Journal of Symbolic Computation 44 (2009) 1326–1345

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials☆

Michael Brickenstein[a,1], Alexander Dreyer[b]

[a] *Mathematisches Forschungsinstitut Oberwolfach, Schwarzwaldstr. 9-11, 77709 Oberwolfach-Walke, Germany*
[b] *Fraunhofer Institute for Industrial Mathematics (ITWM), Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany*

ARTICLE INFO

Article history:
Received 29 October 2007
Accepted 22 February 2008
Available online 13 February 2009

Keywords:
Boolean Gröbner basis
Formal verification
Algebraic cryptoanalysis
Satisfiability

ABSTRACT

This work presents a new framework for Gröbner-basis computations with Boolean polynomials. Boolean polynomials can be modelled 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 specialised data structure for Boolean polynomials based on zero-suppressed binary decision diagrams (ZDDs), which are 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 of handling problems of industrial size.

☆ Partly financed by the *Deutsche Forschungsgemeinschaft* (DFG) under Grand No. GR 640/13-1.
E-mail addresses: brickenstein@mfo.de (M. Brickenstein), alexander.dreyer@itwm.fraunhofer.de (A. Dreyer).
URLs: http://www.mfo.de (M. Brickenstein), http://www.itwm.fhg.de (A. Dreyer).
1 Tel.: +49 7834 979 31; fax: +49 7834 979 55.

0747-7171/$ – see front matter
doi:10.1016/j.jsc.2008.02.017

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 has been known much longer (Buchberger, 1965), their current practical importance is a result of dramatic improvements in performance and algorithms in recent years. It has also been shown, that a specialised implementation can often tackle much harder problems, like Faugère's attacks on hidden field equation (HFE) cryptosystems (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 $x^2 = x$. This occurs in many significant applications like formal verification, but also in cryptography, logic, and many more. This is due to the fact that those special polynomials – the so-called Boolean polynomials – correspond directly to Boolean functions.

Although the treatment of polynomial systems using Gröbner-bases improved considerably in recent years, current implementations have not yet been capable of satisfactorily handling Boolean polynomials from real-world applications. One of the first questions was: Can we use the properties of this simplified class of polynomials to obtain better data structures? Of course, we did also ask, whether we can find algorithmic improvements in this 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 optimisation 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 operations on similar polynomials (differing only in a few terms) occur quite often during Gröbner-basis computations. PolyBoRi also gives an answer to that problem using a cache mechanism on the level of polynomial 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) and 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 behaviour 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 side-effects 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 specialised 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 modelling expressions from propositional logic as polynomials over the finite field with two elements. Allowing values from {0, 1} only, the condition $x = x^2$ holds for all $x \in \mathbb{Z}_2$. Hence, we deal with elements of the polynomial ring $P = \mathbb{Z}_2[x_1, \ldots, x_n]$ restricted by the *field equations*

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

This leads to a degree bound of one on all variables, and therefore we can restrict ourself to a certain class of polynomials in the following.

Definition 1 (*Boolean Polynomials*). Let $p \in \mathbb{Z}_2[x_1, \ldots, x_n]$ be a polynomial, sth

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

with coefficients $a_i \in \{0, 1\}$ and $\nu_{ij} \in \{0, 1\}$. Then p is called a **Boolean polynomial**.

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 $S_p = \{s_1, \ldots, s_m\}$ to p consisting of different subsets $s_k, s_i \neq s_j$ for $i \neq j$, of the variable vector $\{x_1, \ldots, x_n\}$. Then Eq. (2) can be rewritten as

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

with indices $i_j \in \{1, \ldots, n\}$. In fact, there is a one-to-one correspondence between the set of Boolean polynomials in P and the set of all subsets of the power set of $\{x_1, \ldots, x_n\}$ via the mapping which is defined by $S_p \mapsto \sum_{s \in S_p} \left(\prod_{x_\nu \in s} x_\nu \right) = p$.

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 algorithms used have to be tuned in such a way, that this kind of sparsity is preserved.

In our context it is enough to treat Boolean polynomials only, as they are exactly the canonical representative of residue classes in the quotient ring of $\mathbb{Z}_2[x_1, \ldots, x_n]$ modulo the ideal of the field equations $\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.

2.1. Zero-suppressed binary decision diagrams

Binary decision diagrams (BDDs) are widely used in various areas as a unique representation of large sets, which could not be constructed efficiently by an enumerative approach.

