World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:2, No:9, 2008

A P-SPACE Algorithm for Groebner Bases Computation in Boolean Rings

Quoc-Nam Tran

***Abstract*—**The theory of Groebner Bases, which has recently been honored with the ACM Paris Kanellakis Theory and Practice Award, has become a crucial building block to computer algebra, and is widely used in science, engineering, and computer science. It is well-known that Groebner bases computation is EXP-SPACE in a general setting. In this paper, we give an algorithm to show that Groebner bases computation is P-SPACE in Boolean rings. We also show that with this discovery, the Groebner bases method can theoretically be as efficient as other methods for automated verification of hardware and software. Additionally, many useful and interesting properties of Groebner bases including the ability to efficiently convert the bases for different orders of variables making Groebner bases a promising method in automated verification.

***Keywords*—**Algorithm, Complexity, Groebner basis, Applications of Computer Science.

I. Introduction

SINCE its invention in 1965 by Bruno Buchberger, the Groebner basis method has become one of the most important techniques in providing automated problem-solving tools to address challenges in robotics, computer-aided design, systems design, modeling biological systems and many other related areas [4, 5, 28, 31]. The method is implemented in all major computer algebra systems including Mathematica, Macsyma, Magma, Maple and Reduce. These software programs enable computers to manipulate mathematical equations and expressions in symbolic form, and are heavily used in science and mathematics. Buchberger's work has recently been honored with the ACM Paris Kanellakis Theory and Practice Award, which honors specific theoretical accomplishments that significantly affect the practice of computing. Nevertheless, the field is still under active development both in the direction of improving the method by new theoretical insights and in finding new applications.

This paper is dedicated to investigating the theoretical foundations for the Groebner basis method in Boolean rings. We are interested in this special setting because the Groebner basis method in Boolean rings can be used for automated formal verification of hardware and software in computer science.

Unfortunately, it is well-known that Groebner bases computation is EXP-SPACE in a general setting over polynomial rings [20]. Since other approaches for automated formal verification are P-SPACE [3, 21, 22], for the theoretical competitiveness of an alternative approach using Groebner bases it is very important to prove that Groebner bases computation is also P-SPACE.

The paper is organized as follows: In the next section, we will summarize some basic facts about P-SPACE, Boolean rings and the Buchberger's algorithm for Groebner bases computation in a general setting. In Section III, we give a different algorithm for Groebner bases computation in Boolean rings and prove that this new algorithm for Groebner Bases computation is P-SPACE. Finally, in Section IV, we will discuss possible applications of our work for automated formal verification of hardware and software in computer science.

Dr. Q.-N. Tran is a professor at Lamar University, U.S.A.

II. Preliminaries

In this section, we will summarize some basic facts about complexity, Boolean rings and the method of Groebner bases.

A. Time and Space Complexity

To analyze the efficiency of our algorithms, we utilize the complexity of computational problems in terms of the amount of memory that they require. Time and space are two of the most important considerations when we seek practical solutions to many computational problems. In fact, time and space complexity are related to each other. Furthermore, space complexity shares many of the features of time complexity and serves as a further way of classifying problems according to their computational difficulty.

Definition 1: The time and space complexity classes, P, NP, P-SPACE, NP-SPACE, EXP-TIME and EXP-SPACE, are defined as follows.

- P= $\{L|L$ is a language decided by a deterministic Turing machine M that halts on all inputs in $O(n^k)$ steps on any input of length n for some $k\}$.
- NP= $\{L|L$ is a language decided by a nondeterministic Turing machine M that halts on all inputs in $O(n^k)$ steps on any input of length $n\}$.
- P-SPACE= $\{L|L$ is a language decided by a deterministic Turing machine M that halts on all inputs and uses $O(n^k)$ maximum number of tape cells on any input of length n for some $k\}$.
- NP-SPACE= $\{L|L$ is a language decided by a nondeterministic Turing machine M that halts on all inputs and uses $O(n^k)$ maximum number of tape cells on any input of length $n\}$.

- EXP-TIME= $\{L|L$ is a language decided by a deterministic Turing machine M that halts on all inputs in $O(2^{n^k})$ steps on any input of length n for some $k\}$.
- EXP-SPACE= $\{L|L$ is a language decided by a nondeterministic Turing machine M that halts on all inputs and uses $O(2^{n^k})$ maximum number of tape cells on any input of length $n\}$.

We summarize the relationship between complexity classes in the following proposition [24, 25].

Proposition 1: P⊆NP⊆P-SPACE=NP-SPACE⊆EXP-TIME⊆EXP-SPACE.

B. Boolean ring

Boolean algebras, which were introduced by Boole in the 1850's to codify the laws of thought, have become a popular topic of research since then. The discovery in 1930's of the duality between Boolean algebras and Boolean spaces by Stone [26, 27, 6] was a major breakthrough of the field. Stone also proved that Boolean algebras and Boolean rings are the same in the sense that one can convert from one algebraic structure to the other. In spite of its long history and elegant algebraic properties, the Boolean ring representation has rarely been used in the computational context.

