Fraunhofer Institut
Techno- und
Wirtschaftsmathematik

PolyBoRi: A Gröbner Basis Framework for Boolean Polynomials

Michael Brickenstein[1]

Alexander Dreyer[2]

[1] Mathematisches Forschungsinstitut Oberwolfach, Schwarzwaldstr. 9-11, 77709 Oberwolfach-Walke, Germany

[2] Fraunhofer Institute for Industrial Mathematics (ITWM), Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

Abstract

This work presents a new framework for Gröbner basis computations with Boolean polynomials. Boolean polynomials can be modeled in a rather simple way, with both coefficients and degree per variable lying in $\{0, 1\}$. The ring of Boolean polynomials is, however, not a polynomial ring, but rather the quotient ring of the polynomial ring over the field with two elements modulo the field equations $x^2 = x$ for each variable x. Therefore, the usual polynomial data structures seem not to be appropriate for fast Gröbner basis computations. We introduce a specialized data structure for Boolean polynomials based on zero-suppressed binary decision diagrams (ZDDs), which is capable of handling these polynomials more efficiently with respect to memory consumption and also computational speed. Furthermore, we concentrate on high-level algorithmic aspects, taking into account the new data structures as well as structural properties of Boolean polynomials. For example, a new useless-pair criterion for Gröbner basis computations in Boolean rings is introduced. One of the motivations for our work is the growing importance of formal hardware and software verification based on Boolean expressions, which suffer – besides from the complexity of the problems – from the lack of an adequate treatment of arithmetic components. We are convinced that algebraic methods are more suited and we believe that our preliminary implementation shows that Gröbner bases on specific data structures can be capable to handle problems of industrial size.

Keywords: Gröber basis, formal verification, Boolean polynomials, algebraic cryptoanalysis, satisfiability

1 Introduction

Gröbner bases have become a standard tool for treating problems which can be described by polynomial systems. While the concept of Gröbner bases is known much longer, their current practical importance is a result of dramatical improvements in performance and algorithms in recent years. It has also been shown, that a specialized implementation can often tackle much harder problems, like Faugère's HFE-attacks (2003). The motivation for our work was to provide a framework for computations in the following special but nevertheless important case of polynomials: coefficients lie in the field with two elements and exponents are bounded to degree one in each variable. This degree bound usually originates from the application of field equations of the form $x^2 = x$. As mentioned above, this occurs in many significant applications like formal verification but also in cryptography, logic, and many more. This is due to the fact that Boolean polynomials correspond to Boolean functions.

Although Gröbner bases have already become a standard tool for treating polynomial systems, current implementations have not been capable of satisfactorily handling Boolean polynomials from real-world applications yet. One of the first questions was: Can we use the simplified model to get better data structures? Of course, we did also ask, whether we can find algorithmic improvements of the situation.

The role of PolyBoRi in this context is to provide a framework of high performance data structures and example Gröbner bases algorithms. On the other hand it is very clear, that many problems arising from practice can only be tackled, if optimization occurs on many levels: data structures, higher level algorithms, formulation of equations/problems, good monomial orderings...

An important aspect in symbolic computation is that independent of the strategy polynomials can become very big, but usually keep structured (in a very general sense). Using this structure to keep the memory consumption moderate was a primary design goal of PolyBoRi. Another observation is that in Gröbner bases computations often arithmetical operations on similar polynomials (differing only in a few terms) occur. PolyBoRi also gives an answer to that problem using a cache mechanism on the level of substructures.

Even though it is not essential for the present paper, the reader may be interested in the following short description of one important application: **Formal verification** is a key challenge during the design process of digital systems. The goal is to have an automated and dependable way of finding errors in a given layout, before a prototype is built. See also (McMillan, 1993; Hachtel and Somenzi, 1996; Kunz et al., 2002) for more details.

Classical methods for design validation include the simulation of the system with respect to suitable input stimuli as well as tests based on emulations, which use simplified prototypes. The latter may be constructed using field programmable gate arrays (FPGAs). Due to a large number of possible settings, these approaches cannot cover the overall behavior of a proposed implementation. In the worst case, a defective system is manufactured and delivered, which might result in a major product recall.

In contrast, formal verification methods are based on *exact* mathematical methods for automated proving of circuit properties. In this context several approaches like SAT-solving, graph representation of Boolean functions, and (timed) finite automata are already in use for bringing a designer's concept into agreement with the required specifications. Here, formal methods have the ability to disclose unexpected sideeffects early in the design process, and also they may show that certain short-hand assumptions are really true for all input patterns and states.

The ability of checking the validity of a proposed design restricts the design itself: a newly introduced design approach may not be used for an implementation as long as its verification cannot be ensured. In particular, this applies to digital systems consisting of combined logic and arithmetic blocks, which may not be treated with specialized approaches. Here, dedicated methods from computer algebra may lead to more generic procedures, which help to fill the design gap.

Following, we start with a motivation of suitable data structures for handling of Boolean polynomials and continue with some mathematical background. Then we give a brief description of the PolyBoRi framework and the implemented algorithms. Finally, the treatment of some benchmark examples is compared with those of other computer algebra systems.

2 Boolean Polynomials as Sets

We are actually interested in **Boolean polynomials**, i. e. polynomials in the quotient ring $Q = \mathbb{Z}_2[x_1, \cdots, x_n]/\langle x_1^2 - x_1, \cdots, x_n^2 - x_n\rangle$. Hence, we deal with elements of the polynomial ring $P = \mathbb{Z}_2[x_1, \cdots, x_n]$ restricted by the *field equations*

$$x_1^2 = x_1,\, x_2^2 = x_2, \quad \cdots \quad , x_n^2 = x_n\,. \tag{1}$$

Under these conditions a polynomial $p \in Q$ can be written in its expanded form as

$$p = a_1 \cdot x_1^{\nu_{11}} \cdot \ldots \cdot x_n^{\nu_{1n}} + \ldots + a_m \cdot x_1^{\nu_{m1}} \cdot \ldots \cdot x_n^{\nu_{mn}} \tag{2}$$

with coefficients $a_i \in \{0, 1\}$. Furthermore, the constraints $x_i^2 = x_i$ in Equation 1 yield a *degree bound* on all variables of $\nu_{ij} \leq 1$. In particular, the latter can be restated as the condition $\nu_{ij} \in \{0, 1\}$.