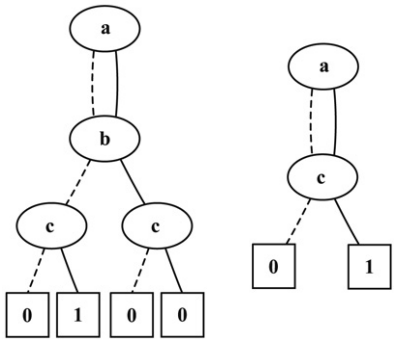
Definition 2 (*Binary Decision Diagram*). A **binary decision diagram** (BDD) is a rooted, directed, and acyclic graph with two terminal nodes {0, 1} and decision nodes. The latter have two ascending edges (high/low or then/else), each of which corresponding to the assignment of true or false, respectively, to a given Boolean variable.

A series of connected nodes of a BDD starting at the root and ending at the terminal node 1 is called a **path**. In case, that the variable order is constant over all paths, we speak of an **ordered** BDD.

This data structure is compact and easy to handle. For a more detailed treatment of the subject see Bryant (1986) and Bérard et al. (1999).

Definition 3. Let b be a binary decision diagram.

- The decision variable associated to the root node of b is denoted by $\mathrm{top}(b)$. Furthermore, $\mathtt{then}(b)$ and $\mathtt{else}(b)$ indicate the (sub-)diagrams, linked to then- and else-edge, respectively, of the root node of b.



(a) Non-canonical. (b) zero-suppressed.

Fig. 1. Different binary decision diagrams representing $a\,c + c$. Edges: `then` (—) and `else` (- -).

- For two BDDs b_1, b_0, which do not depend on the decision variable x, the **if-then-else operator** $ite(x, b_1, b_0)$ denotes the BDD c, which is obtained by introducing a new node associated to the variable x, sth $\texttt{then}(c) = b_1$, and $\texttt{else}(c) = b_0$.

The set of all valid paths of a binary decision diagrams can be used to represent a subset of the power set of the decision variables, which can be accessed easily, by following then- and else-edges. Since any Boolean polynomial p can be identified uniquely with such a subset, BDDs are perfectly suited for storing Boolean polynomials in a natural way. For instance, Fig. 1 shows some BDDs for $p = a\,c + c$. Unlike other BDD approaches, the diagram paths do not represent the valid solutions of the Boolean function behind p, but they form the sets $\{a, c\}$ and $\{c\}$ corresponding to polynomial terms directly.

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.

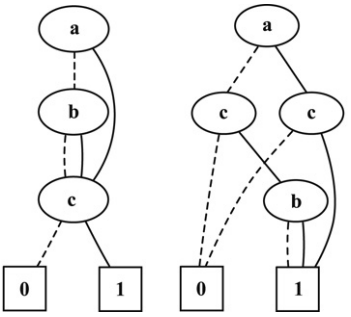
While the last reduction rule is useful for describing numerous Boolean-valued vectors, it is gainless for treating sparse sets. For this case, another variant, namely the ZDD (sometimes also called ZBDD or ZOBDD), has been introduced.

Definition 4 (*ZDD*). Let z be an ordered binary decision diagram with equal subdiagrams merged. Then z is called a **zero-suppressed binary decision diagram** (ZDD), if and only if those nodes are eliminated, whose then-edges point to the 0-terminal.

Note, in this case elimination means, that a nodes n is removed from the diagram and all edges pointing to it are linked to $\texttt{else}(n)$. Fig. 1 illustrates this reduction step of a given binary decision diagram for the polynomial $a\,c + c$. In Fig. 1(a) the then-edge of the right node with decision variable c is pointing to the 0-terminal. Hence, it can be safely removed, without losing information. As a consequence, the then-edge of the b-node is now connected to zero, and hence can also be eliminated. The effect of the complete zero-suppressed node reduction can be seen in Fig. 1(b). Note, that the construction guarantees canonicity of resulting diagrams, see Ghasemzadeh (2005). But still the structure of resulting decision diagrams depends on the order of variables. Fig. 2 shows example ZDDs for a given Boolean polynomial. In particular, the number of diagram nodes is highly sensitive to it, as Fig. 2(a) and (b) illustrate. Therefore, a suitable choice of the order is always a crucial point, when modelling a problem using sets of Boolean polynomials.