Definition 2: A ring $\mathbf{K} = \langle K, +, \cdot, 0, 1\rangle$ is Boolean if $\mathbf{K}$ satisfies $x^2 \approx x, \forall x \in K$.

Lemma 1: If $\mathbf{K}$ is a Boolean ring, then $\mathbf{K}$ is commutative and $x + x \approx 0$ [6].

Every Boolean algebra $(K, \wedge, \vee)$ gives rise to a ring $(K, +, \cdot)$ by defining $a + b = (a \wedge \neg b) \vee (b \wedge \neg a)$ (this operation is called XOR in the case of logic) and $a \cdot b = a \wedge b$. The zero element of this ring coincides with the 0 of the Boolean algebra; the multiplicative identity element of the ring is the 1 of the Boolean algebra. Conversely, if a Boolean ring $\mathbf{K}$ is given, we can turn it into a Boolean algebra by defining $x \vee y = x + y + x \cdot y$ and $x \wedge y = x \cdot y$. Since these two sets of operations are inverses of each other, we can say that every Boolean ring arises from a Boolean algebra, and vice versa. Furthermore, a map $f : A \to B$ is a homomorphism of Boolean algebras if and only if it is a homomorphism of Boolean rings. The categories of Boolean rings and Boolean algebras are equivalent. By using these translations, there exists a Boolean polynomial for each Boolean formula and vice versa.

Since congruences on rings are associated with ideals, it follows that the same must hold for Boolean algebras. An ideal of the Boolean algebra $\mathbf{K}$ is a subset I such that $\forall x, y \in I$ we have $x \vee y \in I$ and $\forall a \in K$ we have $a \wedge x \in I$. This notion of ideal coincides with the notion of ring ideal in the Boolean ring $\mathbf{K}$. An ideal I of R is called prime if $I \neq K$ and if $a \wedge b \in I$ always implies $a \in I$ or $b \in I$. An ideal I of K is called maximal if $I \neq K$ and if the only ideal properly containing I is K itself. These notions coincide with ring theoretic ones of prime ideal and maximal ideal in the Boolean ring $\mathbf{K}$.

Despite its extremely simplicity, the Boolean ring representation has not been used extensively both in logical reasoning and in computation. The main reason, which has been shared by other researchers, is that the XOR operator used in Boolean rings is nilpotent and hence negation does not appear in the normal forms. This makes Boolean ring formulas hard to read for human because one cannot tell which predicate symbol is negated and which one is not. Especially, when a formula is long, it is almost impossible to make a natural interpretation of its meaning.

For model checking, we are interested in checking the correctness of a model only, not what the proofs look like. The efficiency of model checking very much depends on efficient internal data structure, which can provide a uniform representation and fast basic operations. Particularly, unlike Boolean algebras when "don't care" (DC) conditions are involved, Boolean rings can provide a satisfactory algebraic framework for effectively handle of the problems.

C. The Method of Groebner Bases

Once the Boolean formulas have been converted into and equivalent system of polynomials in the corresponding Boolean ring, one can use the results from symbolic computation to perform calculation on the polynomial system. In this section, we give a short introduction to basic facts on admissible term orders, weight vectors, and the method of Groebner bases. We refer to [4, 5, 8, 28, 29, 31] for missing details.

Let K be a computable field such as the field of rational numbers and $K[x_1, \ldots, x_n]$ the polynomial ring in n variables over K. We denote the set of power products in the variables $x_1, x_2, \ldots, x_n$ by $[X]$.

Definition 3: A total order on $[X]$ is called an *admissible term order* iff

1) $1 = x_1^0 \cdot x_2^0 \cdots x_n^0 < t$, $\forall t \in [X] \setminus \{1\}$, and
2) $s < t \Rightarrow s \cdot u < t \cdot u$, $\forall s, t, u \in [X]$.

Let f be a non-zero polynomial in $K[x_1, \ldots, x_n]$ and $\prec$ be an admissible term order. We denote

- $lpp_\prec(f)$ the leading power product of f with respect to $\prec$.
- $lc_\prec(f)$ the leading coefficient of f with respect to $\prec$.
- $in_\prec(f) = lc_\prec(f) \cdot lpp_\prec(f)$ the initial term of f with respect to $\prec$.

Definition 4: [Polynomial Reduction] Let $f, g, h \in K[X], F \subset K[X]$. We say that g reduces to h with respect to f denoted by $g \to_f h$ iff there are power products $s, t \in [X]$ such that s has a non-vanishing coefficient c in g, $s = lpp_\prec(f) \cdot t$, and $h = g - \frac{c}{lc_\prec(f)} \cdot t \cdot f$. We say that g reduces to h with respect to F denoted by $g \to_F h$ iff there is $f \in F$ such that $g \to_f h$.

