basis and XL methods [1, 6, 10, 25].

In this paper, we propose two CS methods to solve Boolean equations, which is equivalent to polynomial equation solving in the finite field $\mathbb{F}_2$. By taking into account of the special property of $\mathbb{F}_2$, our proposed methods are much more efficient and have better properties than the general CS method.

The first major improvement is that we could decompose the zero set of a Boolean equation system as the disjoint union of the zero sets of ascending chains consisting of monic polynomials. As a consequence, we could give an explicit formula for the number of solutions of the equation system.

The well-ordering principle is a basic step of the CS method, which can be used to compute a so called Wu-CS for an equation system. If the Wu-CS satisfies certain properties, it provides at least one solution to the original equation system. The second improvement is that we design well-ordering principles which can be executed in n steps and use a polynomial number of polynomial multiplications, where n is the number of variables. We also design an algorithm, where the degrees of the polynomials occurring in the algorithm do not increase. This allows us to control the size of the polynomials effectively. Since Boolean polynomial equation solving is NP-complete, there exist no universally fast algorithms for this problem. The philosophy behind our algorithms is that we will compute each Wu-CS or branch effectively by controlling the size of the polynomials and reducing the total number of branches using various strategies.

The general CS methods do not have the properties mentioned above [11]. The work [17, 18, 9] considered CS methods for polynomials with coefficients in a field of a positive characteristic. These algorithms also do not have the above mentioned properties.

We implement our algorithm with the C language. Besides the concept of ascending chain, we also used the concept of Wu chain defined in [29] and the concept of weak chain defined in [5] in our program. In order to save space, we use SZDD [21] to represent Boolean polynomials. Experiments show that this could speed up the program significantly.

As experiments, we use our methods to solve equations from cryptanalysis of stream ciphers based on nonlinear filter generators. Extensive experiments have been done for equation systems with variables ranging from 40 to 128. Comparisons with the Gröbner basis method are given. Experiments show that our algorithms provide an effective tool for solving equations over $\mathbb{F}_2$.

The rest of this paper is organized as follows. In Section 2, we introduce the notations and give some preliminary results. In Section 3, we present the CS methods. In Section 4, we present a direct algorithm to decompose the zero set of a polynomial system into the zero sets of monic ascending chains. In Section 5, we discuss the issues in the implementation of the algorithms. In Section 6, our methods are used to solve equations from cryptanalysis of stream ciphers based on nonlinear filter generators. In Section 7, conclusions are given.

2. Notations and Preliminary Results

Let $\mathbb{F}_2$ be the field consisting of 0 and 1. We will consider the problem of equation solving over $\mathbb{F}_2$. Let $\mathbb{X} = \{x_1, \ldots, x_n\}$ be a set of indeterminants and

$$\mathbb{R}_2 = \mathbb{F}_2[\mathbb{X}]/(\mathbb{H})$$

where $\mathbb{H} = \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$. Then $\mathbb{R}_2$ is a Boolean ring[1)]. Note that $\mathbb{R}_2$ has zero divisors. For instance, x_i and $x_i + 1$ are zero divisors. An element P in $\mathbb{R}_2$ is called a *Boolean polynomial*, or simply a polynomial, and has the following *canonical representation*:

$$P = M_s + \cdots + M_0$$

where M_i is a product of several distinct variables.

Let $\mathbb{P}$ be a set of polynomials in $\mathbb{R}_2$. We use $\overline{\text{Zero}}(\mathbb{P})$ to denote the common zeros of the polynomials in $\mathbb{P}$ in the affine space $\mathbb{F}_2^n$, that is,

$$\overline{\text{Zero}}(\mathbb{P}) = \{(a_1, \ldots, a_n), a_i \in \mathbb{F}_2, s.t., \forall P \in \mathbb{P}, P(a_1, \ldots, a_n) = 0\}.$$

Let D be a polynomial in $\mathbb{R}_2$. We define a *quasi variety* to be

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

For a polynomial set $\mathbb{P}$, we use $(\mathbb{P})$ to denote the ideal generated by the polynomials in $\mathbb{P}$. The following are well-known results.

Lemma 2.1 *Let* I *be a polynomial ideal in* $\mathbb{R}_2$.

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

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

(3) *Let* $P \in \mathbb{R}_2$ *and* s *a positive integer. Then* $P^s = P$.

As a consequence of Lemma 2.1(2), we have

Corollary 2.2 *Let* $P \in \mathbb{R}_2 \setminus \mathbb{F}_2$. *Then* $\overline{\text{Zero}}(P) \neq \emptyset$.

Lemma 2.3 *Let* U, V, *and* D *be polynomials and* $\mathbb{P}$ *a polynomial set in* $\mathbb{R}_2$. *We have*

$$(UV + 1) = (\{U + 1, V + 1\}). \tag{1}$$
$$(UV + U + V) = (\{U, V\}). \tag{2}$$
$$\overline{\text{Zero}}(\emptyset/D) = \overline{\text{Zero}}(D + 1). \tag{3}$$
$$\overline{\text{Zero}}(\mathbb{P}) = \overline{\text{Zero}}(\mathbb{P} \cup \{U\}) \cup \overline{\text{Zero}}(\mathbb{P} \cup \{U + 1\}). \tag{4}$$

Proof: We can prove (1) as follows: $(UV + 1) = (UV + 1, (U + 1)(UV + 1)) = (UV+1, U+1) = (V+1, U+1)$. Equation (2) can be proved similarly: $(UV+U+V) = (UV + U + V, (U + 1)(UV + U + V)) = (UV + U + V, UV + V) = (U, V)$. For any element $\alpha \in \mathbb{F}_2^n$, $D(\alpha) \neq 0$ implies $D(\alpha) = 1$. This proves (3). Note that $U(U+1) \equiv 0$. Then (4) is obviously true. □

[1)] A ring is called a Boolean ring if all its elements are idempotent. See page 31 of [24].

3. A Characteristic Set Method in $\mathbb{R}_2$

We will give a CS method to solve Boolean polynomial equations, which is more efficient and has better properties than the general CS method.

3.1 Triangular Sets and Chains

Let $P \in \mathbb{R}_2$. The *class* of P, denoted by $\mathrm{cls}(P)$, is the largest c such that x_c occurs in P. If $P \in \mathbb{F}_2$, we set $\mathrm{cls}(P) = 0$. If $\mathrm{cls}(P) = c > 0$, we call x_c the *leading variable* of P, denoted as $\mathrm{lvar}(P)$. The leading coefficient of P as a univariate polynomial in $\mathrm{lvar}(P)$ is called the *initial* of P, and is denoted as $\mathrm{init}(P)$.

A sequence of nonzero polynomials

$$\mathcal{A}: \quad A_1, A_2, \ldots, A_r \tag{5}$$

is a *triangular set* if either $r = 1$ and $A_1 = 1$ or $0 < \mathrm{cls}(A_1) < \cdots < \mathrm{cls}(A_r)$. For a triangular set $\mathcal{A}$ of form (5), let $\mathbf{I}_\mathcal{A} = \prod_{i=1}^{r} \mathrm{init}(A_i)$.

Let $P = Ix_c + U$ with $I = \mathrm{init}(P)$ and class c. For $Q \in \mathbb{R}_2$, write Q as a polynomial in x_c: $Q = I_1 x_c + U_1$. If $I_1 \neq 0$, the *pseudo-remainder* of Q wrt P is defined as

$$\mathrm{prem}(Q, P) = IQ + I_1 P = IU_1 + I_1 U.$$

If $I_1 = 0$, we define $\mathrm{prem}(Q, P) = Q$. If $P = 1$, define $\mathrm{prem}(Q, P) = 0$. For a triangular set $\mathcal{A}$ of form (5), the *pseudo-remainder* of Q wrt $\mathcal{A}$ is defined as

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

Let $R = \mathrm{prem}(Q, \mathcal{A})$. By Lemma 2.1(3), we have

$$JQ = \sum_i Q_i A_i + R \tag{6}$$

where J is a factor of $\mathbf{I}_\mathcal{A}$ and Q_i are 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 $\mathrm{prem}(\mathbb{P}, \mathcal{A})$ to denote the set of nonzero $\mathrm{prem}(P, \mathcal{A})$ for $P \in \mathbb{P}$.