Hence, a given Boolean polynomial p is defined by the fact, whether each term $x_1^{\nu_{i1}} \cdot \ldots \cdot x_n^{\nu_{in}}$ occurs in it. Analogously, the occurrences of the variables determine each term. One can assign a set $S_p = \{s_i, \cdots, s_m\}$ to p consisting of different subsets s_k, $s_i \neq s_j$ for $i \neq j$, of the variable vector $\{x_1, \cdots, x_n\}$. Then Equation 2 can be rewritten as

$$p = \sum_{s \in S_p} \left(\prod_{x_\nu \in s} x_\nu \right) \quad \text{with} \quad S_p = \{\underbrace{\{x_{i_1}, \cdots, x_{i_{n_1}}\}}_{s_1}, \cdots, \underbrace{\{x_{i_m}, \cdots, x_{i_{n_m}}\}}_{s_m}\}. \tag{3}$$

For practical applications it is reasonable to assume that S_p is sparse, i. e. the set is only a small subset of the power set over the variable vector. Even the s_i can be considered to be sparse, as usually quite few variables occur in a term. Consequently, the strategies of the used algorithms have to be tuned in such a way, that this kind of sparsity is preserved.

2.1 Zero-suppressed Binary Decision Diagrams

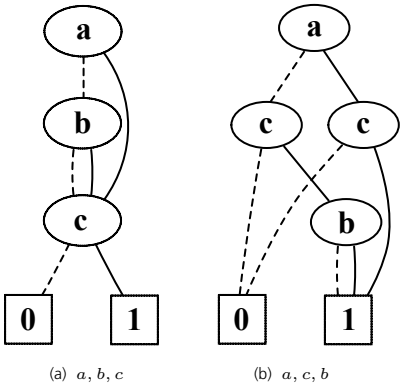
A **binary decision diagram** (BDD) is a rooted, directed, and acyclic graph with two terminal nodes $\{0, 1\}$ and decision nodes. The latter have two ascending edges (high/low or then/else), each of which corresponding to the assignment of true or false, respectively, to a given Boolean variable. In case, that the variable order is constant over all paths, we speak of an **ordered** BDD. For a more detailed treatment of the subject for instance see Ghasemzadeh (2005) and Bérard et al. (1999).

A series of connected nodes of a BDD starting at the root and ending at a terminal node is called **path**. We call a path **valid**, if it finishes at the terminal 1. Since any subset of the power set of the variables can be represented by the set of all valid paths of a suitable BDD, these diagrams are perfectly suited for the representation of Boolean polynomials suggested in Section 2.

For efficiency reasons it is useful to omit variables, which are not necessary to construct the whole set. A classic variant for this purpose is the **reduced-ordered BDD** (ROBDD, sometimes referred to as "*the* BDD"). These are ordered BDDs with equal subdiagrams merged, i. e. if some edges point to equivalent

subdiagrams, those are forced to point to the same diagram and share it. Furthermore, a node elimination is applied, if both descending edges point to the same node.

While the last reduction rule is useful for describing numerous Boolean-valued vectors, it is gainless for treating sparse sets. For this case, **zero-suppressed BDD**s (ZDD, also ZBDD or ZOBDD) have been introduced. They are also ordered and make use of subtree merging, but the node-elimination rule differs. Here, a node is removed if and only if its then-edge points to the 0-terminal. Figure 1 shows example ZDDs for a given Boolean polynomial. Note, that both construc-



(a) a, b, c (b) a, c, b

Figure 1 ZDD representing the polynomial $a\,c + b\,c + c$ for two different variable orders. Dashed/solid connections marking then/else-edges, respectively.

tions guarantee canonicity of resulting diagrams, see Ghasemzadeh (2005). But still the structure of resulting decision diagrams depends on the order of variables. In particular, the number of diagram nodes is highly sensitive to it, as Figures 1(a) and 1(b) illustrate. Therefore, a suitable choice of the order is always a crucial point, when modeling a problem using sets of Boolean polynomials.

2.2 State of the Art

Although graph-based approaches using decision diagrams for polynomials were already proposed before, those were not capable of handling algebraic problems efficiently. This was mainly due to the fact that the attempts were applied to very general polynomials, which cannot be represented as binary decision diagrams in a natural way.

For instance, the use of ZDDs for representing polynomials with integer coefficients can be found in Minato (1995). In this context coefficients and degrees had to be coded in a binary manner, which had lead to large diagram trees, even for rather small polynomials. Assuming bit length of m for each polynomial variable x_ν, a number of m decision variables has to be introduced in order

to represent $x_\nu^1, x_\nu^2, \cdots, x_\nu^{2^m}$. Arbitrary x_ν^n may be obtained by decomposing n into a sum of exponentials with respect to base 2. The same can be done to binary encode the coefficients. For instance, the polynomial $5\,x^2 + 2\,x\,y$ has to be decomposed into $x^2 + 2^2\,x^2 + 2^1\,x^1\,y^1$, with the new set of decision "variables" $\{x^2, x^1, y^1, 2^2, 2^1\}$.

In this general case addition and multiplication correspond to costly set operations involving de- and recoding of coefficient and degree numbers. Another reason, why ZDDs were not used in computer algebra before, is the importance of nontrivial monomial orderings. Usually, computer algebra systems store polynomials with respect to the current monomial ordering (Bachmann and Schönemann, 1998). This enables fast access to the leading term, and efficient iterations over all terms. In contrast, binary decision diagrams are ordered naturally in a lexicographical way. Fortunately, for special cases like the Boolean polynomials described in Section 2, it is possible to implement a search for the leading term and term iterators with suitable effort. Also, the special case of Boolean polynomials can be mapped to ZDDs more naturally, since the polynomial variables are in one-to-one correspondence with the decision variables in the diagram. The same applies for the polynomial arithmetic, which can efficiently be done using basic set operations. The PolyBoRi framework presented in this work is addressed to the utilization of this in a user-friendly environment.