A power product (or term) $u \in [X]$ is said to be a *direct divisor* of another power product $t \neq u$ if u divides t but there is no

power product v such that u divides v and v divides t. In other words, u has exactly one less variables than t.

Definition 5: Let G be a finite subset of $K[X] \setminus \{0\}$, $\prec$ be an admissible term order over $[X]$, and I be an ideal in $K[X]$. Then G is a Groebner basis of I with respect to $\prec$ iff $\langle G_\prec \rangle = \langle I_\prec \rangle$. Furthermore, G is called a minimal Groebner basis iff $lpp_\prec(f) \nmid lpp_\prec(g)$, $\forall f, g \in G, f \neq g$. G is called a reduced Groebner basis iff $\forall f, g \in G, f \neq g$, we cannot reduce f by g. G is normed iff $lc_\prec(g) = 1$, $\forall g \in G$.

The following important theorem is based on [4].

Theorem 1: Let $I = \langle F \rangle$ be an ideal in $K[X]$ and $\prec$ be a term order on $[X]$. The ideal I has a unique finite normed reduced Groebner basis.

Let G be the unique finite normed reduced Groebner basis of I with respect to $\prec$. Every monic monomial m can be reduced by G to an irreducible polynomial denoted by $\text{nf}(m)$. Clearly $(m - \text{nf}(m)) \in I$. We say that a monic monomial m is *minimal reducible* iff m is reducible (i.e. $m \neq \text{nf}(m)$ and all its direct divisors are irreducible.

Definition 6: Let $f, g \in K[X], t = \text{lcm}(lpp_\prec(f), lpp_\prec(g))$. Then

$$\text{cp}(f,g) = (t - \frac{t}{lc_\prec(f) \cdot lpp_\prec(f)} \cdot f, t - \frac{t}{lc_\prec(g) \cdot lpp_\prec(g)} \cdot g)$$

is called the critical pair of f and g. The difference of the elements of the critical pair $\text{s-pol}(f,g) = \frac{t}{lc_\prec(f) \cdot lpp_\prec(f)} \cdot f - \frac{t}{lc_\prec(g) \cdot lpp_\prec(g)} \cdot g$ is called the S-polynomial of f and g.

Buchberger's algorithm [4]:

Input a finite subset $F \subset K[X]$, a term order $\prec$.
Output a Groebner basis G of F w.r.t. $\prec$.

Step-1 $G \leftarrow F$
$C \leftarrow \{\{g_1, g_2\} \mid g_1, g_2 \in G, g_1 \neq g_2\}$
Step-2 While not all pairs $\{g_1, g_2\} \in C$ are marked
choose an unmarked pair $\{g_1, g_2\}$;
mark $\{g_1, g_2\}$;
$h \leftarrow$ normal form of $\text{s-pol}(f,g)$ w.r.t. G ;
if $h \neq 0$ then
$C \leftarrow \{\{g, h\} \mid g \in G\}$;
$G \leftarrow G \cup \{h\}$;
return G.

Lemma 2: Groebner basis computation is EXP-SPACE in general [20].

Given an ideal I and an admissible term order $\prec$, we denote the reduced Groebner basis of I with respect to $\prec$ by $GB(I, \prec)$. The following lemma gives us many different ways to check whether or not a set of polynomials is a Groebner basis.

Lemma 3: Let I be an ideal in $K[X]$, $\prec$ a term order, $F \subset K[X]$, and $\langle F \rangle = I$. The following statements are equivalent [31]:

1) F is a Groebner basis of I with respect to $\prec$.
2) f is reducible to 0 with respect to F, $\forall f \in I$.
3) f is reducible with respect to F, $\forall f \in I \setminus \{0\}$.
4) $\rightarrow_F$ is a Church-Rosser reduction relation.

III. Groebner Bases Computation is P-SPACE in Boolean Rings

In this section, we define a decision problem for Groebner bases computation in Boolean rings using linear algebra, and then prove that the Groebner bases computation is in P-SPACE. We make use of linear algebra techniques in [20], where the authors showed that Groebner bases computation is EXP-SPACE in general. Using the condition $x^2 \approx x$, for all x in a Boolean ring K, it is easy to derive a linear degree bound for polynomials over a Boolean ring as follows.

Proposition 2: The degree of polynomials in a Boolean ring $K[X]$ is bounded by n, where n is the number of variables.

Even though the degree bound for polynomials in $K[X]$ (and hence the degree bound for polynomials in a Groebner basis) is linear in n, which is significantly smaller than the doubly exponential degree bound of $(\frac{d^2}{2}+d)^{2^{n-1}}$ [12] for polynomials of a Groebner basis in a general setting, a polynomial of degree n in a Boolean ring may still have 2^n monomials. This means that a Groebner basis computation in that we store intermediate polynomials may still be EXP-SPACE. Fortunately, one can use on-the-fly techniques in that only necessary intermediate results will be recorded to improve the situation.

