ABSTRACT
This paper proposes an algebraic method to prove the correctness of Arithmetic Program which halts in finally number of steps. The main routine is to simulate the program by a BSS computational model over the real numbers, thus it can be represented by a system of polynomial equations. The problem of proving program correctness will be translated into an algebraic one, which decides if the zeros of two systems of polynomial equations equals. The proof complexity of this method depends on the computational steps of a program.

Keywords: Program Verification, Arithmetic Program, BSS Model, Groebner Bases.

1. INTRODUCTION
It has been well recognized that program verification is important but difficult, because proofs of program correctness are complex and tedious. For programming logic in first-order or higher order logic and some kinds of programming language, some good and efficient methods have been made, especially, proving program correctness using formal methods and might lead to new theoretical insights into this area. This method is based on the algebraic properties of BSS model of computation.

BSS model, defined by L. Blum et al. at the end of 1980s, is a model of machine working on an arbitrary commutative ring (or field) $R$ (see [5, 6]). Their model has given rise to a whole new theory of computability and computational complexity. In the model, a machine has a finite control given by a finite graph, and an unlimited number registers, each capable of holding an element of $R$. The computational steps consist in computing polynomial and possibly deciding the next step by comparing the result of an evaluation with 0. A pair of integer registers can be used as points in order to retrieve and set any register. When $R$ is a real closed field (real number field, e.g.), the computational steps of a program verification methods and might lead to new theoretical insights into this area. This method is based on the algebraic properties of BSS model of computation.

BSS model, defined by L. Blum et al. at the end of 1980s, is a model of machine working on an arbitrary commutative ring (or field) $R$ (see [5, 6]). Their model has given rise to a whole new theory of computability and computational complexity. In the model, a machine has a finite control given by a finite graph, and an unlimited number registers, each capable of holding an element of $R$. The computational steps consist in computing polynomial and possibly deciding the next step by comparing the result of an evaluation with 0. A pair of integer registers can be used as points in order to retrieve and set any register. When $R$ is a real closed field (real number field, e.g.), the computational steps of a BSS model which halts in finally number of steps for its inputs can be represented by a system of polynomial equations, which called the algebraic property of BSS model. This model has a wider range of description than Turing machine, e.g., some problems like Mandelbrot set, Julia set and etc. can’t be adequately addressed within the traditional discrete models of computation, but they can be described well by BSS model [6]. The research of BSS model and its applications has a fast and great development in the last few years [7, 8, 9]. We here restrict that the programs are so-called Arithmetic Program to solve arithmetic problems over a real closed field (see section 2). This is because any arithmetic problem can be represented by an algebraic system of equations in this kind of field.

The basic strategy of our method is as follows. First of all, a program $P$ written by some high level programming language to solve an algorithm problem, is simulated by a BSS model. When $P$ halts in the finitely number of steps, so does the model, thus a system of polynomial equations is obtained, which just describes the computational procedure. Finally, we check whether the same zeros of the two systems of polynomial equations or not (ignoring of all parameter variables); if yes, then the original program $P$ is total correct to solve the arithmetic problem. In the way of Human-computer interaction, we do some examples of verification using the above method via the software system of Mathematica. The complexity of proofs using the method depends on the computational steps of a program.

The structure of this paper is as follows. Section 2 defines Arithmetic Problem and Arithmetic Program, and talks about the algebraic representation of Arithmetic Program. After an brief introduction to BSS model, Section 3 gives firstly the representation of a polynomial equations of the program, then discusses zeros property of the polynomial equations according to Groebner Bases of the ideal. In section 4 contains the steps of verification of Arithmetic Program using algebraic method, and implement the verification of a small practice example.

2. ARITHMETIC PROGRAM
Let $(R, +, *, 0, 1)$ be a (ordered) field, the arithmetic terms over $R$ is defined as follows:

(1) Each constant over $R$ and variable which its domain is over $R$ is an arithmetic term.