Our approach can also be considered in the context of the meta approach of Coudert and Madre (1992). Boolean variables $x_1, \cdots, x_n$ yield 2^n possible configurations in $\{0,1\}^n$ for assigning true or false to each x_ν. Enumerating all valid solution vectors with respect to rather simple relations leads quickly to large and dense subsets of $\{0,1\}^n$. Since those sets cannot be handled efficiently, it had been suggested to store and manipulate the relations, which implicitly define the sets. In the language of computer algebra, the implicit relations are systems of Boolean polynomials. Hence, we can draw profit from the experience with Gröbner bases computations and heuristics for the treatment of polynomial systems. In addition, especially tuned strategies can be refined and developed when obeying the unique properties of Boolean rings.

3 Algebraic Basics

In this section, we recall some algebraic basics, including classical notions for the treatment of polynomial systems, as well as basic definitions and results from

computational algebra. For a more detailed treatment of the subject see the book of Greuel, G.-M. and Pfister, G. (2002) and the references therein.

3.1 Classical Notions

Let $P = K[x_1, \ldots, x_n]$ be the polynomial ring over the field K. A **monomial ordering** on P, more precisely, on the set of monomials $\{x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n} | \alpha \in \mathbb{N}^n\}$, is a well ordering "$>$" (i. e. each nonempty set has a smallest element with respect to "$>$") with the following additional property: $x^\alpha > x^\beta \Rightarrow x^{\alpha+\gamma} > x^{\beta+\gamma}$, for $\gamma \in \mathbb{N}^n$.

An expression λm ($\lambda \in K$, m a monomial) is called a **term** and λ the **coefficient**. An arbitrary element $f \in P$ is called a **polynomial**.

Let $f = \sum_\alpha c_\alpha \cdot x^\alpha$ ($c_{\alpha,i} \in K$) a polynomial. Then

$$\mathrm{supp}(f) := \{x^\alpha | c_\alpha \neq 0\}$$

is called the **support** of f.

Furthermore $\mathrm{lm}(f)$ denotes the **leading monomial** of f, the biggest monomial occurring in f w. r. t. "$>$" (if $f \neq 0$). The corresponding term is denoted by $\mathrm{lt}(f)$ and the coefficient by $\mathrm{lc}(f)$. Moreover, we set

$$\mathrm{tail}(f) := f - \mathrm{lt}(f).$$

If $F \subset P$ is any subset, $L(F)$ denotes the **leading ideal** of F, i. e. the ideal in P generated by $\{\mathrm{lm}(f) | f \in F \backslash \{0\}\}$. The S-Polynomial of $f, g \in P \backslash \{0\}$ with $\mathrm{lm}(f) = x^\alpha$, $\mathrm{lm}(g) = x^\beta$ is denoted by

$$\mathrm{spoly}(f, g) := x^{\gamma-\alpha} f - \frac{\mathrm{lc}(f)}{\mathrm{lc}(g)} x^{\gamma-\beta} g,$$

where $\gamma = \mathrm{lcm}(\alpha, \beta) := (\max(\alpha_1, \beta_1), \ldots, \max(\alpha_n, \beta_n))$. Recall that $G \subset P$ is called a **Gröbner basis** of an ideal $I \subset P$, if $\{\mathrm{lm}(g) | g \in G \backslash \{0\}\}$ generates $L(I)$ in the ring P and $G \subset I$.

Furthermore $V(I)$ denotes the common zeroes of a set of polynomials I.

Definition 3.1 (Standard representation) *Let $f, g_1, \ldots g_m \in P$, and let $h_1, \ldots, h_m \in P$. Then*

$$f = \sum_{i=1}^{m} h_i \cdot g_i \in K[x_1, \ldots, x_n],$$

is called a standard representation of f with respect to $g_1, \ldots, g_m$, if

$$\forall i : h_i \cdot g_i = 0 \text{ or otherwise } \mathrm{lm}(h_i \cdot g_i) \leq \mathrm{lm}(f).$$

The classical product criterion of Buchberger (Buchberger, 1985) reads as follows:

Lemma 3.2 (Product criterion) *Let $f, g \in K[x_1, \ldots, x_n]$ be polynomials. If the equality $\mathrm{lm}(f) \cdot \mathrm{lm}(g) = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$ holds, then $\mathrm{spoly}(f, g)$ has a standard representation w. r. t. $\{f, g\}$.*

Definition 3.3 (Elimination orderings) *Let $R = K[x_1, \ldots, x_n, y_1, \ldots y_m]$. An ordering ">" is called an elimination ordering of $x_1, \ldots, x_n$, if $x_i > t$ for every monomial t in $K[y_1, \ldots, y_m]$ and every $i = 1, \ldots, n$.*

3.2 t-Representations

There is an alternative approach to standard representations formulated in (Becker and Weispfennig, 1993) and used in (Faugère, J.-C., 1999), which utilizes the notion of t-representations. While this notion is mostly equivalent to using syzygies, it makes the correctness of the algorithms easier to understand.

Definition 3.4 (t-representation)
Let t be a monomial, $f, g_1, \ldots g_m \in P$, $h_1, \ldots, h_m \in P$. Then

$$f = \sum_{i=1}^{m} h_i \cdot g_i \in P$$

is called a t-representation of f with respect to $g_1, \ldots, g_m$ if

$$\forall i : \mathrm{lm}(h_i \cdot g_i) \leq t \text{ or } h_i \cdot g_i = 0\,.$$

Example 3.5

- *Let the monomials of P be lexicographically ordered ($x > y$) and let*

$$t = x^5y^5, g_1 = x^2, g_2 = x^5 - y, f = y$$

- *Then $f = x^3 g_1 - g_2$ is a x^5y^5-representation for f.*
- *Each standard representation of f is a $\mathrm{lm}(f)$-representation.*
- *For $t < \mathrm{lm}(f)$ t-representations of f do not exist.*