A polynomial Q is *reduced* wrt $P \neq 0$ if $\mathrm{cls}(P) = c > 0$ and x_c does not occur in Q. A polynomial Q is *reduced* wrt a triangular set $\mathcal{A}$ if P is reduced wrt all the polynomials in $\mathcal{A}$. It is clear that $\mathrm{prem}(P, \mathcal{A})$ is reduced wrt $\mathcal{A}$ for any polynomial P.

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

$$\mathrm{sat}(\mathcal{A}) = \{P \in \mathbb{R}_2 | \, \mathbf{I}_\mathcal{A} P \in (\mathcal{A})\}.$$

The saturation ideal in $\mathbb{R}_2$ is very simple. We have

Lemma 3.1 *For a triangular set $\mathcal{A} = A_1, \ldots, A_p$, $\mathrm{sat}(\mathcal{A}) = (A_1, \ldots, A_p, \mathbf{I}_\mathcal{A} + 1)$.*

Proof: Denote $I = (A_1, \ldots, A_p, \mathbf{I}_{\mathcal{A}} + 1)$. If $P \in \mathrm{sat}(\mathcal{A})$, then there exist polynomials B_i such that

$$\mathbf{I}_{\mathcal{A}} P = \sum_i B_i A_i.$$

Let $A_0 = \mathbf{I}_{\mathcal{A}} + 1$ and substitute $\mathbf{I}_{\mathcal{A}} = A_0 + 1$ into the above equation, we have $P = \sum_i B_i A_i + P A_0 \in I$. We prove one side of the equation. For the other side of the equation, let $P \in I$. Then there exist polynomials C_i such that

$$P = \sum_i C_i A_i + C_0 A_0.$$

Multiply $\mathbf{I}_{\mathcal{A}}$ to both sides of the above equation and note that $\mathbf{I}_{\mathcal{A}}(\mathbf{I}_{\mathcal{A}} + 1) = 0$, we have $\mathbf{I}_{\mathcal{A}} P = \sum_i \mathbf{I}_{\mathcal{A}} C_i A_i$. Then $P \in \mathrm{sat}(\mathcal{A})$. This proves the lemma. □

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

$$\mathcal{A}: \quad A_1 = x_{c_1} + U_1(\mathbb{U}), \cdots, A_p = x_{c_p} + U_p(\mathbb{U}) \tag{7}$$

where $\mathbb{U} = \{x_i | i \neq c_j, j = 1, \ldots, p\}$ is called the *parameter set* of $\mathcal{A}$. Let $q = |\mathbb{U}|$. Then, $p + q = n$. The *dimension* of $\mathcal{A}$ is defined to be $\dim(\mathcal{A}) = q = n - |\mathcal{A}|$. We have

Lemma 3.2 *Let $\mathcal{A}$ be a monic triangular set. Then $|\overline{\mathrm{Zero}}(\mathcal{A})| = 2^{\dim(\mathcal{A})}$.*

Proof: The dimension of $\mathcal{A}$ is the number of parameters of $\mathcal{A}$, that is, $\dim(\mathcal{A}) = |\mathbb{U}|$. For any $x_i \in \mathbb{U}$, we assign values 0 and 1 to x_i. Then, there are $2^{\dim(\mathcal{A})}$ parametric values for $\mathbb{U}$. For each of these parametric values, $\mathcal{A} = 0$ has exactly one solution since $\mathcal{A}$ is monic. This proves the lemma. □

A triangular set $\mathcal{A}$ of form (5) is called an *ascending chain*, or simply a chain, if A_j is reduced wrt A_i for $i < j$. A chain $\mathcal{A}$ is called *conflict* if $\mathbf{I}_{\mathcal{A}} = 0$.

Lemma 3.3 *Let $\mathcal{A}$ be a non-conflict chain. Then $\overline{\mathrm{Zero}}(\mathcal{A}/\mathbf{I}_{\mathcal{A}}) \neq \emptyset$.*

Proof: Let $\mathcal{A} = A_1, \ldots, A_p$ and $A_i = I_i x_{c_i} + U_i$ where $I_i = \mathrm{init}(A_i)$. Since $\mathcal{A}$ is a chain, $\mathbf{I}_{\mathcal{A}}$ is reduced wrt $\mathcal{A}$ and does not contain $x_{c_i}, i = 1, \ldots, p$. Let $\mathbb{U} = \mathbb{X} \setminus \{x_{c_1}, \ldots, x_{c_p}\}$. Since $\mathbf{I}_{\mathcal{A}} \neq 0$, by Corollary 2.2, we can select a value η for $\mathbb{U}$ such that $\mathbf{I}_{\mathcal{A}}(\eta) \neq 0$. Then $I_i(\eta) \neq 0$ and we can solve x_{c_i} from $A_i = I_i(\eta) x_{c_i} + U_i(\eta) = 0$. We thus find an element in $\overline{\mathrm{Zero}}(\mathcal{A}/\mathbf{I}_{\mathcal{A}})$. □

3.2 Well-Ordering Principles

Let $\mathcal{A}: \ A_1, \ldots, A_r$ and $\mathcal{B}: \ B_1, , \ldots, B_s$ be two triangular sets. $\mathcal{A}$ is said to be of *lower ordering* than $\mathcal{B}$, denoted as $\mathcal{A} \prec \mathcal{B}$, if either there is a k such that $\mathrm{cls}(A_1) = \mathrm{cls}(B_1), \ldots, \mathrm{cls}(A_{k-1}) = \mathrm{cls}(B_{k-1})$, while $\mathrm{cls}(A_k) < \mathrm{cls}(B_k)$; or $r > s$ and $\mathrm{cls}(A_1) = \mathrm{cls}(B_1), \ldots, \mathrm{cls}(A_r) = \mathrm{cls}(B_r)$. We have the following basic property for triangular sets.

Lemma 3.4 *Let $\mathcal{A}_1 \succ \mathcal{A}_2 \succ \cdots \succ \mathcal{A}_m$ be a strictly decreasing sequence of triangular sets in $\mathbb{R}_2$. Then $m \leq 2^n$.*

Proof: Note that a polynomial P and $\mathrm{lvar}(P)$ have the same ordering. Since we only consider the ordering of the triangular sets, we may assume that polynomials in the triangular sets are variables. We call the class of the first polynomial in a triangular set to be the *class* of that triangular set. We will construct the maximal triangular set with class c. The triangular set with the highest ordering is $\mathcal{C}_1 = x_n$. The next two triangular sets are $\mathcal{C}_2 = x_{n-1}, \mathcal{C}_3 = x_{n-1}, x_n$. Following these triangular sets are the triangular sets with x_{n-2} as the first polynomial: $\mathcal{C}_4 = x_{n-2}, \mathcal{C}_5 = x_{n-2}, x_n, \mathcal{C}_6 = x_{n-2}, x_{n-1}, \mathcal{C}_7 = x_{n-2}, x_{n-1}, x_n$. Let $\mathcal{C}_1 \succ \cdots \succ \mathcal{C}_{a_k}$ be the triangular sets with class $\geq k$. Then the triangular sets with class $k-1$ are $x_{n-k+1}, \{x_{n-k+1}\} \cup \mathcal{C}_1, \ldots, \{x_{n-k+1}\} \cup \mathcal{C}_{a_k}$. Let a_k be the number of polynomials in the maximal triangular set with class k. We have $a_{k-1} = 2a_k + 1$ and $a_1 = 2a_2 + 1 = 2^2 a_3 + 2 + 1 = a^3 a_4 + 2^2 + 2 + 1 = 2^{n-1} + \cdots 2 + 1 = 2^n - 1$. Considering the trivial triangular set $\{1\}$, we have $m \leq 2^n$.□

By Lemma 3.4, among all the chains contained in a polynomial set $\mathbb{P}$, there exists one with the lowest ordering. Such a chain is called a *CS* of $\mathbb{P}$. We have the following basic property for CSs [23, 28].