Let $F = \{f_1, \ldots f_s\}$ be a set of polynomials in a Boolean ring $K[X]$, and $\prec$ be a term order on $[X]$. Even though we do not know the reduced Groebner basis of F with respect to the term order $\prec$ yet, the existence and uniqueness of such a Groebner basis are guaranteed in Section II. Therefore, for every polynomial p there exists a unique normal form of p with respect to the reduced Groebner basis. Since $p \rightarrow^*_{GB(F,\prec)} \text{nf}(p)$, $p - \text{nf}(p)$ is in $I = \langle F \rangle$ and hence

$$p - \text{nf}(p) = \sum_{i=1}^{s} f_i \cdot h_i$$

for some f_i in F ind h_i in $K[X]$. In other words, $\text{nf}(p)$ is the smallest monic polynomial with respect to the term order $\prec$ in the I-coset of p. Alternatively, [15] showed that finding the normal form of a polynomial can be transformed into solving a linear algebra system of size $2^{O(n)} \times 2^{O(n)}$ without knowing the reduced Groebner basis of I with respect to the term order $\prec$. If we expand all polynomials (including the unknown polynomials h_i and $\text{nf}(p)$) to sums of monomials: $h_i = \sum_{x \in [X], \deg(x) \leq n} h_{i,x} \cdot x$, $f_i = \sum_{x \in [X], \deg(x) \leq n} f_{i,x} \cdot x$ and $\text{nf}(p) = \sum_{x \in [X], \deg(x) \leq n} r_x \cdot x$ where the $h_{i,x}$ and r_x are unknown coefficients, we have

$$\begin{aligned} p &= \sum_{x \in [X], \deg(x) \leq n} r_x \cdot x + \\ &\quad \sum_{i=1}^{s} (\sum_{x \in [X], \deg(x) \leq n} f_{i,x} \cdot x) \cdot \\ &\quad (\sum_{x \in [X], \deg(x) \leq n} h_{i,x} \cdot x) \\ &= \sum_{x \in [X], \deg(x) \leq n} (r_x + \\ &\quad \sum_{i=1}^{s} \sum_{u,v \in [X], u \cdot v = x} f_{i,u} \cdot h_{i,v}) \cdot x \\ &= M.b \end{aligned} \tag{1}$$

where $b = (h_{1,1}, \ldots, h_{1,x}, \ldots, h_{s,1}, \ldots, h_{s,x}, r_1, \ldots, r_x, \ldots)^T$, and M is a matrix of Boolean values (i.e. 0 and 1). The rows of matrix M correspond to terms $1, \ldots, x_1, \ldots, x_1 \cdot x_2 \cdots x_n$ and the columns correspond to the unknowns $h_{i,x}$s and r_xs

for all monomial x from 1 to $x_1 \cdot x_2 \cdots x_n$. Matrix M is free of rows and columns with all zeros, and the rows are rearranged with respect to the order of the monomials. For the columns, we arrange the columns correspond to h_xs before the columns correspond to r_xs. Also, column r_x corresponds to term x will come before column r_y corresponds to term y if $x \prec y$. Finding nf(p) can be done using the following algorithm:

Algorithm [Normal Form]:

Given a set of polynomials F, a term order $\prec$ and a polynomial p.
Find the normal form nf(p) of p with respect to $I = \langle F \rangle$ and $\prec$.
Step_1 Build M and b as in Equation 1.
Step_2 Find a full row rank sub-matrix
1a. Add the first non-zero row of M into an empty matrix M^+
1b. For row from 2 to the last row of M
If rank$(M^+ \cup row) \neq$ rank(M^+),
add row into M^+
Step_3 Find a full column rank sub-matrix
1a. Add the first non-zero column of M^+ into an empty matrix M'
Add the corresponding element of vector b into an empty vector b'
1b. For col from 2 to the last column of M^+
If rank$(M' \cup col) \neq$ rank(M'),
add col into M'
add the corresponding element of vector b into b'.
Return the solution of $p = M'.b'$

It is easy to see that Algorithm "Normal Form" always terminates and returns the normal form of p with respect to the given ideal I and term order $\prec$.