Notation: Given a representation $p = \sum_{i=1}^{m} h_i \cdot f_i$ with respect to a family of polynomials $f_1, \ldots f_m$, we may shortly say that p has a **nontrivial t-representation**, if a t-representation of p exists with

$$t < \max\{\mathrm{lm}(h_i \cdot f_i) | h_i \cdot f_i \neq 0\}.$$

For example, $\mathrm{spoly}(f_i, f_j)$ has a nontrivial t-representation if there exists a representation of $\mathrm{spoly}(f_i, f_j)$ where the summands have leading terms smaller than

$$\mathrm{lcm}(\mathrm{lm}(f_i), \mathrm{lm}(f_j)).$$

Theorem 3.6 *Let $F = (f_1, \ldots, f_k)$, $f_i \in K[x_1, \ldots, x_n]$, be a polynomial system. If for each f, $g \in F$ $\mathrm{spoly}(f, g)$ has a nontrivial t-representation w. r. t. F, then F is a Gröbner basis.*

Proof: For a full proof see (Becker and Weispfennig, 1993). A more sophisticated version of this theorem can be formulated and proven analogously to (Greuel, G.-M. and Pfister, G., 2002, p. 142). □

4 The PolyBoRi Framework

With PolyBoRi, we have implemented a C++ library for *Poly*nomials over *Bo*olean *Ri*ngs, which provides high-level data types for Boolean polynomials and monomials, exponent vectors, as well as for the underlying polynomial rings. The ring variables may be identified by their indices or by a custom string. Polynomial structures and monomials use ZDDs as internal storage type, but this is hidden from the user. The current implementation uses the decision-diagram management from CUDD (Somenzi, 2005). Its functionality is included using interface classes, which allows an easy replacement of the underlying BDD system without extensive rewriting of crucial PolyBoRi procedures.

In addition, basic polynomial operations – like addition and multiplication – have been implemented and associated to the corresponding operators. In order to enable efficient implementation, these operations were reformulated in terms of set operations, which are compatible with the ZDD approach. This also applies to other classical functionality like degree computation and leading-term computations. The ordering-dependent functions are currently available for lexicographical, degree-lexicographical (graded-lexicographical) ordering (with first variable being the largest one), and degree-reverse-lexicographical ordering, whereas in the latter case the variables are treated in reversed order for efficiency reasons. Product orderings consisting of blocks of these are currently at experimental state.

A complete Python (Rossum and Drake, 2006) interface allows for parsing of complex polynomial systems, and also sophisticated and easy extendable strategies for Gröbner base computations have been made possible by this. An extensive testsuite, which mainly carries satisfiability examples, and also some from cryptography, is used to ensure validity during development. Also, with the tool ipython the PolyBoRi data structures and procedures can be used interactively as a command line tool. In addition, routines for interfacing with the computer algebra system Singular (Greuel et al., 2005) are under development.

4.1 Polynomial Arithmetic

Boolean polynomial rings are motivated by the fact, that logical operations on bits can be reformulated in terms of addition and multiplication of $\mathbb{Z}_2$-valued variables. Representing polynomials as ZDDs these operations may also be implemented as set operations. E. g., adding the polynomials $p = \sum_{s \in S_p} \left(\prod_{x_\nu \in s} x_\nu \right)$ and $q = \sum_{s \in S_q} \left(\prod_{x_\nu \in s} x_\nu \right)$, with S_p and S_q as illustrated in Equation 3 (Section 2), is just $p + q = \sum_{s \in S_{p+q}} \left(\prod_{x_\nu \in s} x_\nu \right)$, where $S_{p+q} = (S_p \cup S_q) \backslash (S_p \cap S_q)$. Although each of these three operations is already available for ZDDs, it is usually more preferable to have them replaced by one specialized procedure. This avoids large intermediate sets (like $S_p \cup S_q$) and repeated iterations over both arguments. Algorithm 1 below shows a recursive approach for such an addition. Note

Algorithm 1 Recursive addition $h = f + g$

Require: f, g Boolean polynomials.
  if $f = 0$ **then**
    $h = g$
  else if $g = 0$ **then**
    $h = f$
  else if $f = g$ **then**
    $h = 0$
  else
    set $x_\nu = \mathrm{top}(f)$, $x_\mu = \mathrm{top}(g)$
    if $\nu < \mu$ **then**
      $h = \mathrm{ite}(x_\nu, \mathrm{then}(f), \mathrm{else}(f) + g)$
    else if $\nu > \mu$ **then**
      $h = \mathrm{ite}(x_\mu, \mathrm{then}(g), f + \mathrm{else}(g))$
    else
      $h = \mathrm{ite}(x_\nu, \mathrm{then}(f) + \mathrm{then}(g), \mathrm{else}(f) + \mathrm{else}(g))$
  return h

that $\mathrm{top}(p) = \min \bigcup_{s \in S_p} s$ denotes the variable associated to the root node of the current ZDD. Also, then- or else-branch of the latter correspond to polynomials referred to as $\mathrm{then}(p)$ and $\mathrm{else}(p)$, respectively. Since the indices of $\mathrm{top}(p), \mathrm{top}(q)$ are greater than i, the **if-then-else operator** $\mathrm{ite}(x_i, p, q) \equiv x_i \cdot p + q$ used here,

can just be generated by linking then- and else-branches of the new root node for x_i to p and q, respectively.

In a similar manner multiplication is given in Algorithm 2. The advantage of

Algorithm 2 Recursive multiplication $h = f \cdot g$

Require: f, g Boolean polynomials.
  if $f = 1$ **then**
    $h = g$
  else if $f = 0$ or $g = 0$ **then**
    $h = 0$
  else if $g = 1$ or $f = g$ **then**
    $h = f$
  else
    $x_\nu = \mathrm{top}(f)$, $x_\mu = \mathrm{top}(g)$
    if $\nu < \mu$ **then**
      set $p_1 = \mathrm{then}(f)$, $p_0 = \mathrm{else}(f)$, $q_1 = g$, $q_0 = 0$
    else if $\nu > \mu$ **then**
      set $p_1 = \mathrm{then}(g)$, $p_0 = \mathrm{else}(g)$, $q_1 = f$, $q_0 = 0$
    else
      set $p_1 = \mathrm{then}(f)$, $p_0 = \mathrm{else}(f)$, $q_1 = \mathrm{then}(g)$, $q_0 = \mathrm{else}(g)$
    $h = \mathrm{ite}(x_{\min(\nu,\mu)}, p_0 \cdot q_1 + p_1 \cdot q_1 + p_1 \cdot q_0, p_0 \cdot q_0)$
  return h