Lemma 3.5 *Let $\mathcal{A}$ be a CS of a polynomial set $\mathbb{P}$. If P is reduced wrt $\mathcal{A}$, then a CS of $\mathbb{P} \cup \{P\}$ is of lower ordering than $\mathcal{A}$.*

Let $\mathbb{P}$ be a polynomial set. We set $\mathbb{P}_0 = \mathbb{P}$ and choose a CS $\mathcal{B}_0$ of $\mathbb{P}_0$. Let $\mathbb{R}_0 = \mathrm{prem}(\mathbb{P}_0, \mathcal{B}_0)$. Suppose that $\mathbb{R}_0 \neq \emptyset$. Then we form a new polynomial set $\mathbb{P}_1 = \mathbb{P} \cup \mathbb{R}_0$. Choose now a CS $\mathcal{B}_1$ of $\mathbb{P}_1$. By Lemma 3.5, $\mathcal{B}_1$ is of lower ordering than $\mathcal{B}_0$. Continuing in this way, we will obtain successively $\mathbb{P}_i, \mathcal{B}_i, \mathbb{R}_i, i = 1, 2, \ldots$, for which

$$\mathcal{B}_0 \succ \mathcal{B}_1 \succ \mathcal{B}_2 \succ \cdots .$$

By Lemma 3.4, the sequence can only be a finite one so that up to a certain stage m we should have $\mathbb{R}_m = \emptyset$. According to [28, 29], the above procedure can be exhibited in the form of the scheme (8) below:

$$\begin{array}{rcccccccc} \mathbb{P} & = & \mathbb{P}_0 & \mathbb{P}_1 & \cdots & \mathbb{P}_i & \cdots & \mathbb{P}_m & \\ & & \mathcal{B}_0 & \mathcal{B}_1 & \cdots & \mathcal{B}_i & \cdots & \mathcal{B}_m = \mathcal{C} & \\ & & \mathbb{R}_0 & \mathbb{R}_1 & \cdots & \mathbb{R}_i & \cdots & \mathbb{R}_m = \emptyset & \end{array} \tag{8}$$

where

$$\mathbb{P}_i = \mathbb{P}_{i-1} \cup \mathbb{R}_{i-1} \tag{9}$$

$\mathcal{B}_i$ is a CS of $\mathbb{P}_i$, and $\mathbb{R}_i = \mathrm{prem}(\mathbb{P}_i, \mathcal{B}_i)$. As a consequence of Lemma 3.4, we have

Proposition 3.6 *In procedure (8), we have $m < 2^n$.*

In scheme (8), $\mathcal{B}_m = \mathcal{C}$ verifies

$$\text{prem}(\mathbb{P}, \mathcal{C}) = \{0\} \text{ and } \text{Zero}(\mathbb{P}) \subset \text{Zero}(\mathcal{C}). \tag{10}$$

Any chain $\mathcal{C}$ satisfying property (10) is called a *Wu-CS* of $\mathbb{P}$. We have the following key property of a Wu-CS.

Theorem 3.7 (Well-ordering principle in $\mathbb{R}_2$) *Let $\mathcal{C}$ be a Wu-CS of a polynomial set $\mathbb{P}$. Then we have*

$$\begin{aligned}
&\overline{\text{Zero}}(\mathbb{P}) \\
&= \overline{\text{Zero}}(\mathcal{C}/\mathbf{I}_{\mathcal{C}}) \bigcup \cup_{i=1}^{p} \overline{\text{Zero}}(\mathbb{P} \cup \mathcal{C} \cup \{I_i\}) &(11)\\
&= \overline{\text{Zero}}(\mathcal{C} \cup \{I_1 + 1, \ldots, I_p + 1\}) \bigcup \cup_{i=1}^{p} \overline{\text{Zero}}(\mathbb{P} \cup \mathcal{C} \cup \{I_1 + 1, \ldots, I_{i-1} + 1, I_i\}) &(12)
\end{aligned}$$

where $I_i, i = 1, \ldots, p$ are the initials of the polynomials in $\mathcal{C}$. When $i < 0$, we assume that I_i does not occur in the formula.

Proof: Equation (11) is a direct consequence of the remainder formula (6). Equation (12) is a consequence of (11), (3), (1), and the fact that $\overline{\text{Zero}}(P) \cup \overline{\text{Zero}}(Q) = \overline{\text{Zero}}(P) \cup \overline{\text{Zero}}(Q/P) = \overline{\text{Zero}}(P) \cup \overline{\text{Zero}}(\{P + 1, Q\})$. □

This result is significant because it represents the zero set for a general polynomial set as the zero set of a chain. By Lemma 3.3, if the CS is non-conflict, then $\overline{\text{Zero}}(\mathbb{P}) \neq \emptyset$.

In procedure (8), the size of $\mathbb{P}_i$ could increase very fast. We may adopt the following way to compute $\mathbb{P}_i$ and Theorem 3.7 is still valid.

$$\mathbb{P}_i = \mathbb{P} \cup \mathcal{B}_{i-1} \cup \mathbb{R}_{i-1} \tag{13}$$

A more drastic way to reduce the size of $\mathbb{P}_i$ is proposed by Wu [30]. Instead of (13), we use the following formula to compute $\mathbb{P}_i$

$$\mathbb{P}_i = \mathcal{B}_{i-1} + \mathbb{R}_{i-1}. \tag{14}$$

Then $|\mathbb{P}_i|$ is always less than or equal to $|\mathbb{P}|$. In this case, procedure (8) will terminate, but C_m is not a Wu-CS of $\mathbb{P}$ any more. We have the following result.

Theorem 3.8 (Modified well-ordering principle) *Let $\mathcal{C}$ be a chain computed from a polynomial set $\mathbb{P}$ with procedures (8) and (14), $I_j, j = 1, \ldots, s$ the initials of the polynomials in $\mathcal{C} = \mathcal{B}_m, \mathcal{B}_{m-1} \ldots, \mathcal{B}_0$ with the initials of polynomials in $\mathcal{C}$ appearing first in the sequence, and $H_j = \text{prem}(I_i, \mathcal{C}), j = 1, \ldots, s$. Then, we have*

$$\overline{\text{Zero}}(\mathbb{P}) = \overline{\text{Zero}}(\mathcal{C} \cup \{H_1 + 1, \ldots, H_s + 1\}) \bigcup \cup_{i=1}^{s} \overline{\text{Zero}}(\mathbb{P} \cup \mathcal{C} \cup \{H_1 + 1, \ldots, H_{i-1} + 1, H_i\}) \tag{15}$$

Proof: Let $K_l = \prod_{i=1}^{l} I_i$ and $J_l = \prod_{i=1}^{l} H_i$. From [30], we have

$$\overline{\text{Zero}}(\mathbb{P}) = \overline{\text{Zero}}(\mathcal{C}/K_m) \bigcup \cup_{i=1}^{s} \overline{\text{Zero}}(\mathbb{P} \cup \mathcal{C} \cup \{I_i\}/K_{i-1}) \tag{16}$$

Since $\mathcal{C}$ is a chain, the initials of the polynomials in $\mathcal{C}$ are reduced wrt $\mathcal{C}$. Hence for the initials $I_i, i = 1, \ldots, t$ of $\mathcal{C}$, we have $H_i = I_i$ and $K_i = J_i$. For $j > t$, $H_j = \text{prem}(I_j, \mathcal{C})$. Then for $j > t$, by the remainder formula (6), we have $\overline{\text{Zero}}(\mathcal{C}/K_j) = \overline{\text{Zero}}(\mathcal{C}/K_t \prod_{i=t+1}^{j} I_i) = \overline{\text{Zero}}(\mathcal{C}/J_t \prod_{i=t+1}^{j} H_i) = \overline{\text{Zero}}(\mathcal{C}/J_j)$. Then, (16) becomes

$$\overline{\text{Zero}}(\mathbb{P}) = \overline{\text{Zero}}(\mathcal{C}/J_m) \bigcup \cup_{i=1}^{s} \overline{\text{Zero}}(\mathbb{P} \cup \mathcal{C} \cup \{H_i\}/J_{i-1}).$$

Now, equation (15) can be proved similarly to (12). □

Note that in (12) and (15), we obtain disjoint decomposition for the zero set $\overline{\text{Zero}}(\mathbb{P})$. The technique to obtain this kind of decomposition was introduced in [5, 22].

3.3 Zero Decomposition Theorems in $\mathbb{R}_2$

We now give the zero decomposition theorem (ZDT). Notice that the following ZDT given in [29] is still valid and the proof is also the same.

Theorem 3.9 (ZDT) *For a finite polynomial set $\mathbb{P}$, there is an algorithm to determine non-conflict chains $\mathcal{A}_j, j = 1, \ldots, s$, such that*

$$\overline{\text{Zero}}(\mathbb{P}) = \cup_{j=1}^{s} \overline{\text{Zero}}(\mathcal{A}_j/\mathbf{I}_{\mathcal{A}_j}).$$

In $\mathbb{R}_2$, we give the following more elegant form of zero decomposition theorem.