Example 1: Let $F = \{x + x \cdot y,\ y + x \cdot y\}$ and $\prec$ be the lexicographic order on $[x, y]$ where $x \prec y$. In this example we illustrate how the normal form of a polynomial $p = y$ with respect to $I = \langle F \rangle$ and $\prec$ can be calculated using Algorithm "Normal Form". First, we expand all polynomials (including the unknown polynomials h_i and nf(p)) to sums of monomials:

$$\begin{aligned} p &= (r_1 + r_x \cdot x + r_y \cdot y + r_{xy} \cdot x \cdot y) + (x + x \cdot y)(b_1 + \\ &\quad b_x \cdot x + b_y \cdot y + b_{xy} \cdot x \cdot y) + (y + x \cdot y)(c_1 + \\ &\quad c_x \cdot x + c_y \cdot y + c_{xy} \cdot x \cdot y \\ &= r_1 + (r_x + b_x + b_1) \cdot x + (c_1 + r_y + c_y) \cdot y + \\ &\quad (r_{xy} + b_1 + c_y + b_x + c_1) \cdot x \cdot y \end{aligned}$$

The corresponding linear algebra system is $y = M \cdot b$, where $b = (b_1, b_x, b_y, b_{xy},\ c_1, c_x, c_y, c_{xy},\ r_1, r_x, r_y, r_{xy})^T$ and

$$M = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

Notice that the rows of matrix M correspond to terms $1, x, y, x \cdot y$ and the columns correspond to the unknowns $b_1, b_x, b_y, b_{xy},\ c_1, c_x, c_y, c_{xy},\ r_1, r_x, r_y, r_{xy}$. The rank of M is 4. Following Algorithm "Normal Form", Column 1, 5, 9 and 10 of matrix M will be added into M'

$$M' = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{matrix}$$

The solution of the linear algebra system $p = M' \cdot b'$ is $(1, 1, 0, 1)^T$, where $b' = (b_1, c_1, r_1, r_x)^T$. This means that nf$(p) = 0 + 1 \cdot x = x$. Moreover, $h_1 = 1$, $h_2 = 1$, and nf$(p) + f_1 \cdot h_1 + f_2 \cdot h_2 = x + (x + x \cdot y) \cdot 1 + (y + x \cdot y) \cdot 1 = y = p$.

It is easy to double check using Buchberger algorithm that the reduced Groebner basis of F with respect to $\prec$ is $\{x + y\}$ and therefore the result from Algorithm "Normal Form" is the same as when the Groebner basis is used for calculating the normal form of p.

To analyze the complexity of Algorithm "Normal Form", we notice that linear algebra operations can be done using parallel computation. Following [13] we first formalize the work of parallel algorithms using a parallel random access machine consists of a set of processors P_0, P_1, . . ., an unbounded global memory, a set of input registers, and a finite program. Each processor has an accumulator, an unbounded local memory, a program counter, and a flag indicating whether the processor is running or not. All memory locations and accumulators are capable of holding arbitrary non-negative integers. The program consists of a sequence of possibly labeled instructions chosen from the following list

Instruction	Note
LOAD operand	From memory into the accumulator
STORE operand	Write to memory
ADD operand	Increase the value at operand
SUB operand	Decrease the value at operand
JUMP label	Change program counter
READ operand	From input register into the accumulator
FORK label	Start the first inactive processor at label
HALT	Stop the processor

Each operand may be a literal, an address or an indirect address. Each processor may access either global memory or its local memory, but not the local memory of any other processor. Initially, the input to the machine is placed in the input registers, all memory is cleared, the length of the input is placed in the accumulator of P_0, and P_0 is started. At each step in the computation, each running processor simultaneously executes the instruction given by its program counter in one unit of time, then advances its counter by one unless the instruction causes a jump. A FORK *label* instruction executed by processor P_i selects the first inactive processor P_j, clears P_j's local memory, copies P_i's accumulator into P_j's accumulator and starts P_j running at *label*. Simultaneous reads of a location in global memory are allowed, but if two processors try to write into the same memory location simultaneously, the parallel machine immediately halts and rejects the input. Several processors may read a location while

one processor writes into it; all reads are performed before the the value of the location is changed. Execution continues until a HALT is executed by processor P_0 or when two processors attempt to write into the same memory location simultaneously. The input is accepted only if there is some computation in which P_0 halts with a one in its accumulator; the time required to accept the input is the minimum over all such computations of the number of instructions executed by P_0.

Lemma 4: [13] Let L be accepted by a deterministic $T(n)$ time-bounded parallel random access machine, where n is the size of input. Then L is accepted by $T(n)^2$ space-bounded Turing machine.

Proof: We construct a Turing machine that simulates the work of the parallel random access machine by keeping track of the contents of P_0's accumulator when it halts and verifying that no two writes occur simultaneously at the same memory location. To enumerate the active processors at any level of the computation tree (see Figure 1), one needs at most $\log(2^{T(n)}) = T(n)$ space to write down a processor number. Writing down the contents of an accumulator takes at most $T(n) + \log n = O(T(n))$ space because addition and subtraction are the only arithmetic operators, and numbers can increase in length by at most one at each step. Writing down the level of the computation tree takes $\log T(n)$ space, and the program counter takes only constant space.