the recursive formulation is, that one easily can look up in a cache, whether the sum $f+g$, or the product $f \cdot g$, has already been computed before. The lookup can be placed in the beginning of the procedure, right after the trivial if-statements. Since this also applies to those subpolynomials, which are generated by $\mathrm{then}(f)$ and $\mathrm{else}(f)$, it is very likely, that common subexpressions can be reused. This holds even more, in the case of Gröbner base computations, in which likewise polynomials occur quite often. This is caused by those multiplication and addition operations used in Buchberger-based algorithms for elimination of leading terms and the tail-reduction process. Hence, while generating new Gröbner base elements the procedure results in summing up the same terms (up to some factor), which can be represented by combinations of subdiagrams of the original ZDDs.

4.2 Monomial Orderings

The operations treated in Section 4.1 are independent of the actual monomial ordering. Crucial for Gröbner algorithms is the computation of the **leading term** or **leading monomial**. Both concepts are equal in our context, and mean the largest monomial, with respect to the current "$<$"-relation. Lexicographically, the **leading monomial** is just the product of all node variables in the first valid path of the underlying ZDD, i. e. the sequence of nodes from the root down to the 1-leaf consisting of those nodes adjacent by then-branches only.

In case of degree orderings, one has to work harder. For instance, the leading monomial for the degree-lexicographical ordering can be found by iterating over all monomials (see Section 4.3) as follows: initially, degree and monomial of the first term are stored. If incrementing to the next term leads to a strictly higher degree, both – degree and monomial – are replaced by the current ones. This naïve approach does not make use of recursions, and hence it cannot be cached efficiently. A more suitable variant is given in Algorithm 3.

Algorithm 3 Recursive leading term and degree (degree-lexicographical)
$\{h, d\} = \mathrm{lead\_and\_deg}(f) = \{\mathrm{lead}(f), \deg(f)\}$

Require: f Boolean polynomial.
  if f is constant **then**
    $h = 1$, $d = 0$
  else
    $\{h_1, d_1\} = \mathrm{lead\_and\_deg}(\mathrm{then}(f))$, $\{h_0, d_0\} = \mathrm{lead\_and\_deg}(\mathrm{else}(f))$
    if $d_0 < d_1 + 1$ **then**
      $h = \mathrm{top}(f) \cdot h_1$, $d = d_1 + 1$
    else
      $h = h_0$, $d = d_0$
  return $\{h, d\}$

Sometimes the degree of a polynomial is cheap to compute, for instance, if an upper bound, like the *sugar* value discussed in Section 5.1 can be used, as Algorithm 4 illustrates. In any case, the number of ring variables may always be

Algorithm 4 Recursive degree computation $d = \deg(f, d_{\max})$ with upper bound

Require: f Boolean polynomial, $d_{\max}$ upper bound for degree
  if f is constant **then**
    $d = 0$
  else
    $d_1 = \deg(\mathrm{then}(f), d_{\max} - 1) + 1$
    if $d_1 = d_{\max}$ **then**
      $d = d_1$
    else
      $d = \max(d_1, \deg(\mathrm{else}(f), d_{\max}))$
  return d

used for such an upper bound. Also caching is useful, since immediately a single call of $\deg(f)$ makes $\deg(g)$ available on the cache for all recursively generated subpolynomials. Having such a kind of cheap $\deg$-functionality available, one can formulate Algorithm 5, which only generates the leading term, but not the other terms of the polynomial.

Note, that similar algorithms can be formulated for the degree-reverse-lexicographical ordering (with reversed variable order). For this purpose, the strict

Algorithm 5 Recursive leading term $h = \mathrm{lead}(f)$ (degree-lexicographical)

Require: f Boolean polynomial.
  if $\deg(f) = 0$ **then**
    $h = 1$
  else if $\deg(f) = \deg(\mathrm{then}(f)) + 1$ **then**
    $h = \mathrm{top}(f) \cdot \mathrm{lead}(\mathrm{then}(f))$
  else
    $h = \mathrm{lead}(\mathrm{else}(f))$
  return h

less-comparison in Algorithm 3 has to be replaced by *less or equal*, and in Algorithm 5, the else-branch has to be tested instead of the then-branch.

4.3 Iterators

PolyBoRi's polynomials also provide term access. For this purpose iteration over all monomials was implemented in the style of *Standard Template Library*'s (STL) iterators, obtained using `begin()` and `end()` member functions, like in Stepanov and Lee (1994). Very much like a generalization of the pointer concept, such a kind of **iterator** can be dereferenced to gain constant, i. e. read-only, access to the current term, and incremented to go to the next term in question. Also, comparison with other iterators of the same type is possible. In particular, equality with a special end marker yields the end of the iteration. This ensures compatibility with STL algorithms, originally designed for template classes like `std::vector` and `std::list`.

This kind of term iterator was implemented by a stack, which stores a sequence of references pointing to the diagram nodes. Initially, these are generated from following the first valid path. The resulting term is then stored using a temporary variable. Incrementing the iterator is equivalent to popping the top element from the stack as long as the corresponding nodes have invalid else-edges only. Then the subdiagram adjacent to this edge, and also its first valid path, is put on the stack, in order to represent the next lexicographical term. After popping/filling the temporary term value has to be updated subsequently.

In addition to the natural order of the underlying ZDD, iterators have been implemented for all supported monomial orderings. This hides the fact, that the internal data structure is actually ordered lexicographically. Hence, we have a sophisticated programming interface, which allows the formulation of general procedures in the manner of computational algebra, without the need for caring about certain properties of binary decision diagrams or the current ordering.

5 Algorithmic Aspects in Higher Level Computations

PolyBoRi implements basic polynomial arithmetic as well as higher level functions from computational algebra as Gröbner basis algorithms and normal form computations.