Theorem 3.10 (Disjoint Monic ZDT) *For a finite polynomial set $\mathbb{P}$, we can find monic chains $\mathcal{A}_j, j = 1, \ldots, s$, such that*

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

and $\overline{\text{Zero}}(\mathcal{A}_i) \cap \overline{\text{Zero}}(\mathcal{A}_j) = \emptyset$ for $i \neq j$. As a consequence, we have

$$|\overline{\text{Zero}}(\mathbb{P})| = \sum_{i=1}^{s} 2^{\dim(\mathcal{A}_i)}.$$

Proof: By Theorem 3.7, we have (12). If $\mathcal{C}$ is monic, then $I_i + 1 = 0$. Let $\mathcal{A}_1 = \mathcal{C}$ and repeat procedure (8) for $\mathbb{P}_1 = \mathcal{C} \cup \{I_1 + 1, \ldots, I_p + 1\}$. Otherwise, repeat procedure (8) for $\mathbb{P}_1$ and $\mathbb{P}_2 = \mathbb{P} \cup \mathcal{C} \cup \{I_1 + 1, \ldots, I_{i-1} + 1, I_i\}$. Since I_i is reduced wrt $\mathcal{C}$, according to Lemma 3.5, the new chains obtained in this way will be of lower ordering than that of $\mathcal{C}$. By Lemma 3.4, the procedure will end in a finite number of steps and all the chains obtained are monic. Since the components are disjoint in (12), by Lemma 3.2, the number of solutions are $\sum_{i=1}^{s} 2^{\dim(\mathcal{A}_i)}$. □

We give a precise description for this ZDT in Algorithm **DMZDT**.

Example 3.11 *Let $P = x_1x_2x_3 - 1$. By Theorem 3.9, we have $\overline{\text{Zero}}(P) = \overline{\text{Zero}}(P/x_1x_2)$. By Theorem 3.10 or Algorithm 1, $\overline{\text{Zero}}(P) = \overline{\text{Zero}}(x_1 + 1, x_2 + 1, P) \cup \overline{\text{Zero}}(x_1, P) \cup \overline{\text{Zero}}(x_1 + 1, x_2, P) = \overline{\text{Zero}}(x_1 + 1, x_2 + 1, x_3 + 1)$.*

Algorithm 1 — DMZDT($\mathbb{P}$)

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

Output: A sequence of monic chains $\mathcal{A}_i$ such that $\text{Zero}(\mathbb{P}) = \cup_i \text{Zero}(\mathcal{A}_i)$ and $\text{Zero}(\mathcal{A}_i) \cap \text{Zero}(\mathcal{A}_j) = \emptyset$ for $i \neq j$.

1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$, do
  2.1 Choose a $\mathbb{P}$ from $\mathbb{P}^*$. $\mathbb{P}^* = \mathbb{P}^* \setminus \{\mathbb{P}\}$
  2.2 Set $\mathbb{Q}$ to be a copy of $\mathbb{P}$.
  2.3 Do
    $\mathcal{C}$ = A CS of $\mathbb{Q}$.
    $\mathbb{R} = \text{prem}(\mathbb{Q} \setminus \mathcal{C}, \mathcal{C})$.
    $\mathbb{Q} = \mathbb{Q} \cup \mathbb{R}$ (or $\mathbb{P} \cup \mathcal{C} \cup \mathbb{R}$).
  Until $\mathbb{R} = \emptyset$.
  2.4 Let $\mathbb{I} = \{\text{init}(P) \neq 1 \mid P \in \mathcal{C}\} = \{I_1, \ldots, I_s\}$.
  2.5 If $\mathbb{I} = \emptyset$, $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{C}\}$.
  2.6 Else, do
    Let $J = \prod_{i=1}^{s} I_i$.
    If $J \neq 0$, do $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathcal{C} \cup \{I_1 + 1, \ldots, I_s + 1\}\}$.
    For i from 1 to s, do.
      $\mathbb{P}_1 = \mathbb{P} \cup \mathcal{C} \cup \{I_1 + 1, \ldots, I_{i-1} + 1, I_i\}$.
      $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathbb{P}_1\}$.

3 Return $\mathcal{A}^*$.

Example 3.12 *Let* $P = \{x_1x_2 + x_2 + x_1 + 1\}$. *By By Algorithm 1,* $\overline{\text{Zero}}(P) = \overline{\text{Zero}}(\mathcal{A}_1) \cup \overline{\text{Zero}}(\mathcal{A}_2)$, *where* $\mathcal{A}_1 = x_1, x_2{+}1, \mathcal{A}_2 = x_1{+}1$. *Then,* $|\overline{\text{Zero}}(P)| = 2^0{+}2^1 = 3$.

3.4 Complexity Analysis of the Modified Well-ordering Principle

We will show that the key step of the zero decomposition, the modified well-ordering principle, can be done in a polynomial number of steps and with a polynomial number of polynomial multiplications.

We repeat the modified well-ordering principle here.

$$\begin{array}{rccccccc} \mathbb{P} = & \mathbb{P}_0 & \mathbb{P}_1 & \cdots & \mathbb{P}_i & \cdots & \mathbb{P}_m & \\ & \mathcal{B}_0 & \mathcal{B}_1 & \cdots & \mathcal{B}_i & \cdots & \mathcal{B}_m = \mathcal{C} & \\ & \mathbb{R}_0 & \mathbb{R}_1 & \cdots & \mathbb{R}_i & \cdots & \mathbb{R}_m = \emptyset & \end{array} \tag{17}$$

where $\mathbb{P}_i = \mathcal{B}_{i-1} \cup \mathbb{R}_{i-1}$, $\mathcal{B}_i$ is a CS of $\mathbb{P}_i$, and $\mathbb{R}_i = \text{prem}(\mathbb{P}_i, \mathcal{B}_i)$. The following lemma gives a bound for the length of procedure (17).

Lemma 3.13 *Let* $\mathcal{B}_0 \succ \mathcal{B}_1 \succ \cdots \succ \mathcal{B}_m$ *be the sequence of chains in procedure (17). Then* $m \leq n$.

Proof: We denote by $X^{(i)}$ the set of the leading variables of polynomials occurring in $\mathcal{B}_0, \ldots, \mathcal{B}_i$. We will prove that

$$|X^{(0)}| < |X^{(1)}| < \cdots < |X^{(m-1)}| < |X^{(m)}|.$$

Since there are at most n leading variables in a chain, the above formula implies that $m \leq n$. Essentially, we need to prove that if $X^{(i)} = X^{(i+1)}$ then $\mathbb{R}_{i+1} = \emptyset$ and the procedure will end. First let us note that $\mathbb{P}_{i+1} = \mathcal{B}_i \cup \mathbb{R}_i$, and for $x_k \in X^{(i)}$ there exists at most one polynomial in $\mathbb{P}_{i+1}$ with x_k as the leading variable. If $X^{(i)} = X^{(i+1)}$, we will show that the leading variables of the polynomials in $\mathbb{R}_i$ are in $X^{(i-1)}$. Otherwise, let P be a polynomial with leading variable x_{i_0} which is the one with the smallest order and not in $X^{(i-1)}$. We consider two cases: (1) $x_{i_0} \prec x$ for $x \in X^{(i)}$. Then P is a polynomial in $\mathbb{P}_{i+1}$ with smallest order and should be in the basic set $\mathcal{B}_{i+1}$, which contradicts to the fact $X^{(i)} = X^{(i+1)}$. (2) $\mathcal{B}_i = B_1, \ldots, B_s$ and $x_{i_0} \succ B_t \succ B_{t-1} \succ \cdots B_1$. Take t to be the largest to satisfy the above condition. Since $\mathbb{P}_{i+1} = \mathcal{B}_i \cup \mathbb{R}_i$, when selecting a basic set for $\mathbb{P}_{i+1}$, $B_1, \ldots, B_t$ could be first selected. Since $P \in \mathbb{R}_i$, it must be reduced wrt $\mathcal{B}_i$. Hence, the next polynomial to be selected is P and P will be in $\mathcal{B}_{i+1}$, which again contradicts to $X^{(i)} = X^{(i+1)}$. We have proved that if $X^{(i)} = X^{(i+1)}$ then the leading variables of the polynomials in $\mathbb{R}_i$ are in $X^{(i-1)}$. As mentioned before, for each $x_k \in X^{(i-1)}$, there is at most one polynomial in $\mathbb{P}_i$ involving x_k. So $\mathbb{R}_i$ is a chain. For the same reason, $\mathcal{B}_i \cup \mathbb{R}_i$ is also a chain which should be $\mathcal{B}_{i+1}$. Hence $\mathbb{R}_{i+1} = \emptyset$ and we proved the result. □