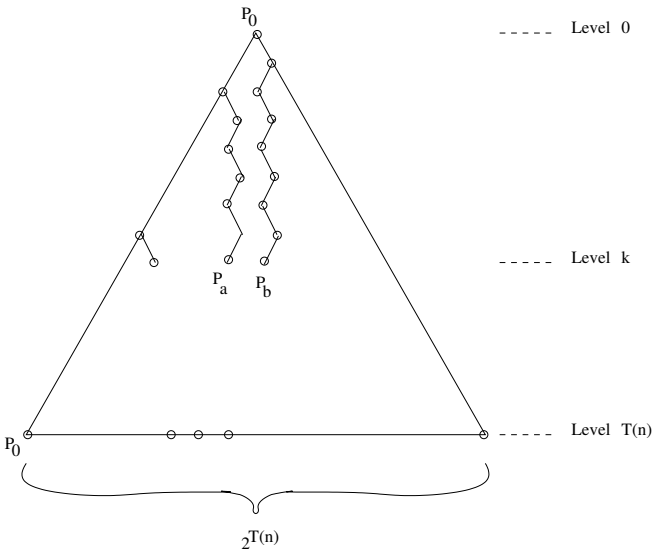

Figure 1. Computation tree

From any node on the tree, there are at most two children on the next level. Also, on any level of the tree there is only one P_0 node. Since the parallel machine is deterministic, at any step for each of the running processors there is exactly one instruction which can be executed. At level k, the Turing machine checks if P_0 executed the i^{th} instruction of its program, leaving c in its accumulator by recursively checking the instruction executed by P_0 at level $k-1$ and the ensuing contents of its accumulator, and the contents of the memory location referenced by instruction i. Since we need to go up to the root of the tree, $T(n) \cdot T(n) = O(T^2(n))$ memory space are needed. To verify that two writes do not occur simultaneously at level k, the Turing machine cycles through all pairs of possible active processors, check the executed instructions of the processors, the contents of their accumulators, and the contents of the memory locations referenced by the instructions. Again, $2 \cdot T(n) \cdot T(n) = O(T^2(n))$ memory space are needed. ■

We now state and analyze the complexity of Algorithm "Normal Form". Notice that we do not want to write down the whole matrix M because by doing so it would require an exponential amount of memory space. We will show how to solve the linear algebra system using on-the-fly calculations.

Lemma 5: Algorithm "Normal Form" is in P-SPACE.

Proof: Let s be the number of polynomials in F and S be the biggest number of monomials in all polynomials of F. Finding the value of any element in M requires $O(s \cdot S \cdot n)$ memory space. Furthermore, Csanky [9] has given parallel algorithms that takes $O(\log^2(2^n)) \sim O(n^2)$ time and uses $O(2^{4n})$ processors for: (a) inverting an order 2^n matrix, (b) solving a system of 2^n linear equations in 2^n unknowns, (c) computing an order 2^n determinant, (d) finding the characteristic polynomial of an order 2^nmatrix. The bound on the number of processors has been decreased to $O(2^{2.876 \cdot n})$ in [23] and $O(2^{2.851 \cdot n})$ in [14]. It is also known that the rank of a Hermitian matrix is equal to the number of its nonzero characteristic roots. Hence, if M is a Hermitian matrix and $f_M(\lambda) = \det(\lambda.I - M) = \lambda^k + c_1.\lambda^{k-1} + \cdots + c_k$ is its characteristic polynomial, then $\text{rank}(A) = k - i$, where $0 \leq i \leq k$ is the largest integer such that $c_{k-i} \neq 0$ and $c_{k-i+1} = c_{k-i+1} = \cdots c_k = 0$

One can compute the rank of a sub-matrix M' of M in $O(n^2)$ time as follows [18]:

1) First, one calculates $M^T \cdot M$. This takes $O(n)$ time and uses $O(n^2)$ processors.
2) Next, one calculates the coefficients $c_1, \ldots, c_k$ of the characteristic polynomial of $M^T M$. This takes $O(\log^2(2^n)) \sim O(n^2)$ time and uses $O(2^{4n})$ processors. The bound on the number of processors has been decreased to $O(2^{2.851 \cdot n})$ as mentioned before.
3) Finally, one determines the largest integer i such that $c_{k-i} \neq 0$ and $c_{k-i+1} = c_{k-i+1} = \cdots c_k = 0$. This can be done in $O(\log 2^n) \sim O(n)$ time and $O(2^n)$ processors using the fan-in technique. Then $\text{rank}(M') = k - i$.

These $O(n^2)$ time-bounded parallel algorithms, which uses $O(2^{2.851 \cdot n})$ processors and shares a common memory, can be converted into a $O((n^2 \cdot 2.851 \cdot n)^2) \sim O(n^6)$ space-bounded Turing machine using Lemma 4. Therefore, Algorithm "Normal Form" is in P-SPACE. ■

We now define a decision problem for Groebner bases computation in Boolean rings using linear algebra as follows.

Problem 1: [*Groebner bases*] Given a set of polynomials F in $K[X]$ and a term order $\prec$ on $[X]$, does it have 1 in the set $\{m - \text{nf}(m) : \text{for all minimal reducible monomial } m \text{ of degree at most } n\}$?