2.2. State of the art

Although graph-based approaches using decision diagrams for polynomials have already been proposed, they were not capable of handling algebraic problems efficiently. This was mainly due to



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

Fig. 2. ZDDs representing the polynomial $a\,c + b\,c + c$ for two different variable orders. Edges: `then` (—) and `else` (- -).

the fact that the attempts were applied to very general polynomials with integer coefficients and unbounded exponents, 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 a bit length of m for each polynomial variable x_ν, a number of m decision variables has to be introduced in order to represent $x_\nu^1, x_\nu^2, \ldots, x_\nu^{2^m}$. 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.

Apart from our own research, in recent times progress has been made in applying decision diagram methods to problems from computational algebra and cryptography. Chai et al. (2008) suggested ZDD-based data structures for improving a characteristic set method for solving Boolean equations. In addition, Michon et al. (2004) used BDDs for efficient handling of conjunctions of logical expressions for the analysis of HFE cryptosystems, but they did not utilise polynomial structures. In both cases, the use of decision diagrams resulted in significant improvements in performance and memory consumption.

Despite these efforts the important problem of dealing with nontrivial monomial orderings has not yet been resolved. 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 polynomial arithmetic, which can efficiently be done using basic set operations. The PolyBoRi framework presented in this work is addressed to the utilisation 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 $x_1, \ldots, x_n$ 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.

There exists a good theoretical basis for dealing with Boolean polynomial and field equations. In Hansen et al. (2006) the classical Gröbner-basis theory was adjusted to the settings of the quotient

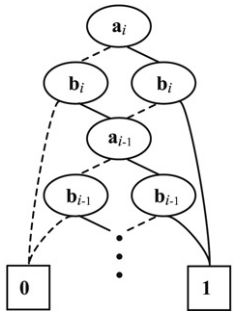


Fig. 3. ZDD structure representing the ith carry bit of an n-bit adder.

ring modulo the field polynomials. An equivalent approach considering ideals (which contain these field equations) in the polynomial ring was done in Brickenstein et al. (in press).

2.3. Modelling digital components

Here we present an application example, which is motivated from arithmetical blocks arising in digital systems design. An adder block unit takes two n-bit inputs a and b, and computes the sum s consisting of the lowest n bits of the addition result. In addition, another n bit output c gives a sequence of carry-overs.

In order to formulate the problem in terms of Boolean polynomials, the Boolean variables $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$ are introduced for bitwise handling of the inputs. Analogously, the sequences $s_0, \ldots, s_{n-1}$, and $c_0, \ldots, c_{n-1}$ are used to model the outgoing sums and carry bits. The system can be described by the following equations

$$\begin{aligned} c_i + \mathit{carry}_i &= 0 \\ s_i + a_i + b_i + \mathit{carry}_i &= 0 \end{aligned} \quad \text{for } i = 0, \ldots, n-1, \tag{4}$$

where the carry_i denote Boolean polynomials, which are recursively defined by setting $\mathit{carry}_{-1} = 0$ and $\mathit{carry}_i = a_i \cdot (b_i + \mathit{carry}_{i-1}) + b_i \cdot \mathit{carry}_{i-1}$. As already mentioned before, the size of a ZDD is very sensitive to the order of the variables. For efficient generation and handling of the ZDD structure it is crucial to use a variable order, which is motivated by the topology of the digital system. For this purpose, variables corresponding to outputs have to be placed before them, marking inputs. Additionally, obeying the recursive generation of carry_i the variables a_i and b_i have to be selected reverse alternating.

Hence, selecting $a_{n-1}, b_{n-1}, a_{n-2}, b_{n-2}, \ldots, a_0, b_0$ we obtain the ZDD structure of carry_i as shown in Fig. 3. In this case the polynomial of the most significant carry bit has $2^n - 1$ terms. The usual computational approach of storing the polynomial term-wise already causes memory problems for rather small n, even in case bit vectors are used as storage type. In contrast, the plaited structure of the ZDD leads to a linear growth of the number of nodes as the following theorem shows:

Theorem 5. *Let z_n be the ZDD, which corresponds to the Boolean polynomial carry_{n-1} in the variables $a_{n-1} > b_{n-1} > a_{n-2} > b_{n-2} > \cdots > a_0 > b_0$, where carry_i is recursively defined by*

$$\mathit{carry}_0 = a_0 \cdot b_0 \tag{5}$$

$$\mathit{carry}_i = a_i \cdot (b_i + \mathit{carry}_{i-1}) + b_i \cdot \mathit{carry}_{i-1}. \tag{6}$$

Then z_n has $3n - 1$ nodes.