(2) If $p, q$ are constants, variables or rational polynomials over $R$, then $p+q, p-q, p*q, p/q$ are arithmetic terms.

The arithmetic formulas over $R$ is defined by the following statement:

(1) If $p, q$ are two arithmetic terms over $R$, then $p=0, q=0, p=q, p=q, p\geq q, p\leq q, p\leq q$ over an ordered field) are arithmetic formulas.
A problem is called an Arithmetic Problem, if its solution can be described by the zeros of some arithmetic formula. For example, the problem of evaluating polynomial, \( y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \) is an Arithmetic Problem.

For the arithmetic operation of two rational polynomials, they can be translated a rational polynomial. According to Property 1, which is based on the property of polynomial equations below.

\[
\begin{align*}
\left\{ \begin{array}{l}
y - x^2 - 3 = 0 \\
x(xu^2 - 1) = 0
\end{array} \right. \quad \text{or} \quad \left\{ \begin{array}{l}
y - x + 1 = 0 \\
xu^2 + 1 = 0
\end{array} \right.
\]

Furthermore, it can be represented by a system of polynomial equations below.

\[
\begin{align*}
(y - x^2 - 3)(y - x + 1) &= 0 \\
(y - x^2 - 3)(xu^2 + 1) &= 0 \\
x(xu^2 - 1)(y - x + 1) &= 0 \\
x(xu^2 - 1)(xu^2 + 1) &= 0
\end{align*}
\]

Remark 1: Most of practical mathematical problems belong to the arithmetic problems over the real number field, and some arithmetic programs can be written by using various kinds of high level programming language to solve these problems.

3. SOME RESULTS ON BSS MODEL AND GROEBNER BASES

In [5, 6], two kinds of machine are defined: finite dimensional and infinite dimensional ones, the difference lying in the presence of an infinite number of registers in the input, output and state space, and of some machinery which is necessary in order to address such registers. We here will only introduce the finite dimensional model, because it can be proved that an infinite dimensional model is equivalent to a finite dimensional one in practical.

In the rest of the paper, the "field" will always mean "real closed field" although the original definition is a "ordered field".

A finite dimensional machine \( M \) over \( R \) consists of three spaces: the input space \( \overline{T} = R^l \), the output space \( \overline{O} = R^m \) and the state space \( \overline{S} = R^n \), together with a finite directed connected graph with node set \( \overline{N} = 1, 2, ..., N(N > 1) \) divided in four subsets: input, output, branch and computation nodes.

Node 1 is the only input node, having no incoming edge and one outgoing edge; node \( N \) is the only output node, having no outgoing edge. They have associated with linear functions, mapping respectively the input space to the state space and the state space to the output space. Any other node \( k \in 2, 3, ..., N - 1 \) can be of the following types:

(1). A branch node \( k \) has two outgoing edges, giving out successors \( \beta^+(n) \) and \( \beta^-(n) \). There is a polynomial function \( h_k : \overline{S} \rightarrow R \) associate to \( k \), and for a given \( \alpha \in \overline{S} \), branching on \( \beta^+(n) \) or \( \beta^-(n) \) will depend upon whether or not \( h_k(\alpha) \geq 0 \).

(2). A computation node has a single outgoing edge, so that a next node \( \beta \) (\( n \)) is defined, associated with it a polynomial map \( g_n : \overline{S} \rightarrow \overline{S} \). If \( R \) is a field then \( g_n \) could be taken rational.

We can view \( M \) as a discrete dynamical system over the full state space \( \overline{N} \times \overline{S} \). \( M \) induces a computing endomorphism on the full state space:

\[
\begin{align*}
&<1, \alpha > \rightarrow <1, \beta(1), \alpha > \\
&<N, \alpha > \rightarrow <N, \alpha >
\end{align*}
\]
<k, α > = β(k), g_k(α) > if k is a computation node

<k, α > = {<β^(n)(α), α > if h_k(α) ≥ 0
<β^(n)(α), α > if h_k(α) < 0}

if k

is a branching node.