We derive an algorithm to solve the decision problem for Groebner bases computation in Boolean rings using linear algebra as follows:

Algorithm [GB using Linear Algebra]:

Given a set of polynomials F and a term order $\prec$.

Find the reduced Groebner basis of $I = \langle F \rangle$ with respect to $\prec$.

Step_1 Set $G' = \emptyset$; Build matrix M and vector b as in Equation 1.

Step_2 For all monomial m, $1 \not\prec m \prec x_1 \cdot x_2 \cdots x_n$ do
If $1 = m + \text{nf}(m)$ then stop and return $\{1\}$;
Add $m + \text{nf}(m)$ into G' when m is minimal reducible.

Step_3 return G'.

Example 2: Let $F = \{x + x \cdot y,\ y + x \cdot y\}$ and $\prec$ be the lexicographic order on $[x, y]$ where $x \prec y$. In this example we illustrate how the reduced Groebner basis of $I = \langle F \rangle$ with respect to $\prec$ can be calculated using Algorithm "GB using Linear Algebra". As illustrated in Example 1, the corresponding linear algebra system is $p = M' \cdot b'$, where $b' = (b_1, c_1, r_1, r_x)^T$ and

$$M' = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{matrix}$$

The solution of the linear algebra system $m = M' \cdot b'$ for monomials $m = x$ and $m = y$ are $(0, 0, 0, 1)^T$ and $(1, 1, 0, 1)^T$, respectively. We do not need to find the normal form of $x \cdot y$ because one of its divisors, y, is reducible and hence $x \cdot y$ is not minimal reducible monic monomial. Therefore, the set of polynomials $m + \text{nf}(m)$ for all minimal reducible monic monomials of degree ≤ 2 is $\{x + y\}$. This is indeed the reduced Groebner basis of I with respect to $\prec$.

It is easy to see that Algorithm "GB using Linear Algebra" always terminates. The correctness of the algorithm is guaranteed by the following lemma.

Lemma 6: The set of polynomials $m - \text{nf}(m)$ for all minimal reducible monic monomials of degree $\leq n$ is equal to the reduced Groebner basis G of $I = \langle F \rangle$ with respect to $\prec$.

Proof: We denote the set of polynomials $m - \text{nf}(m)$ for all minimal reducible monic monomials of degree $\leq n$ by G'. Clearly $G' \subseteq I$. All polynomial in G can be written in the form $m - \text{nf}(m)$, where m is the leading term of the polynomial. Since G is reduced, m must be a minimal reducible monic monomial of degree $\leq n$, and hence $G \subseteq G'$. That is, for all $f \in I$, f is reducible by G and hence by G'. Therefore, G' is a Groebner basis of I with respect to $\prec$. It is easy to see that $G'$is reduced and monic. Consequently, $G' = G$. ■

Lemma 7: Algorithm "GB using Linear Algebra" is in P-SPACE.

Proof: Step 2 of the algorithm enumerates all monic monomials up to degree n. In every pass through the loop, one needs to check at most $\sum_{i=1}^{n-1} \frac{n!}{i! \cdot (n-i)!} = 2^n - 2$ direct divisors of m and the monomial m itself to see if m is minimal reducible. In case m is minimal reducible, we output $m - \text{nf}(m)$. According to Lemma 5, this step requires a $O(n^6)$ space-bounded Turing machine. Therefore, the algorithm is in P-SPACE. ■

IV. Conclusion and Discussion

Algebraic reasoning such as Groebner basis-based or Hilbert's Nullstellensatz-based reasoning, has been used in propositional proof systems [7, 2]. For an appropriate measure of proof size, [7] showed that the Groebner basis algorithm finds a proof of a tautology in time polynomial in the size of the smallest such proof. Furthermore, the Groebner basis-based system polynomially simulates Horn clause resolution, and quasi-polynomially simulates resolution. In other words, Groebner proofs will have a better than worst-case behavior on the same classes of inputs than resolution does. On the other hand, there are simple tautologies that have polynomial-size Groebner proofs but require exponential-size resolution proofs. In comparison with Nullstellensatz-based system [2], [7] showed that there is a family of tautologies that have degree 3 Groebner refutations, but they require $\Theta(\sqrt{n})$ degree Nullstellensatz refutations, where n is the number of variables. Thus, there is an exponential separation between the Groebner basis-based and the Hilbert's Nullstellensatz-based systems.

Groebner bases considered as a Church-Rosser reduction relation or a term rewriting system has been used for propositional satisfiability in [10, 11, 17, 16, 19]. Techniques from algebraic geometry have also been proposed for symbolic model checking in lieu of BDDs [1].