Proof. If $n = 1$ the claim is trivial. For $n > 1$ Eq. (6) adds three new nodes to the ZDD structure of carry_{n-2}, which concludes the theorem. □

This enables fast polynomial generation and operations during Gröbner-base computations. In PolyBoRi it is possible to use such adder structures with thousands of bits almost instantaneously. Although it is very unlikely, that adder blocks with bit width larger that 128 will occur in a given design, such structures may occur in formal verification. For instance, this is the case when properties, which last for several clock cycles, are unrolled and reformulated in a time-independent way.

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 and Pfister (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

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

is called the **support** of f.

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

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

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

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

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

Definition 6 (*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

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

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

Lemma 7 (*Product Criterion*). *Let $f, g \in K[x_1, \ldots, x_n]$ be polynomials. If the equality $\operatorname{lm}(f) \cdot \operatorname{lm}(g) = \operatorname{lcm}(\operatorname{lm}(f), \operatorname{lm}(g))$ holds, then $\operatorname{spoly}(f, g)$ has a standard representation w.r.t. $\{f, g\}$.*

Definition 8 (*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 (1999), which utilises 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 9 (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 10. • Let the monomials of P be lexicographically ordered ($x > y$) and let

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

- Then $f = x^3g_1 - g_2$ 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 11. *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 and Pfister (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 also available.

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 test suite, 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. For instance, 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 Eq. (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 specialised procedure. This avoids large intermediate sets (like $S_p \cup S_q$) and repeated iterations over both arguments.

Algorithm 1 shows a suitable implementation of the Boolean polynomial addition. Since the indices

Algorithm 1 Recursive Boolean addition

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

of $\mathrm{top}(p)$, $\mathrm{top}(q)$ are greater than i, the if-then-else operator $\mathrm{ite}(x_i, p, q)$, which is equivalent to $x_i \cdot p + q$ here, can be implemented cheaply by linking then- and else-branches of the new root node for x_i to p and q, respectively.

The procedure also includes a cache lookup, immediately after the initial if-statements, which treat trivial cases only. The lookup can be implemented cheaply, because polynomials have a unique representations as ZDDs. Hence, previous computations of the sums of the form $f + g$ can be reused. The advantage of a recursive formulation is, that this also applies to those subpolynomials, which are generated by $\mathtt{then}(f)$ and $\mathtt{else}(f)$. Because of the recurring multiplication and addition operations, which are used in Buchberger-based algorithms for elimination of leading terms and the

tail-reduction process, it is very likely, that common subexpressions can be reused during Gröbner-base computation. Currently, the default settings of the underlying BDD library are used for cache-size and related parameters. Fine-tuning and optimisation in this area are subject of further research.

In a similar manner a kind of Boolean multiplication is given in Algorithm 2. The procedure

Algorithm 2 Recursive Boolean multiplication

Require: f, g Boolean Polynomials
Ensure: $h = f \star_2 g$
  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
    if isCached($\star_2, f, g$) **then**
      $h =$ cache($\star_2, f, g$)
    else
      $x_\nu = \mathrm{top}(f), x_\mu = \mathrm{top}(g)$
      if $\nu < \mu$ **then**
        set $p_1 = \mathtt{then}(f), p_0 = \mathtt{else}(f), q_1 = g, q_0 = 0$
      else if $\nu > \mu$ **then**
        set $p_1 = \mathtt{then}(g), p_0 = \mathtt{else}(g), q_1 = f, q_0 = 0$
      else
        set $p_1 = \mathtt{then}(f), p_0 = \mathtt{else}(f), q_1 = \mathtt{then}(g), q_0 = \mathtt{else}(g)$
      $h = \mathrm{ite}(x_{\min(\nu,\mu)}, p_0 \star_2 q_1 + p_1 \star_2 q_1 + p_1 \star_2 q_0, p_0 \star_2 q_0)$
      cache($\star_2, f, g$) $= h$
  return h

computes the unique representative of the product of two Boolean polynomials modulo the field equations. This multiplication is denoted by $\star_2$ in the following, while $\cdot$ means the usual multiplication. If variables of right- and left-hand side polynomials are distinct, both operations coincide.