The computation of M under input α is the orbit generated in the full state space by the computing endomorphism starting from <1, l(α) >. If the orbit reaches a fixed point of the form <N, β > for some β ∈ S we say that the machine halted, and that its output is α(β).

**Property 3.** Any arithmetic program written by a high level programming language (Pascal Language e.g.) can be simulated in O(n) by a BSS model with finite dimensional, where n is the number of the instructions in P.

**Proof.** A BSS model is just like a flow diagram in programming language, so this is a simple procedure of simulation. For assignment instructions, this is only corresponding. The instructions of input and output will be translated respectively input and output nodes, a few of nodes must be added because BSS model requires that they have the property of unique one. As for branch instructions, they become at least two nodes because the decidable condition is the first coordinate variable in the model. The complexity of simulation, therefore, is O(n).

**Corollary 4.** Let P be an arithmetic program, L a BSS model of computation. If P halts under some inputs in T steps, then L halts in O(T) for the same inputs.

If a finite-dimensional BSS machine over R halts in T steps, then its computation can be represented by the following polynomial equations [5].

\[
x_{k-1} u_{k-1} + 1)(x_{k-1} u_{k-1} - 1) = 0
\]

(3.2)

where k = 1, 2, ..., T, u_{k-1} is a parameter variable, β (n_{k-1}, x_{k-1} u_{k-1}) and g_k(n_{k-1}, x_{k-1}) are polynomials.

Groebner Basis is an important concept and technical tool in algebraic geometry. Let I⊂k[x_1, ..., x_n] be an ideal, I =< f_1, ..., f_s >, the variable has the lexicographic order x_1 > x_2 > ... > x_n. Then each f_i ∈k[x_1, ..., x_n] has a unique leading term LT(i), we denote by LT(i) the set of leading terms of elements of I. A Groebner Basis of I is a finite subset G ={g_1, ..., g_r} if

<LT(I) >= LT(g_1), ..., LT(g_r) >.

**Lemma 5.** [12] Let G be a Groebner Basis of an ideal I⊂k[x_1, ..., x_n], f ∈k[x_1, ..., x_n]. Then f ∈ I if and only if G reduces f to 0.

Elimination theory is a main method in solving a system of polynomial equations using Groebner Basis. The \(i\)th elimination ideal \(I_i\) of above ideal I is the ideal \(I \cap k[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]\). Thus \(I_i\) consists of all consequences of \(f_1 = ... = f_s = 0\) which eliminate the variables \(x_1, ..., x_i\).

**Lemma 6.** (Elimination Theorem [12]) Let \(I⊂k[x_1, ..., x_n]\) be an ideal and let G be a Groebner Basis of I with respect to lex order where \(x_1 > x_2 > ... > x_n\).

Then, for every \(0 \leq l \leq \infty\), the set \(G_l = G \cap k[x_1, ..., x_{n-l}]\) is a Groebner Basis of the \(l\)th elimination ideal \(I_l\).

A zero of elimination ideal \(I_l\) is said to be a partial zero of the original ideal I. The Extension Theorem gives a sufficient condition which can extend a partial zero to a complete zero of I. This theorem holds over an algebraically closed field. We talk below the extension of zeros over a real closed field R. Let \(I⊂R[x, y, z_1, ..., z_n]\) is an ideal and let the variable order be \(x > y > z_1 > ... > z_n\).

\(1 =< f_1, ..., f_{n+1} >\), where

\[
\begin{align*}
f_1 &= z_1 - f_1(x) \\
f_2 &= z_2 - f_2(x, z_1) \\
&\quad \vdots \\
f_n &= z_n - f_n(x, z_1, ..., z_{n-1}) \\
f_{n+1} &= y - f_{n+1}(x, z_1, ..., z_n)
\end{align*}
\]

(3.1)

**Lemma 7.** Let \(I⊂R[x, y, z_1, ..., z_n]\), where \(f_i\) have the above form. Then a zero \((x_0, y_0)\) of the second elimination ideal \(I_2\) can be extended to a complete one of I over real closed field.