However, the obstacles for an *effective use* of these concepts for model checking are that

- In model checking, one has to deal with a huge number (hundred or thousand) of variables .
- Even though we have a polynomial bound for the degree of Boolean polynomials and we will not encounter the coefficient swell problem during the calculation of Groebner bases in model checking, polynomial reductions may still be exponential.
- Traditional work in Groebner basis computation is focused on general polynomials where special care must be given for the coefficient swell problem.
- All of the current implementations of Buchberger's algorithm for Groebner basis computation either based on conventional representation of polynomial or linear algebra are not efficient enough for model checking.

We showed in this paper that there exists an algorithm for Groebner basis computation in Boolean rings that is P-SPACE. With this discovery, the Groebner Bases method is theoretically as efficient as other methods for automated verification of hardware and software. However, the algorithm we found

is far from practical use. We have been working on new data structures, techniques and algorithms for Groebner basis computation in Boolean rings where specific structures of model checking problems has been taken into account [30]. A more practical algorithm for Groebner basis computation in Boolean rings may help to develop more robust and scalable model checking methods based on novel and alternative technologies.

REFERENCES

[1] G. S. Avrunin. Symbolic model checking using algebraic geometry. In *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, pages 26–37, London, UK, 1996. Springer-Verlag.

[2] P. Beame, R. Impagliazzo, J. Krajicek, T. Pitassi, and P. Pudlak. Lower bounds on Hilbert's nullstellansatz and propositional proofs. In *Proceddings of the 35th Annual Symposium on Foundations of Computer Science*, pages 794–806, Santa Fe, NM, 1994.

[3] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of Design Automation Conference (DAC'99)*, 1999.

[4] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimensional Polynomial Ideal (in German)*. PhD thesis, Institute of Mathematics, Univ. Innsbruck, Innsbruck, Austria, 1965.

[5] B. Buchberger. Groebner Bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, chapter 6, pages 184–232. Reidel Publishing Company, Dodrecht, 1985.

[6] S. N. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 1981.

[7] M. Clegg, J. Edmonds, and R. Impagliazzo. Using Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of STOC'96*, pages 174–183, Philadelphia, PA, USA, 1996. ACM.

[8] S. Collart, M. Kalkbrener, and D. Mall. Converting bases with the Groebner walk. *Journal of Symbolic Computation*, 24(3/4):465–469, 1997.

[9] L. Csanky. Fast parallel matrix conversion algorithms. *SIAM J. Comput.*, 5:618–623, 1976.

[10] N. Dershowitz, J. Hsiang, G. Huang, and D. Kaiss. Boolean ring satisfiability.

[11] N. Dershowitz, J. Hsiang, G. Huang, and D. Kaiss. Intersection-based methods for satisfiability using boolean rings.

[12] T. D. Dubé. The structure of polynomial ideals and Groebner bases. *SIAM J. Comput.*, 19(4):750–773, 1990.

[13] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the* 10^{th} *annual ACM symposium on Theory of Computing*, pages 114–118. ACM Press, 1978.

[14] Z. Galil and V. Pan. Parallel evaluation of the determinant and of the inverse of a matrix. *Inform. Proc. Lett.*, 30:41–45, 1989.

[15] G. Hermann. Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. *Mathematische Annalen*, 95:736–788, 1925.

[16] J. Hsiang. Refutational theorem proving using term rewriting systems. *Artificial Intelligence*, 25:255–300, 1985.

[17] G. Huang, N. Dershowitz, and J. Hsiang. Satisfiability testing using simplification in boolean rings.

[18] O. Ibarra, S. Moran, and L. Rosier. A note on the parallel complexity of computing the rank of order n matrices. *Inform. Proc. Lett*, 11(4):162, 1980.

[19] D. Kaiss and N. Dershowitz. Boosting satisfiability testing using boolean rings.

[20] K. Kuehnle and E. Mayr. Exponential space computation of groebner bases. In *Proceedings of ISSAC*. ACM, 1996.

[21] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[22] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[23] F. Preparata and D. Sarwata. An improved parallel processor bound in fast matrix inversion. *Inform. Proc. Lett.*, 1978.

[24] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, Boston, 2nd edition, 1997.

[25] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2nd edition, 2005.

[26] M. Stone. The theory of representation for boolean algebras. *Trans. Amer. Math. Soc.*, 1936.

[27] M. Stone. Applications of the theory of boolean rings to general topology. *Trans. Amer. Math. Soc.*, 1937.

[28] Q.-N. Tran. A fast algorithm for Groebner basis conversion and its applications. *Journal of Symbolic Computation*, 30:451–468, 2000.

[29] Q.-N. Tran. A new class of term orders for elimination and applications. *Journal of Symbolic Computation*, 42, 2007.

[30] Q.-N. Tran and M. Y. Vardi. Groebner bases computation in boolean rings for symbolic model checking. In *Proceedings of IASTED 2007 Conference on Modeling and Simulation*, 2007.

[31] F. Winkler. *Polynomial Algorithms in Computer Algebra*. Texts and Monographs in Symbolic Computation. Springer-Verlag, 1996.