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.

It was a major part of our research to make such functionality available for various nontrivial monomial orderings. In particular, this applies to such orderings, which obviously do not match with the natural order of the terms stored by a ZDD. By taking advantage of the cache and the uniqueness properties of the data structure is was possible to implement these functions with a reasonable small computational overhead.

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.

Sometimes the degree of a polynomial is cheap to compute, for instance, if an upper bound can be used. It is a common practice in computational algebra to have such a degree bound (or *sugar* value,

Algorithm 3 Recursive leading term and degree lead_and_deg(f) (degree-lex.)

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

see Giovini et al., 1991) arising from intermediate polynomials. It can be generated using basic degree formulas, like

$$\deg(f + g) \leq \max(\deg(f), \deg(g)). \tag{7}$$

In PolyBoRi these degree bounds are of even greater use. In degree orderings they can be utilised in a dual-purpose manner: 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 well in practice. Algorithm 4 illustrates the degree computation. In any case, the number n

Algorithm 4 Recursive degree computation with upper bound

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

of ring variables may always be used for such an upper bound, i.e. $\deg(f) = \deg(f, n)$. 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 – unlike Algorithm 3 – not the other terms of the polynomial.

Note, that similar algorithms can be formulated for the degree-reverse-lexicographical ordering with reversed variable order. The reason for the reversion of the variables (which can be hidden easily from the user) is that the implementation can be done more efficiently. For instance, for computing lead_and_deg w.r.t. this ordering one just has to replace the strict *less*-comparison in

Algorithm 5 Recursive leading term (degree-lexicographical)

Require: f Boolean polynomial.
Ensure: $h = \mathrm{lm}(f)$
  if $\deg(f) = 0$ **then**
    $h = 1$
    if isCached(lm, f) **then**
      $h = \mathrm{cache}(\mathrm{lm}, f)$
    else
      if $\deg(f) = \deg(\mathtt{then}(f)) + 1$ **then**
        $h = \mathrm{top}(f) \cdot \mathrm{lm}(\mathtt{then}(f))$
      else
        $h = \mathrm{lm}(\mathtt{else}(f))$
      $\mathrm{cache}(\mathrm{lm}, f) = h$
  return h

Algorithm 3 by *less or equal*. Analogously, in Algorithm 5, it has to be tested for $\deg(f) \neq \deg(\mathtt{else}(f))$ instead of $\deg(f) = \deg(\mathtt{then}(f)) + 1$.

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 generalisation 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. 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. On dereferencing a temporary monomial holding the current term is generated.

In addition to the natural order of the underlying ZDD, more complex 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 like 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. Normal forms

A good example for this redesign of existing algorithms is the classical normal form computation of Algorithm 6.

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

A more suitable approach in PolyBoRi would be the one of Algorithm 7.

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**
    $h := \frac{f}{\mathrm{lm}(g)}$ /* division by remainder */
    /* sth the result corresponds to terms in f divisible by $\mathrm{lm}(g)$ */
    $f := f - h \star_2 g$ /* no term of f is divisible by $\mathrm{lm}(g)$ any more */
  return f

The latter 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 do not 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.

The algorithm makes more sense in the case of Boolean polynomials. In fact, in the non-Boolean case a single reduction step can give worse results and would be harder from the computational point of view than a reduction step in the classical Buchberger normal form algorithm.

Example 12. We consider the polynomials $f = x^2 + x \cdot y$ and $g = x + y$ in $\mathbb{Q}[x, y]$ with lexicographical ordering $(x > y)$. Then $\mathrm{spoly}(f, g) = 0$, but

$$f + \frac{f}{\mathrm{lm}(g)} \cdot g = f - (x + y) \cdot g = -x \cdot y - y^2.$$

The purpose of the the construction was to cancel all multiples of $\mathrm{lm}(g)$ in f, which includes $x \cdot y$ in this example. Apparently the strategy fails in this general case.

The following theorem shows, that in the case of Boolean polynomials (and Boolean multiplication) the situation is better.