Theorem 3.14 *Let $l = |\mathbb{P}|$. The modified well-ordering principle, or procedure (17), terminates for at most $n+1$ iterations and needs $O(n^2 l)$ polynomial multiplications.*

Proof: By Lemma 3.13, in procedure (17), $m \leq n$. That is, the procedure will stop after at most $n+1$ iterations. Notice that to do a pseudo-remainder needs two polynomial multiplications. To compute $\mathbb{R}_i$, since $|\mathbb{P}_i| \leq l$, we need to do at most $2|\mathbb{R}_i||\mathcal{B}_i| \leq 2nl$ multiplications. So the total number of multiplications is at most $m * (2nl) \leq 2n(n+1)l$. □

3.5 Using Wu Chains and Weak Chains

Note that the output of Algorithm 1 is a sequence of chains. To improve the efficiency of the algorithm, we could use other types of chains.

A triangular set $\mathcal{A}$ of form (5) is said to be a *Wu chain* if $\text{init}(A_i)$ is reduced wrt $\mathcal{A}_{i-1}$. $\mathcal{A}$ is called a *weak chain* if $\text{prem}(\text{init}(A_i), \mathcal{A}_{i-1}) \neq 0$.

The concept of Wu chain is defined in [29]. The concept of weak chain is defined in [5]. Similar to [29] and [5], we can develop zero decomposition theorems for these types of chains. The purpose of using these chains is to reduce the size of the polynomials occurring in the algorithm.

We have three ways to generate new polynomial sets: (9), (13), and (14) and three types of chains. Therefore, we have nine types of combinations to do zero decomposition. We will compare these approaches in Section 6.

Algorithm 2 — TDZDT($\mathbb{P}$)

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

Output: A sequence of monic chains $\mathcal{A}_i$ such that $\text{Zero}(\mathbb{P}) = \cup_i \text{Zero}(\mathcal{A}_i)$ and $\text{Zero}(\mathcal{A}_i) \cap \text{Zero}(\mathcal{A}_j) = \emptyset$.

1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$ do
  2.1 Choose a $\mathbb{Q}$ from $\mathbb{P}^*$. $\mathbb{P}^* = \mathbb{P}^* \setminus \{\mathbb{Q}\}$
  2.2 Set $\mathcal{A} = \emptyset$.
  2.3 While $\mathbb{Q} \neq \emptyset$ do
    2.3.1 If $1 \in \mathbb{Q}$, $\text{Zero}(\mathbb{Q}) = \emptyset$. Set $\mathbb{Q} = \mathcal{A} = \emptyset$ and goto 2.4.
    2.3.2 Let $\mathbb{Q}_1 \subset \mathbb{Q}$ be the polynomials with the highest class.
    2.3.3 Let $Q \in \mathbb{Q}_1$ be a polynomial whose initial is of the lowest ordering.
    2.3.4 Let $Q = Ix_c + U$ such that $\text{cls}(Q) = c$, $\text{init}(Q) = I$.
    2.3.5 If $I = 1$, do
      $\mathcal{A} = \mathcal{A} \cup \{Q\}$.
      $\mathbb{Q} = (\mathbb{Q} \setminus \mathbb{Q}_1) \cup \text{prem}(\mathbb{Q}_1, Q)$.
    2.3.6 Else, do
      Let $Q_1 = x_c + U$, $\mathbb{Q}_2 = \mathbb{Q}_1 \setminus \{Q\}$.
      $\mathcal{A} = \mathcal{A} \cup \{Q_1\}$.
      $\mathbb{Q} = (\mathbb{Q} \setminus \mathbb{Q}_1) \cup \{I + 1\} \cup \text{prem}(\mathbb{Q}_2, Q_1)$.
      $\mathbb{P}_1 = (\mathbb{Q} \setminus \{Q\}) \cup \{IU + U + I\} \cup \mathcal{A}$.
      $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathbb{P}_1\}$.
  2.4 If $\mathcal{A} \neq \emptyset$, do
    Set $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$.
3 Return $\mathcal{A}^*$

4. A Top-Down Algorithm for Zero Decomposition

In the preceding section, the zero decomposition algorithm repeatedly uses the well-ordering principle to obtain the CSs. The algorithm follows the traditional way of doing the elimination [23, 28]. It processes bottom up, that is, it starts from the polynomials with the lowermost classes and works the way to polynomials with higher classes. Another approach is to work top-down, that is, it starts from the polynomials with the highest class [4, 15, 16, 27].

In this section, we will give a more direct algorithm **TDZDT** to obtain a monic zero decomposition based on the top-down idea. Again, by taking into the special properties of $\mathbb{R}_2$, our decomposition algorithm has stronger properties.

Theorem 4.1 *Algorithm* **TDZDT** *is correct and to obtain each chain A_i in the algorithm, we need $O(nl)$ polynomial multiplications where $l = |\mathbb{P}|$.*

Proof: Consider the set $\mathbb{Q}$ of polynomials in the algorithm. $\mathbb{Q}_1 \subset \mathbb{Q}$ is the set

of polynomials with the highest class and $Q = Ix_c + U \in \mathbb{Q}_1$ a polynomial whose initial is of the lowest ordering. If $I = 1$, then for $P = I_1 x_c + U_1 \in \mathbb{Q}_1$ we have $P_1 = \text{prem}(P, Q) = P + I_1 Q$. As a consequence, $\text{Zero}(\{Q, P\}) = \text{Zero}(\{Q, P_1\})$. Therefore, we have

$$\overline{\text{Zero}}(\mathbb{Q}) = \overline{\text{Zero}}((\mathbb{Q} \setminus \mathbb{Q}_1) \cup \{Q\}) \cup \text{prem}(\mathbb{Q}_1, Q)).$$

If $I \neq 1$, by (4) and (2), we can split the zero set of $\mathbb{Q}$ as two disjoint parts:

$$\begin{aligned}
\overline{\text{Zero}}(\mathbb{Q}) &= \overline{\text{Zero}}(\mathbb{Q} \cup \{I+1\}) \cup \overline{\text{Zero}}(\mathbb{Q} \cup \{I\}) \\
&= \overline{\text{Zero}}((\mathbb{Q} \setminus \{Q\}) \cup \{Q_1, I+1\}) \cup \overline{\text{Zero}}((\mathbb{Q} \setminus \{Q\}) \cup \{I, U\}) \qquad (18) \\
&= \overline{\text{Zero}}((\mathbb{Q} \setminus \{Q\}) \cup \{Q_1, I+1\}) \cup \overline{\text{Zero}}((\mathbb{Q} \setminus \{Q\}) \cup \{IU+U+I\}) \qquad (19)
\end{aligned}$$

where $Q_1 = x_c + U$. Equation (19) comes from (2). The first part can be treated similarly to the case of $I = 1$ and the second part will be treated recursively with algorithm **TDZDT**. This proves that if $\mathcal{A}_i, i = 1, \ldots, s$ are the output of the algorithm, then $\overline{\text{Zero}}(\mathbb{P}) = \cup_i \overline{\text{Zero}}(A_i)$.

The termination of the algorithm can be proved in two steps. First, we will show the termination for the inner loop (step 2.3), that is, for each finite polynomial set $\mathbb{Q}$, the algorithm will terminate. After each iteration of the loop, the polynomial Q will be added to $\mathcal{A}$ and the highest class of the polynomials in $\mathbb{Q}$ will be reduced. Hence, this loop will end and give a chain $\mathcal{A}$. Second, we need to show the termination for the outer loop (step 2). For a polynomial set $\mathbb{P}$, we assign an index $(c_n, \ldots, c_1)$ where c_i is the number of polynomials in $\mathbb{P}$ and with class i. In the algorithm, there are essentially two cases where new polynomial sets are generated. In the first case, we replace $\mathbb{Q}$ with $\mathbb{Q}' = (\mathbb{Q} \setminus \mathbb{Q}_1) \cup \{Q\} \cup \text{prem}(\mathbb{Q}_1, Q)$. In the second case, we add $\mathbb{Q}'' = (\mathbb{Q} \setminus \{Q\}) \cup \{IU + U + I\}$ to $\mathbb{P}^*$. It is clear that the index of $\mathbb{Q}'$ or $\mathbb{Q}''$ is less than the index of $\mathbb{Q}$ in the lexicographical ordering in both cases. Due to Dickson's lemma, a strictly decreasing sequence of indexes must be finite. This proves the termination of the algorithm.