**Proof:** These polynomial functions have special form, for each \(f_i(0 \leq i \leq n)\), both the degree and coefficient of leading term are 1 with respect to the variable order \(z_1 > ... > z_n\). Thus, given a value of \(x, z_1, ..., z_n\) have a unique value respectively, and y is determined uniquely by \(x, z_1, ..., z_n\). Furthermore, The elimination ideal \(I_2\) is composed of those polynomials which only contains variable \(x, y\). Based on the unique of y when x is given, a zero of \(I_2\) will be extended to a zero of I. Therefore, the conclusion holds.

In previous (3.2), the \(u_{k-1}\) will be determined by \(x_{k-1}\). We observe the second kind of polynomials which \(β\) is a polynomial function about \(x_{k-1} u_{k-1}\). Combining with the first kinds of equations, we will find the \(β\) has nothing to do with \(u_{k-1}\). According to Lemma 7, the partial zeros of I, which only contains \(x_0, y\) and \(u_{k-1}\), will be extended to a complete zero of I, where I is the ideal generated by all polynomials of the above equations (3.2).

For the previous example, the following is a PASCAL program, say P, to compute the original problem.
PROGRAM
VAR x, y: Real;
BEGIN
READ(x);
IF x>0 THEN y := x^2 + 3
ELSE y := x - 1;
WRITE('y =', y)
END.

P halts in the 5th steps, and its corresponding BSS model can be represented by the following system of polynomial equations (simplification form).

\[ r_{1,0} = x; r_{1,1} = r_{2,0} = r_{1,0}; r_{2,1} = r_{1,1}; r_{3,0} = r_{2,0}; r_{3,1} = 0 = (x^2 + 3)(n_2 - 3) - (x - 1)(n_2 - 4); n_0 = 1; n_1 = 2; r_{1,0} (u^2 r_{1,0} + 1) = 0; y = r_{3,1} = 2; r_{1,0} (u^2 r_{1,0} + 1) = 0; u = r_{1,1} = 2; n_0 = 2(u^2 r_{1,0} - u^2 r_{1,0} + 2) + 3/2 u^2 r_{1,0} (u^2 r_{1,0} + 1) \]

Let the order of variables be

\[ \{r_{1,0}, r_{1,1}, r_{2,0}, r_{2,1}, r_{3,0}, r_{3,1}, n_0, n_1, n_2, y, x, u \} \]

Via computer algebra system Mathematica, we get a Groebner Bases of these polynomials in the above system of polynomial equations w.r.t. the order.

\[ G = \{-6 - 4 u^2 x - 2 x^2 + 3 u^2 x^2 + 4 u^4 x^2 - u^4 x^2 - u^4 x^2 - u^4 x^2 + 2 y, -8 - 4 u^2 x + u^4 x^2 + 2 n_2, -2 + n_1, -1 + n_0, -6 - 4 u^2 x - 2 x^2 + 4 u^4 x^2 - u^4 x^2 - u^4 x^4 + 2 r_{3,1} + x + u^4 x^4, x + r_{3,0}, r_{2,1} + x + r_{2,0}, r_{1,1} + x + r_{1,0} \} . \]

4. MATHOD OF ARITHMETIC PROGRAM VERIFICATION

Here we use logic language to define the correctness of a program. Let Q be a problem, P a program to solve Q. P is said to be sound if, for given variables values, each output of P is a solution of Q. On the contrary, if for given variables values, each solution of Q is an output of P, then P is called completeness. So a program is correct if and only if P is sound and complete.