Theorem 13. *Let f, g be Boolean polynomials in $\mathbb{Z}_2[x_1, \ldots, x_n]$, $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$. Then*

$$r := f - \frac{f}{\mathrm{lm}(g)} \cdot g$$

does not contain any term, which is a multiple of $\mathrm{lm}(g)$: $\frac{r}{\mathrm{lm}(g)} = 0$. In particular, this claim holds for

$$r_2 := f - \frac{f}{\mathrm{lm}(g)} \star_2 g$$

as well. Additionally, $\mathrm{supp}(r) \not\supset \mathrm{supp}(\mathrm{spoly}(f, g))$, if $r \neq \mathrm{spoly}(f, g)$.

Proof. We only have to show that $\frac{f}{\mathrm{lm}(g)} \cdot \mathrm{tail}(g)$ does not contain multiples of $\mathrm{lm}(g)$, as $\frac{f}{\mathrm{lm}(g)} \cdot \mathrm{lm}(g)$ cancels with terms in f. W. m. a. $\mathrm{lm}(g) \neq 1$ (in this case $\mathrm{tail}(g) = 0$). Then by elementary properties of monomial orderings no term in $\mathrm{tail}(g)$ is divisible by $\mathrm{lm}(g)$. On the other hand every term in $\frac{f}{\mathrm{lm}(g)}$ does not contain a variable occurring in $\mathrm{lm}(g)$ (the maximal exponent per variable is 1), so multiplying with this terms does not contribute to divisibility by $\mathrm{lm}(g)$. This proves the first claim. The last claim can be shown analogously. □

5.2. Gröbner-basis

The first real Gröbner-basis algorithm implemented in PolyBoRi is an enhanced and specialised 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. We call the improved version for this special case symmgbGF2. 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 optimised in many systems.

5.2.1. 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).

Algorithm 8 Search for reductor

Require: $p \neq 0$ polynomial, G polynomials with $\mathrm{lm}(g) \neq \mathrm{lm}(h), \forall g, h \in G, g \neq h, S = \{\mathrm{lm}(g) | g \in G\}$,
$\mathrm{lm2p} : S \to G$ map with $\mathrm{lm}(g) \mapsto g, \forall g \in G$ /* leading term back to polynomial */
Ensure: D set of possible possible reductors
$t := \{s \in S | s \text{ divides } \mathrm{lm}(p)\}$ /* implementable 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 a 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 lm2p has complexity $\mathcal{O}(\log_2(\#S))$
- lm2p is precomputed.

This follows from the general principle, first to minimise the set of considered leading terms via set operations, and then to access the actual polynomial via a hash lookup. One 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.

5.2.2. Criteria

Criteria for keeping the set of critical pairs in the Buchberger algorithm small are a 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 14. *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 $v_1, \ldots, v_n \in \{0, 1\}$ subject to the condition $g(v_1, \ldots, v_n) = 1$. The above implication gives $f(v_1, \ldots, v_n) = 1$.

Then we can change the value of x_i without affecting the value of r, i.e.

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

but $f(v_1, \ldots, v_i + 1, \ldots, v_n) = 0$, since 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 15. 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.

5.3. 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.

6. Benchmarks and practical results

This section presents some benchmarks comparing PolyBoRi to general-purpose and specialised computer algebra systems. Note, that it only presents the state of PolyBoRi in the development version at the end of August 2007. 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, with the exception of FGb, where degree-reverse-lexicographic was used. 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 naturally for many examples.

We compared the following system releases with the development version of PolyBoRi's symmgbGF2 (for Magma, see Bosma et al., 1997):

Maple	11.01, June 2007	Gröbner package, default options
FGb	1.34, Oct. 2006	via Maple 11.01, command: fgb_gbasis
Magma	2.13-10, Feb. 2007	command: GroebnerBasis, default options
Singular	3-0-3, May 2007	std, option(redTail)

The examples were chosen from current research problems in formal verification and algebraic crypto-analysis. The latter yields systems of equations, which are comparable to those arising from formal verification, but they are more scalable. We use an algebraic attack to a small-scale variant of the *Advanced Encryption Standard* (AES) (National Institute of Standards and Technology, 2001). This approach is based on Cid et al. (2005) and was provided by Bulygin (private communication). We made some optimisations on the formulation of the equations in the formulation. This is done by an approach similar to the one of Toli and Zanoni (2004), but over GF2. Details will be published in a later article.

Furthermore, examples using *Courtois Toy Cipher* (CTC; see Courtois, 2006), which are due to Martin Albrecht (Albrecht, 2006), were tested. The systems describing the formal verification of multipliers were provided by Wedler (private communication).