These algorithms from computational algebra have been adjusted to the facts that

- We have a very special situation: only coefficients 0 or 1, no exponents greater than 1.
- The framework can only represent Boolean polynomials (which is sufficient for the practice, since Boolean functions are equivalent to Boolean polynomials), but not general polynomials, in particular not the field equations themselves.
- Our data structures behave completely different, some operations are more costly, some are faster.

Paying attention to these points it is possible to achieve high performance using PolyBoRi.

5.1 Leading Terms

It is a common practice in computational algebra to have a degree bound (or *sugar* value, see Giovini et al., 1991) of intermediate polynomials, which can be generated using basic degree formulas, like

$$deg(f+g) \leq \max(deg(f), deg(g)) \,.$$

In PolyBoRi these degree bounds are of even greater use.

Even in degree orderings you can make use of them: Having the degree bound you can speed up leading-term calculations, having the leading term you can improve the degree bound (this is not the exact, original sugar strategy, but it behaves very useful in practice).

5.2 Normal Forms

A good example for this redesign of existing algorithms is the classical normal form algorithm:

Algorithm 6 Buchberger normal form

Require: G finite tuple of Boolean polynomials, f Boolean polynomial.
  while $f \neq 0$ and $\exists g \in G : \mathrm{lm}(g) | \mathrm{lm}(f)$ **do**
    $f := \mathrm{spoly}(f, g)$
  return f

An algorithm more suitable in PolyBoRi would be the following:

Algorithm 7 Greedy normal form

Require: G finite tuple of Boolean polynomials, f Boolean polynomial.
  while $f \neq 0$ and $\exists g \in G : \mathrm{lm}(g) | \mathrm{lm}(f)$ **do**
    $h := f / \mathrm{lm}(g)$ /* division by remainder, so the resulting terms correspond to terms in f divisible by the lead of g */
    $f := f - h \cdot g$ /* no term of f is divisible by $\mathrm{lm}(g)$ any more */
  return f

This last algorithm combines many small steps. The cost of the single steps can be higher using ZDD operations, but the combined step can be done much faster. The high cost (compared to classical polynomial representations) of these single additions might be surprising in the first moment, but can be explained quite easily. Good normal form strategies try to select a monomial for g, whenever possible. Then of course classical structures like linked list don't need a general addition, but can simply pop the first element (term) from the list. This can be done in constant time. In fact only applying this greedy technique to the case, where g is a monomial, already gives a quite good normal form implementation in PolyBoRi. Of course, it is a matter of heuristics to decide, when it might be better only to perform a single reduction step.

5.3 Gröbner Basis

The first real Gröbner basis algorithm implemented in PolyBoRi is an enhanced and specialized variant of the slimgb (Brickenstein, 2006), which was implemented first in Singular. Slimgb is a Buchberger algorithm, which was designed to reduce the intermediate expression swell. In particular it features a good strategy for elimination orderings (e. g. lexicographical orderings) using a special weighted length function, which not only considers the number of terms of a polynomial, but also their degree. We will concentrate in the presentation of the results on Gröbner bases computations, as there exists a large example set and it is a task, which is optimized in many systems.

5.3.1 Implementation Tricks

The availability of ZDDs for set operations can also used for other things than polynomial representation. For instance, having a polynomial p, the search for a polynomial q in your generations with the property, that $\mathrm{lm}(q)$ divides $\mathrm{lm}(p)$ can be implemented using set operations in the following way:

Algorithm 8 Search for reductor

Require: G Tuple of generating polynomials, each one has a different leading term, set of leading terms S, $\mathrm{lm2p}$ map (which maps a leading term to the corresponding ideal generator), polynomial $p \neq 0$
$S := \{\mathrm{lm}(g) | g \in G\}$
$\mathrm{lm2p} : S \to G$ /* map back leading terms to polynomials */
$t := \{s \in S | s \text{ divides } \mathrm{lm}(p)\}$ /* this last step can be implemented as a single ZDD Operation */
$D := \{\mathrm{lm2p}(s) | \text{ for } s \in t\}$
return D

This presented algorithm is supposed to be much faster than linear search, under the following (sensible) assumptions

- S is a large set
- each leading term in S is unique
- m has quite small degree compared to the number of variables
- D is small
- a call of $\mathrm{lm2p}$ has complexity $\mathcal{O}(\log_2(\#S))$
- $\mathrm{lm2p}$ is precomputed

This follows from the general principle, first to minimize the set of considered leading terms via set operations, and then to access the actual polynomial via a hash lookup. You can also use a similar technique, when applying the product criterion. There are many other possibilities to use the ZDDs for improving Gröbner basis computations.

5.3.2 Criteria

Criteria for keeping the set of critical pairs in the Buchberger algorithm small are central part of Gröbner basis algorithms. In most implementations the chain criterion and product criterion or variants of them are used.

These are of quite general type. This leads to the question, whether we can formulate new criteria for our particular case. There are two types of pairs to consider: Boolean polynomials with field equations, and pairs of Boolean polynomials. We concentrate on the first kind of pairs here.

Theorem 5.1 *Let f be a Boolean polynomial in $\mathbb{Z}_2[x_1, \ldots, x_n]$, $f = l \cdot g$, l a polynomial with linear leading term x_i, g a polynomial. Then* $\mathrm{spoly}(f, x_i^2 + x_i)$

has a nontrivial t-representation against the system consisting of f and the field equations.

Proof: First, we consider the case $g = 1$. In this situation the following formula holds: $\mathrm{lm}(f) = x_i$. Let r be a reduced normal form of $\mathrm{spoly}(f, x_i^2 + x_i)$ against f and the field equations. Then r is (tail) reduced, so it is a Boolean polynomial and irreducible against f, so x_i does not occur. In particular considered as a Boolean function it is independent from the value of x_i.

Since r is a linear combination of f and field equations (which are zero considered as Boolean functions) we get:

$$r(x_1, \ldots, x_n) = 1 \Rightarrow f(x_1, \ldots, x_n) = 1.$$

Now, we assume that $r \neq 0$. As a nonzero Boolean polynomial corresponds to a nonzero Boolean function, we know, that there exist $v_1, \ldots, v_n \in \{0, 1\}$, s. t. $g(v_1, \ldots, v_n) = 1$. The above implication gives, that $f(v_1, ..., v_n) = 1$.