Finally, we will analyze the complexity of the inner loop of the algorithm (step 2.3), that is, the complexity to obtain a chain from $\mathbb{Q}$. After each iteration, the highest class of the polynomials in $\mathbb{Q}$ will be reduced at least by one. Then, this loop will execute at most n times. If $I = 1$, then the new $\mathbb{Q} = (\mathbb{Q} \setminus \mathbb{Q}_1) \cup \text{prem}(\mathbb{Q}_1, Q)$ contains at most $l - 1$ polynomials. If $I \neq 1$, the newly generated polynomial set $\mathbb{Q} = (\mathbb{Q} \setminus \{Q\}) \cup \{IU + U + I\}$ contains at most l polynomials. Then, after each iteration, the new $\mathbb{Q}$ contains at most l polynomials. In each iteration, we also need to compute at most $l-1$ pseudo-remainders. Since the initial of Q is 1, each pseudo-remainder wrt Q needs one polynomial multiplication. Then we need to do $l - 1$ polynomial multiplications in each iteration. In all, the algorithm needs $O(nl)$ polynomial multiplications. □

Example 4.2 *Let* $\mathbb{P} = \{x_1x_2 + x_2 + x_1 + 1\}$. *Since* $\mathbb{P}$ *contains one polynomial, we have* $Q = (x_1 + 1)x_2 + x_1 + 1 = Ix_2 + U$, *and* $Q_1 = x_2 + x_1 + 1$. *Then* $\overline{\text{Zero}}(\mathbb{P}) = \overline{\text{Zero}}(I +$

Algorithm 3 — TDZDTA($\mathbb{P}$)

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

Output: A sequence of monic chains $\mathcal{A}_i$ such that $\mathrm{Zero}(\mathbb{P}) = \cup_i \mathrm{Zero}(\mathcal{A}_i)$ and $\mathrm{Zero}(\mathcal{A}_i) \cap \mathrm{Zero}(\mathcal{A}_j) = \emptyset$.

1 Set $\mathbb{P}^* = \{\mathbb{P}\}$, $\mathcal{A}^* = \emptyset$.
2 While $\mathbb{P}^* \neq \emptyset$ do
  2.1 Choose a $\mathbb{Q}$ from $\mathbb{P}^*$. $\mathbb{P}^* = \mathbb{P}^* \setminus \{\mathbb{Q}\}$
  2.2 Set $\mathcal{A} = \emptyset$.
  2.3 While $\mathbb{Q} \neq \emptyset$ do
    2.3.1 If $1 \in \mathbb{Q}$, $\mathrm{Zero}(\mathbb{Q}) = \emptyset$. Set $\mathbb{Q} = \mathcal{A} = \emptyset$ and goto 2.4.
    2.3.2 Let $\mathbb{Q}_1 \subset \mathbb{Q}$ be the polynomials with the highest class.
    2.3.3 Let $\mathbb{Q}_2 = \emptyset$, $\mathbb{P}_1 = \mathbb{Q} \setminus \mathbb{Q}_1$.
    2.3.4 While $\mathbb{Q}_1 \neq \emptyset$ do
      Let $Q = Ix_c + U \in \mathbb{Q}_1$, $\mathbb{Q}_1 = \mathbb{Q}_1 \setminus \{Q\}$.
      $\mathbb{P}_2 = \mathbb{P}_1 \cup \mathbb{Q}_1 \cup \mathbb{Q}_2 \cup \{I, U\}$.
      $\mathbb{P}^* = \mathbb{P}^* \cup \{\mathbb{P}_2\}$.
      $\mathbb{Q}_2 = \mathbb{Q}_2 \cup \{x_c + U\}$, $\mathbb{P}_1 = \mathbb{P}_1 \cup \{I + 1\}$.
    2.3.5 Let $Q = x_c + U \in \mathbb{Q}_2$.
    2.3.6 $\mathcal{A} = \mathcal{A} \cup \{Q\}$.
    2.3.7 $\mathbb{Q} = \mathbb{P}_1 \cup \mathrm{prem}(\mathbb{Q}_2, Q)$.
  2.4 If $\mathcal{A} \neq \emptyset$, do
    Set $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$.
3 Return $\mathcal{A}^*$

$1, Q_1) \cup \overline{\mathrm{Zero}}(I, U) = \overline{\mathrm{Zero}}(x_1, Q_1) \cup \overline{\mathrm{Zero}}(x_1+1, x_1+1)$. *After simplification, we obtain the decomposition:* $\overline{\mathrm{Zero}}(\mathbb{P}) = \overline{\mathrm{Zero}}(x_1, x_2+1) \cup \overline{\mathrm{Zero}}(x_1+1)$ *and* $|\overline{\mathrm{Zero}}(\mathbb{P})| = 2^0+2^1 = 3$.

Although the number of polynomial multiplications needed in the algorithm is small, the degree and the size of the polynomials could increase very fast due to the multiplication of polynomials. We may adopt the following strategy to reduce the degree of the polynomials occurring in the algorithm. Before doing the pseudo remainders, we reduce the initials of the polynomials in $\mathbb{Q}_1$ in step 2.3.2 of the Algorithm 2 to 1. In that case, the pseudo-remainder needs additions only: for $P = x_c + U_1$ and $Q = x_c + U_2$, $\mathrm{prem}(Q, P) = U_1 + U_2$. As a consequence, degree of $\mathrm{prem}(Q, P)$ is less than or equal to the degrees of P and Q. Based on the above idea, we give the algorithm **TDZDTA**.

Theorem 4.3 *Algorithm* **TDZDTA** *is correct. The algorithm does not need polynomial multiplications and the degree of all the polynomials occurring in the algorithm is bounded by* $\max_{P \in \mathbb{P}} \deg(P)$.

Proof: Algorithm 3 is basically Algorithm 2. The only difference is that before doing pseudo-remainder in step 2.3.7, we reduce the initials of the polynomials in $\mathbb{Q}_1$ to 1 with formula (18). In this case, the pseudo-remainder of two polynomials becomes the addition of the two polynomials. Then the algorithm does not need polynomial multiplications. Also note that addition of polynomials does not increase the degree. This prove the theorem. □

5. Implementation of the Algorithms

We implemented the algorithms introduced in this paper with C language. In this section, we discuss several key issues that affect the efficiency of the program.

5.1 Polynomial Size vs Decomposition Branches

There exist two extreme methods to solve a set of Boolean equations.

A1 We could assign each variable the values of 0 and 1 and test whether the equations are satisfied. This is basically to compute the truth-table.

A2 Due to (2), a system of equations can be "easily" reduced to one equation. By Corollary 2.2, a non-constant polynomial equation must have solutions which can be found easily if such a polynomial is given.

The problem with approach A1 is that we need to check 2^n sets of values. But to check whether a set of values is a solution of the equations, we do not need to compute large polynomials. On the other side, in approach A2, we need only consider one polynomial. But, this polynomial could be very large. The two extreme cases are of course very inefficient. It seems that all of the approaches is trying to find an *optimized balance point* between the size increase of the polynomials and the number of cases to be checked.

In the case of CS method, each polynomial set $\mathbb{P}$ in $\mathbb{P}^*$ (step 2 of Algorithms 1 and 2) is called a *branch*. The problem is to find a balance point between the size of polynomials and the number of *branches*. For instance, Algorithm 3 does not increase the degree of the polynomials and will generally produce polynomials of small sizes, but it will produce more branches. On the other hand, Algorithm 2 produces less branches, but it generally will produce larger polynomials than Algorithm 3. In our implementation, we adopt the following:

Principle of Balance Between Sizes and Branches. *Try to produce as few branches as possible under the constraint that the memory of the computers be sufficiently used.*

According to our experiments, the size of the polynomials can be effectively controlled by using the splitting formula (18) and different types of chains introduced in Section 3.5. The main problem is branch control. Here are several possible ways to reduce the number of branches.