Suppose Q and P are characterized as two systems of polynomial equations, say (I), (II). All polynomials in (I) and (II) generate two ideals respectively, say (I) and (II). From the view of algebra, \( <I> \supset <II> \) means that P is sound, and \( <II>_1 \subseteq <I> \) means that P is complete, where \( <II>_1 \) is an elimination ideal which only contains some variables occurring in \( <I> \) and a zero of \( <II>_1 \) can be extended a complete one of \( <II> \). Therefore, P is correct if the two systems of polynomials have the same ideal which modular some parameter variables in \( <II> \). On this grounds, we get the following result.

**Theorem 8.** Let Q be an arithmetic program problem, P be an arithmetic program to solve Q and P halt in finitely many of steps. The method below decide whether or not P solves Q correctly.

Step 1. Represent Q by a system of polynomial equations, say (I).

Step 2. Simulate P by a finite-dimensional BSS model that halts in the finitely many of steps.

Step 3. Represent the above derived BSS by a system of polynomial equations, say (II).

Step 4. Compute the Groebner Bases respectively, say H and G, of two ideals generated by the polynomials in (I) and (II).

Step 5. Check whether or not all polynomials in H are reduced to 0 by G. If yes, then P is sound.

Step 6. Let M be the subset consisting all polynomials in G which only those variables in H occur. Check whether or not all polynomials of M are reduced to 0 by H. If yes, then P is complete.

Step 7. If P is both sound and complete, then claim that P solves Q correctly. Otherwise, P doesn’t solve Q correctly. **Proof.** "\( \Rightarrow \)" If P is correct, then solutions of arithmetic problem Q should be the solutions of program P, so H reduces all polynomials of M to 0, i.e., P is complete. Ignoring of the values of parameter variables, zero points of P should be those of Q, i.e. G reduce all polynomials of H to 0, therefore P is sound-ness.

\( \subset \) The soundness of P guarantees that the solutions of P is those of Q when P neglecting parameter variables. Those polynomials of M are Groebner bases of the elimination ideal which eliminates all parameter variables in G (Elimination Theorem [12]), so the solutions of M is the partial solutions of G. Considering the special form of the algebraic representation of BSS model, which the lead term coefficient of parameter variables is 1, so the partial solutions can be extended to a complete solutions of P (Extension Theorem [12]). That is to say, M has a solutions, then P also has one and the solution of P is the extension solution. The completeness of P guarantees that H reduce all polynomials of M to 0, i.e., the solutions of Q are those of M. Furthermore, those solutions can also be extended to the solutions of P. So the program P is correct.

In the previous example, when the order of variables be \( \{y, x, u\} \), via the software system of Mathematica, we obtain the Groebner Bases H of all polynomials in the polynomial equations corresponding to the original arithmetic problem.

\[ H = \{- x + u^4 x^3, -6 - x - 4 u^2 x - x^2 + u^2 x^2 + 4 u^4 x^2 - u^4 x^2 + 2 y\} . \]

We let G reduce H, in step 2, it is reduced to 0, so the program P is sound. In G, we investigate the set of polynomials M only concluding the variables y, x, u appeared in G.

\[ M = -6 - 4 u^2 x - 2 x^2 + u^2 x^2 + 4 u^4 x^2 - u^4 x^3 + u^4 x^4 + 2 y . \]

In step 1, that H reducing M is 0. Therefore, the program P is correct.

**Remark 2 :**

1. In the procedure of practical verification, the computation of Groebner Bases has been done as a software package in most systems of computer algebra such as Mathematica, Maple and etc., so the testing speed is very fast under the real execution procedure.

2. In order to extend the range of verification of arithmetic program, we are exploring to use real RAM (Random Access Machine) program simulating the arithmetic program firstly. And then we simulate the RAM program by a BSS model. Where the real RAM means to extend RAM over integer numbers to real numbers, that is to say, the arbitrarily integer is extended of as an arbitrarily real number in the input-output tapes. This can dispose the lower machine instructions.
3. In the two procedures of an Arithmetic Problem and BSS model are presented by a system of polynomial equations, it is worthwhile to research how to correspond to the two kinds of parameters.

5. REFERENCES