Then we can change the value of x_i without affecting the value of r

$$r(v_1, \ldots, v_i + 1, \ldots, v_n) = 1,$$

but

$$f(v_1, \ldots, v_i + 1, \ldots, v_n) = 0,$$

as x_i only occurs in the one term x_i of f. This contradicts the above implication between r and f. So $r = 0$ and $\mathrm{spoly}(f, x_i^2 + x_i)$ has a standard representation.

Now, we consider a general Boolean polynomial g. $\mathrm{spoly}(l, x_i^2 + x_i)$ has a standard representation against l and the field equations:

$$\mathrm{spoly}(l, x_i^2 + x_i) = \sum_{j=1}^{n} h_j \cdot x_j^2 + x_j + \alpha \cdot l\,,$$

for polynomials α, h_j ($j \in \{1, \ldots, n\}$):

$$x_j^2 \cdot \mathrm{lm}(h_j) \leq \mathrm{lm}(\mathrm{spoly}(l, x_i^2 + x_i)) < x_i^2\,, \quad \mathrm{lm}(\alpha \cdot x_i) < x_i^2\,.$$

We multiply this equation by g and get by that fact, that x_i does not occur in g:

$$\begin{aligned} \mathrm{spoly}(l \cdot g, x_i^2 + x_i) &= \mathrm{spoly}(l \cdot g, g \cdot x_i^2 + x_i) - \mathrm{tail}(g) \cdot (x_i^2 + x_i) \\ &= g \cdot \mathrm{spoly}(l, x_i^2 + x_i) - \mathrm{tail}(g) \cdot (x_i^2 + x_i). \end{aligned}$$

Using the standard representation for $\mathrm{spoly}(l, x_i^2 + x_i)$ from above, both summands have a t-representation for a monomial $t < x_i^2 \cdot \mathrm{lm}(g)$, so we also get a nontrivial t-representation in the sum. □

Remark 5.2 *The polynomials l and g are indeed Boolean polynomials, as a Boolean polynomial only factors in Boolean polynomials (this can be seen using degree formulas). Together with the product criterion, we get, that we have only to consider pairs of Boolean polynomials f with field equations for variables x, which do not occur in an irreducible nonlinear factor of f. In the above proof, we make use of the fact, that we only consider well orderings, when claiming, that x_i does not occur in the tail of f.*

5.4 Gröbner Proof System

The Gröbner proof system (Clegg et al., 1996) is a combination of backtracking for calculation. Traditional SAT-solvers using backtracking split a logical expression into clauses, which have to be satisfied simultaneously (Kunz et al., 2002). This algorithm works the following way. On each level of the calculation a value for a chosen variable is plugged in. If even a single clause is unsatisfiable, then the system is obviously unsatisfiable. Then the other branch (the chosen value of the opposite variable) has to be checked.

The Gröbner proof system works similar to these classical SAT-solvers. The difference is, that the criterion for a system to be obviously unsatisfiable is that a run of the Buchberger algorithm with degree bound yields one (so the ideal is the whole ring). This algorithm has been implemented in a first experimental version. It will be a challenge for the future to find good strategies and heuristics for this very high level algorithm.

This section presents some benchmarks comparing PolyBoRi to general purpose and specialized computer algebra systems. Note, that it only presents the state of PolyBoRi in the development version at the end of December 2006. Since the project is very young we can expect major performance improvements for sure in the near future.

The following timings have been done on a AMD Dual Opteron 2.2 GHz (all systems have used only one CPU) with 16 GB RAM on Linux.

The used ordering was lexicographical ordering. PolyBoRi also implements degree ordering, but for the presented practical examples elimination orderings seem to be more appropriate. A recent development in PolyBoRi was the implementation of block orderings, which behave very natural for many examples.

We compared the following system releases

- Magma 2.13-8, command: GroebnerBasis, default options
- PolyBoRi CVS Dez 06, slimgb with default options

- Singular 3-0-3 (beta): slimgb, option(redTail) analogous to the default in PolyBoRi
- Maple 10.06 : Gröbner package, default options

We also tried the Maple interface to FGb (Faugère, J.-C., 2006), but the documentation didn't provide a way to use the lexicographical ordering, which we consider to be an appropiate ordering for these problems. Using a degree ordering in FGb we got worse results. In the spirit of a fair competition, we decided not to include FGb in our tables.

The examples were chosen from current research problems in formal verification and algebraic crypto analysis.

The basis for AES (small scale) attack was provided by Stanislav Bulygin (private communication). We made some optimizations on the formulation of the equations on it. The CTC example is due to Martin Albrecht (Albrecht, M., 2006). The systems describing the formal verification of multipliers were provided by Markus Wedler (private communication).

All timings of the computations are summarized in Table 1 below.

Example	Vars./Eqs.		PolyBoRi		Singular		Magma		Maple	
ctc-5-3	189	354	3.04 s	49 MB	32 s	69 MB	83 s	64 MB	>1800 s	>89 MB
ctc-8-3	297	561	4.8 s	52 MB	117 s	154 MB	817 s	335 MB		
ctc-15-3	549	1044	8.04 s	69 MB	748 s	379 MB	>3000 s	>570 MB		
aes-10-1-1-4pp	164	184	0.14 s	*	0.25 s		0.92 s	9.25 MB	>1000 s	
aes-7-1-2-4pp	204	255	3.24 s	50 MB	18 s		366 s	211 MB		
aes-10-1-2-4pp	288	318	6.7 s	51 MB	1080 s	694 MB	1007 s	476 MB	>70 h	>324 MB
mult4x4	55	48	0.01 s	*	0.01 s	0.7 MB	0.91 s	10 MB	0.99 s	9.8 MB
mult5x5	83	84	0.022 s	*	0.03 s	0.7 MB	31.5 s	44 MB	23.89 s	16 MB
mult6x6	117	106	0.047 s	*	0.169 s	2.9 MB	4582 s	1044 MB		
mult8x8	203	188	1 s	*	106 s	153 MB				
mult10x10	313	294	2.5 min	86 MB						

* too short to trace memory usage

Table 1 Timings and memory usage for benchmark examples

The authors of this article are quite convinced, that the default strategy of Magma is not well suited for these examples (walk, see Collart et al. (1997), or homogenization). However, when we tried a direct approach in Magma, it ran very fast out of memory (at least in the larger examples). So we can conclude, that the implemented GrÃ¶bner basis algorithm in PolyBoRi offers a good performance combined with low memory consumption. Part of the strength in directly computing GrÃ¶bner bases (without walk or similar techniques) is inherited from the slimgb algorithm in Singular. On the other hand our data structures provided a fast way to rewrite polynomials, which might be of bigger importance than sparse strategies in the presented examples.