S1 The following strategy can be used to reduce the number of branches without increase the size of the polynomials. For a polynomial set $\mathbb{P}$, we select a polynomial of the form $x_c + U$ where U is a monomial not involving x_c and replace x_c in $\mathbb{P}$ by U. This process does not change the zero set of $\mathbb{P}$. Experiments show that most of branches have no solutions and this strategy can be used to detect the emptiness in an early stage in many cases.

S2 When adding a new polynomial, say the product of initials $I = \prod_{i=1}^{s} I_i$, to a polynomial set $\mathbb{P}$, we use the following procedure to split the zero set as several disjoint ones

$$\overline{\text{Zero}}(I) = \overline{\text{Zero}}(I_1) \cup \overline{\text{Zero}}(I_2, I_1 + 1) \cup \cdots \cup \overline{\text{Zero}}(I_s, I_{s+1} + 1, \ldots, I_1 + 1).$$

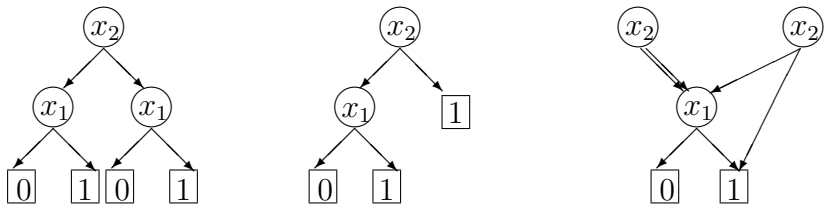
When combining with strategy S1, this strategy could simplify the decomposition procedure significantly.

S3 A well known strategy to simplify the problem is to select one or several variables, say x_c, which occur most often in $\mathbb{P}$ and consider $\mathbb{P}_{x_c=0}$ and $\mathbb{P}_{x_c=1}$ separately.

For a specific problem, we could use one or several of the above strategies together.

5.2 Using SZDD to Save Space Usage

We could encounter large space problem in two cases. First, a single polynomial produced in the algorithms could be large. Second, for some problems, the algorithm could produce a large number of branches.



(1) $P_1 = x_2x_1 + x_1$ (2) $P_2 = x_2 + x_1$ (3) SZDD for $\{P_1, P_2\}$

Fig. 1. SZDD for a polynomial set

The classic method of SZDD (shared zero-suppressed binary decision diagram) could be used to solve this problem [21]. Briefly speaking, for a set of polynomials $\mathbb{P}$, we could represent $\mathbb{P}$ as an SZDD in three steps:

- For each $P \in \mathbb{P}$, let $P = Ix_c + U$ such that $c = \text{cls}(P)$ and $I = \text{init}(P)$. We use a tree to represent P, where x_c is the root, I is the right child, and U is and the left child. Continue the above procedure for I and U recursively. This representation is called a *recursive* representation of P. In Fig. 1, (1) and (2) are recursive representations of P_1 and P_2.

- In the recursive representation of P, we share all the equivalent sub-graphs. The obtained representation is called the *ZDD* of P.
- For all polynomials in $\mathbb{P}$, we unite their ZDDs into one graph with the ZDDs of polynomials in $\mathbb{P}$ sharing their equivalent sub-graphs. In Fig. 1(3), we give the SZDD of $\{P_1, P_2\}$.

As shown by Table 3, using SZDD to represent Boolean polynomials could speedup the program significantly. ZDD representations are used to speedup the computation of Gröbner bases in [1].

6. Cryptanalysis of a Class of Stream Ciphers with CS Method

6.1 Nonlinear Filter Generators

Stream ciphers are an important class of encryption algorithm [20]. In this paper, we consider stream ciphers based on the linear feedback shift register (LFSR).

An 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 $c_L \neq 0$. 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

$$s_i = c_1 s_{i-1} + c_2 s_{i-2} + \cdots c_L s_{i-L}, i = L, L+1, \cdots . \tag{20}$$

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 (20) has period $2^L - 1$ [20].

Let $m_0, m_1, \ldots$ be the *plaintext digits*. We may use the sequence (20) as *key-stream* to generate the *ciphertext digits:*

$$c_i = m_i \oplus s_i, i = 0, 1, \ldots$$

where $\oplus$ is the XOR function. Decryption is defined by $m_i = c_i \oplus s_i$.

For a given sequence s_j of sufficient length, the Berlekamp-Massey algorithm may be used to recover the c_i in polynomial time [20]. Then, to use s_j as the key-stream is not secure. An often used technique 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 polynomial in $\mathbb{R}_2$ with m variables. We assume that $m \leq L$. Then we can use f and the sequence (20) to generate a new sequence as follows

$$z_i = f(s_{i-m}, \ldots, s_{i-1}), i = m, m+1, \cdots . \tag{21}$$

A combination of an LFSR and a nonlinear polynomial f is called a *nonlinear filter generator* (NFG). The sequence (21) could be used as the key-stream.

The filter functions used in this paper are from [3, 10]:

- 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, $x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_5 + x_6x_7 + x_3 + x_2 + x_1$

6.2 Algebraic Attack of Nonlinear Filter Generators with CS Method

By an algebraic attack of the nonlinear filter generator, we mean to recover the initial state of the LFSR from a certain number of key-stream in (21). Equivalently, we need 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

$$z_i = f(s_{i-m}, \ldots, s_{i-1}), i = m, \ldots, m + k \tag{22}$$

where k is a positive integer and s_i satisfy (20). Successful attacks on many kinds of stream ciphers were reported using the XL method [6, 7] and the Gröbner basis method [10].

We use the software package based on our algorithms to solve equation system (22). The statistic results are given in Tables 1 - 4. In these tables, L is the length of the LFSR, #sol is the number of solutions of the equation system. The experiments were done on a PC with a 3.19GHz CPU, 2G memory, and a Linux OS. The running times are given in seconds.

In Table 1, we give the running times of using Algorithm **DMZDT** to solve equations (22) generated with the filter generator GanFil 6. In the experiments, we set $k = L - 1$ in (22). For such a system, the number of equations and the number of variables are the same. The purpose of this experiment is to compare different versions of Algorithm **DMZDT**. The parameters "chain", "wuchain", and "wchain" mean that we use the chain, the Wu chain, and the weak chain defined in Section 3.5 respectively. The parameter in the first column means that we use (9), (13), or (14) in the well-ordering principle respectively. We can see that the approaches based on (14) are generally faster than other approaches. For the three types of chains, no single approach is better in all cases.

$L=$		40	60	81	100	128
#sols=		2	4	8	16	128
(9)	chain	1.49	0.12	1.52	4.00	5.68
	wuchain	1.47	0.79	1.95	18.87	37.46
	wchain	0.58	0.29	0.78	0.18	12.40
(13)	chain	1.03	0.05	6.56	0.32	1.23
	wuchain	0.72	0.12	0.50	3.55	5.34
	wchain	0.40	0.19	0.37	3.26	14.36
(14)	chain	0.06	0.08	0.17	0.37	2.77
	wuchain	0.06	0.06	0.12	0.31	1.13
	wchain	0.16	0.23	0.50	2.16	3.81

Table 1. Solving equations (22) with NFG Canfil6 using Algorithm **DMZDT**.

In Table 2, we give the results for solving equations (22) generated with different NFGs functions and $k = L-1$. The parameters in the first column give the NFGs used in the computation. The results show that when $k = L - 1$, the equation system (22) generally does not have a unique solution and the number of solutions could be large. This means that we cannot recover the initial state S_0 uniquely. The parameter #cs is the number of branches occurring in the computation process. Notice that #cs is much larger than #sol in most cases and the ratio (#cs)/(#sol) gives an approximate measure of the effectiveness of the algorithm on the corresponding problem. In four cases, the timing is marked with a $*$. This means that polynomials of lower degrees (annihilators) generated with methods introduced in [6, 7, 10] are used to speedup the computation. As a consequence, the number of solutions in these cases becomes very small. From the results for Canfil6, we can see that Algorithm **TDZDT** is generally faster than Algorithm **DMZDT**.

In order to recover the initial state S_0 uniquely, we increase k in (22) to m until the equation system (22) has a unique solution. The experimental results are given in Table 3, where the parameter $r = m/L$. From Table 3, we can obtain two conclusions. First, r varies from 1 to 2.8, which means that we generally need no more than $3L$ equations in order to find a unique solution in (22). Second, for the new equation system containing m equations, the running time is much faster than the equation system with L equations. The reason is that since the system has a unique solution, the number of branches need to be checked is much smaller.

In Table 4, we give the running times for four examples using the SZDD and the recursive representations for the polynomials respectively. These data show that using SZDD could significantly speedup the program.

Of the existing methods to solve Boolean equations, the Gröbner basis (GB) methods [1, 10, 25] are most close to our CS method. Table 5 gives a comparison of the GB method and the CS method. The timings for the GB method are from [10], which are based on the F5 method and are collected on an HP workstation with an Alpha EV68

	$L=$	40	60	81	100	128
CanFil1	time	0.03	0.04	0.25	0.21	13.11
	#sols	30	27	270	180	2340
	#cs	246	353	1937	445	1370
CanFil2	time	0.02	0.06	0.03	0.11	0.16
	#sols	7	7	6	56	76
	#cs	101	721	368	685	586
CanFil3	time	0.02	0.45	218.20	0.44*	1.27*
	#sols	36	660	14400	2	1
	#cs	95	1911	10590	1060	3725
CanFil4	time	0.03	29.99	1431.16	0.04*	11.53*
	#sols	48	6720	38400	2	12
	#cs	226	2667	22501	15	17564
CanFil5	time	0.00	0.00	0.00	0.01	0.01
	#sols	1	1	1	1	1
	#cs	2	2	2	2	2
CanFil6	time	0.00	0.01	0.01	0.03	0.26
	#sols	2	4	8	16	128
	#cs	7	55	112	124	995
CanFil7	time	0.01	0.02	0.05	0.14	6.71
	#sols	1	16	28	144	1360
	#cs	47	186	431	395	21815
CanFil8	time	0.01	0.03	0.12	0.21	4.14
	#sols	3	1	40	20	200
	#cs	76	483	1629	2480	47915
CanFil9	time	1.87*	0.67	1.71	1.90	12.1*
	#sols	5	20	6	8	358
	#cs	681	958	812	631	13170
CanFil10	time	0.17	0.09	0.08	42.44	13.76
	#sols	3	2	5	6	11
	#cs	203	721	477	148315	83860

Table 2. Cryptanalysis of Nonlinear Filter Generators with **TDZDT**.

processor at 1000MHz. The timings for the CS method are from Table 3. We should mention that the comparison is not precise in that the input polynomial systems are not exactly the same and the results from [10] are done several years ago. The main purpose of this comparison is to show that the methods proposed in this paper could provide a new effective tool for solving Boolean equations.

From these results, we may conclude that our CS method is comparable with the F5 method for these problems.

Filters	L=	40	60	81	100	128
CanFil1	time	0.04	0.00	0.01	0.05	0.06
	r	1.3	1.9	1.9	1.4	1.8
CanFil2	time	0.03	0.05	0.02	0.10	0.07
	r	1.1	1.2	1.7	1.4	1.7
CanFil3	time	1.77	0.01	0.29	0.76*	1.27*
	r	1.6	1.9	2.0	1.2	1.0
CanFil4	time	0.63	0.01	0.01	0.01*	0.02*
	r	1.5	2.8	1.9	1.5	1.4
CanFil5	time	0.00	0.00	0.00	0.01	0.01
	r	1	1	1	1	1
CanFil6	time	0.01	0.00	0.01	0.03	0.06
	r	1.3	1.8	1.8	1.6	1.8
CanFil7	time	0.01	0.01	0.01	0.07	0.07
	r	1	2.0	1.9	1.5	1.7
CanFil8	time	0.02	0.03	0.02	0.23	0.22
	r	1.1	1.0	1.9	1.4	1.7
CanFil9	time	4.83*	0.56	1.63	1.93	50.78*
	r	1.2	1.7	1.4	1.1	1.7
CanFil10	time	0.17	0.06	0.06	0.10	0.32
	r	1.1	1.5	1.5	1.4	1.6

Table 3. Find a unique solution with **TDZDT**.

L	NFG	use SZDD	not use SZDD
60	Canfil9	0.55	1018
100	Canfil3	0.76	38
128	Canfil1	13.1	111
128	Canfil4	0.02	36

Table 4. The improvements of using SZDD for Algorithm **TDZDT**

7. Conclusions

In this paper, we present several methods to solve nonlinear equation systems over the finite field $\mathbb{F}_2$ based on the idea of CS. Due to the special property of $\mathbb{F}_2$, the given CS methods are much more efficient and have better properties than the general CS method. In particular, the well-ordering principle can be executed in a polynomial number of steps and uses a polynomial number of polynomial multiplications.

We use our methods to solve equations raised from cryptanalysis of stream ciphers based on nonlinear filter generators. Extensive experiments have been done for equation systems with variables ranging from 40 to 128. The purpose of the experiments is two folds: to compare different variants of our algorithms and to show that our

Examples	L	GB	CS
CanFil 1	80	1.1	0.01
CanFil 1	128	10.1	0.06
CanFil 2	80	1.1	0.02
CanFil 2	128	10.3	0.07
CanFil 3	80	1.3	0.29
CanFil 3	128	12.4	1.27
CanFil 4	80	1	0.01
CanFil 4	128	9.5	0.02
CanFil 5	80	0.1	0.01
CanFil 5	128	9.1	0.01

Examples	L	GB	CS
CanFil 6	80	0.8	0.01
CanFil 6	128	8.9	0.06
CanFil 7	80	0.97	0.01
CanFil 7	128	10	0.07
CanFil 8	40	18.1	0.02
CanFil 8	80	3645	0.02
CanFil 9	40	21.2	4.83
CanFil 9	70	2001	1.63(L=81)
CanFil 10	40	16.5	0.17
CanFil 10	89	14421	0.1(L=100)

Table 5. Timings for Gröbner Basis and CS Method

algorithm could provide an effective tool for solving equations over $\mathbb{F}_2$.

References

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

[2] Bryant, R.E. Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers*, **35**(8), 677-691, 1986.

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

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

[5] 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.

[6] Courtois, N. Higher Order Correlation Attacks, XL Algorithm, and Cryptanalysis of Toyocrypt, *ICISC*, LNCS 2587, 182-199, Springer, 2002.

[7] Courtois, N. Algebraic Attacks on Stream Ciphers with Linear Feedback, *EUROCRPYT 2003*, LNCS 2656, 345-359, Springer, 2003.

[8] Davis, M. and and Putnam, H. A Computing Procedure for Quantification Theory, *J. ACM*, **7**(3), 201-215, 1960.

[9] 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.

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

[11] Gallo, G. and Mishra, B. Efficient Algorithms and Bounds for Wu-Ritt Characteristic Sets, in *Progress in Mathematics*, **94**, 119-142, Birkhauser, Boston, 1991.

[12] Gao, X.S. and Luo, Y. A Characteristic Set Method for Difference Polynomial Systems, *ICPSS*, Nov. 24-26, Paris, 2004.

[13] Gao, X.S., Chai, F., Yuan, C. A Characteristic Set Method for Equation Solving in F2 and Applications in Cryptanalysis of Stream Ciphers, *MM-Preprints*, 42-56, 2006.

[14] He, S. and Zhang, B. Solving SAT by Algorithm Transform of Wu's Method, *J. Comput. Sci. and Tech.*, **14**, 468-480, 1999.

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

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

[17] Li, B. An Algorithem to Decompose a Polynomial Ascending Set into Irredncible Ones, *Acta Anal. Funct. Appl.*, **7**(2), 97-105, 2005.

[18] 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.

[19] Mao, W. and Wu, J. Application of Wu's Method to Symbolic Model Checking, *Proc. ISSAC'05*, 237-244, ACM Press, New York, 2005.

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

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

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

[23] Ritt, J.F. *Differential Algebra*, Amer. Math. Soc. Colloquium, 1950.

[24] Rudeanu, S. *Boolean Functions and Equations*, North-Holland, Amsterdam, 1974.

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

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

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

[28] Wu, W.T. Basic Principles of Mechanical Theorem-proving in Elementary Geometries, *J. Sys. Sci. & Math. Scis.*, **4**, 207-235, 1984. *J. Automated Reasoning*, **2**, 221-252, 1986.

[29] Wu, W.T. On Zeros of Algebraic Equations - An Application of Ritt Principle, *Chinese Science Bulletin*, **31**, 1-5, 1986.

[30] Wu, W.T. Some Remarks on Characeteristic-Set Formation, *MM-Preprints*, Vol. 3, 27-29, 1989.