While we used the normal slimgb algorithm for the presented examples, we were able to tackle much harder problems like 12BIT-Multiplier, AES small scale chiffre SR(10-1-2-8), SR(10-2-1-8), SR(10-2-2-4) using optimized scripts.

In this way the initial performance of PolyBoRi seems to be very promising. It can be seen, that the advantage of PolyBoRi grows with the number of variables. For many practical applications this size will even be bigger. We are very confident, that it will be possible to tackle some of these problems in future by using more specialized approaches. This is a key point in the development of PolyBoRi to facilitate problem specific, high performance solutions.

Acknowledgements

This work has been partly financed by the *Deutsche Forschungsgemeinschaft* (DFG) under Grand No. GR 640/13-1, and it has been supported by the Rheinland-Pfalz cluster of excellence *Dependable Adaptive Systems and Mathematical Modelling* (DASMOD). In addition, the authors thank Prof. Gert-Martin Greuel and Prof. Gerhard Pfister (both Department of Mathematics, University of Kaiserslautern, Germany) for their encouragement.

References

Albrecht, M., 2006. Algebraic Attacks on the Courtois Toy Cipher. Diplomarbeit, Universität Bremen.

Bachmann, O., Schönemann, H., 1998. Monomial Representations for Gröbner Bases Computations. In: Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'98). ACM Press, pp. 309–316.

Becker, T., Weispfennig, V., 1993. Gröbner bases, a computational Approach to Commutative Algebra. Graduate Texts in Mathematics, Springer Verlag.

Bérard, B., Bidoit, M., Laroussine, F., Petit, A., Petrucci, L., Schoenebelen, P., McKenzie, P., 1999. Systems and software verification: model-checking techniques and tools. Springer-Verlag New York, Inc., New York, NY, USA.

Bosma, W., Cannon, J., Playoust, C., 1997. The Magma algebra system I. Journal of Symbolic Computation, 24, 3/4, 235–265.

Brickenstein, M., 2006. Slimgb: Gröbner Bases with Slim Polynomials. In: Rhine Workshop on Computer Algebra. pp. 55–66, proceedings of RWCA'06, Basel, March 2006.

Buchberger, B., 1985. A Criterion for Detecting Unnecessary Reductions in the Construction of a Gröbner Basis. In: Bose, N. K. (Ed.), Recent trends in multidimensional system theory.

Buchberger, B., 1965. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal. Dissertation, Universität Innsbruck.

Cid, C., Murphy, S., Robshaw, M., 2005. Small Scale Variants of the AES. Fast Software Encryption - FSE2005, LNCS 3557, 145–162.

Clegg, M., Edmonds, J., Impagliazzo, R., 1996. Using the Groebner basis algorithm to find proofs of unsatisfiability. pp. 174–183.

Collart, S., Kalkbrener, M., Mall, D., 1997. Converting Bases with the Gröbner Walk. Journal of Symbolic Computation, 24, 465–469.

Coudert, O., Madre, J. C., 1992. Implicit and incremental computation of primes and essential primes of Boolean functions. In: Design Automation Conference. pp. 36–39.

Courtois, N., 2006. How fast can be algebraic attacks on block ciphers? Cryptology ePrint Archive, Report 2006/168.
URL `http://eprint.iacr.org/2006/168.pdf`

Faugère, J.-C., 2003. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: Advances in Cryptology - CRYPTO 2003, Lecture Notes in Computer Science 2729/2003. pp. 44–60.

Faugère, J.-C., 1999. A new Efficient Algorithm for Computing Gröbner Bases (F_4). Journal of Pure and Applied Algebra 139 (1–3), 61–88.

Faugère, J.-C., 2006. FGb/Maple interface.
URL `http://fgbrs.lip6.fr/salsa/Software/`

Ghasemzadeh, M., Nov. 2005. A new algorithm for the quantified satisfiability problem, based on zero-suppressed binary decision diagrams and memoiza-

tion. Ph.D. thesis, University of Potsdam, Potsdam, Germany.
URL http://opus.kobv.de/ubp/volltexte/2006/637/

Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C., 1991. One sugar cube, please or Selection strategies in Buchberger algorithms. In: Watt, S. (Ed.), Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computations, ISSAC'91. ACM press, pp. 49–54.

Greuel, G.-M., Pfister, G., Schönemann, H., 2005. SINGULAR 3.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern.
URL http://www.singular.uni-kl.de

Greuel, G.-M. and Pfister, G., 2002. A SINGULAR Introduction to Commutative Algebra. Springer Verlag.

Hachtel, G. D., Somenzi, F., 1996. Logic Synthesis and Verification Algorithms. Kluwer Academic.

Kunz, W., Marques-Silva, J., Malik, S., 2002. SAT and ATPG: Algorithms for Boolean decision problems, 309–341.

McMillan, K. L., 1993. Symbolic Model Checking. Kluwer Academic Publishers, Norwell, MA, USA.

Minato, S., Mar. 1995. Implicit manipulation of polynomials using zero-suppressed BDDs. In: Proc. of IEEE The European Design and Test Conference (ED&TC'95). pp. 449–454.

Rossum, G. V., Drake, F. L., November 2006. The Python Language Reference Manual. Network Theory Ltd., Bristol, United Kingdom.

Somenzi, F., 2005. CUDD: CU decision diagram package. University of Colorado at Boulder, release 2.4.1.
URL http://vlsi.colorado.edu/~fabio/CUDD/

Stepanov, A. A., Lee, M., 1994. The Standard Template Library. Tech. Rep. X3J16/94-0095, WG21/N0482.