All timings of the computations are summarised in Table 1 below. 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 homogenisation). 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 Gröbner-basis algorithm in PolyBoRi offers a good performance combined with low memory consumption. The relatively large offset of about 50 MB in memory usage is due to an initial memory pool for the cache management, whose size depends on the available RAM. Part of the strength in directly computing Gröbner-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 greater 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 12-BIT-Multiplier, AES small scale chiffre SR(10-1-2-8), SR(10-2-1-8), SR(10-2-2-4) using optimised scripts.

In order to treat classes of examples, for which the lexicographical ordering is not the best choice, PolyBoRi is also equipped with other monomial orderings. Although, its internal data structure is

Table 1
Timings and memory usage for benchmark examples.

Example	Vars./Eqs.		PolyBoRi		FGb		Maple		Magma		Singular	
			s	MB	s	MB	s	MB	s	MB	s	MB
aes_7_1_2_4_pp	204	225	0.15	53.82	2812.30	55.42	>1 h		345.82	206.22	27.41	25.77
aes_10_1_1_4_pp	164	184	0.02	52.70	1.97	32.69	2.99	70.48	0.28	9.03	0.02	1.75
aes_10_1_2_4_pp	288	318	0.22	55.59	>1 h		>1 h		847.36	471.58	183.73	55.86
ctc_3_3	117	117	0.04	52.80	6.56	16.66	10.40	67.96	1.62	10.39	0.77	5.25
ctc_5_3	189	189	0.10	63.05	602.20	30.68	929.77	231.39	84.65	67.79	3.63	14.75
ctc_8_3	297	297	0.15	64.23	>1 h		>1 h		758.96	299.08	14.32	33.31
ctc_15_3	549	549	0.28	68.71	>1 h		>1 h		>1 h		66.63	116.98
mult3 × 3	33	28	0.01	54.52	0.07	4.15	0.04	3.58	0.00	6.55	0.01	0.64
mult4 × 4	55	48	0.00	54.54	1.76	5.50	1.96	4.87	0.31	10.48	0.02	0.66
mult5 × 5	83	74	0.01	54.66	219.09	6.37	236.14	6.87	30.41	46.05	0.01	1.67
mult6 × 6	117	106	0.03	54.92	Failed		>1 h		>1 h		4.28	21.19
mult8 × 8	203	188	0.40	55.43	>1 h		>1 h		>1 h			>15 GB
mult10 × 10	313	294	18.11	85.91		>15 GB		>15 GB		>15 GB		>15 GB

Table 2
Timings and memory usage for benchmark examples, w. r. t. various orderings.

Example	Vars./Eqs.		Order	PolyBoRi		Magma		FGb	
				s	MB			s	MB
uuf50_8	50	218	lp	11.58	71.58	12.26	26.04		
			dlex	10.95	71.06	12.33	29.02		
			dp_asc	7.78	70.82	9.23	37.02	76.27	7.57
uuf75_8	75	325	lp	843.38	819.80	14 014.66	1633.62		
			dlex	553.43	490.86	14 290.90	2439.53		
			dp_asc	448.53	472.04	13 678.87	2539.24	99 721.46	8958.36
uuf100_01	100	430	lp	44 779.77	12 309.79	>2 days			
			dlex	11 961.86	6 101.43	>2 days			
			dp_asc	10 635.72	6 146.47	>2 days		Failed	

ordered lexicographically, the computational overhead of the degree orderings implementation is small enough, that the advantage of these orderings become effective. Table 2 illustrates this for a series of random unsatisfiable examples Hoos and Stützle (2000). The latter arise from benchmarking SAT-solvers, which can handle them very quickly, as their conditions are easy to contradict. But they are still a challenge for the algebraic approach. The strength of PolyBoRi is visible in the more complex examples, as it scales better than the other systems in tests. The computations of the these problems include a large number of generators, consisting of initially short polynomials, which leads to large intermediate results. The algorithmic improvement of symmgbGF2 and the optimised pair handling render the treatment of these example with algebraic methods possible.

In addition the performance of PolyBoRi is compared with the freely available SAT-solver MiniSat2 (release date 2007-07-21), which is state-of-the-art among publicly available solvers (Eén and Sörensson, 2003). The examples consist of formal verification examples corresponding to digital circuits with n-bit multipliers and the pigeon hole benchmark, which is the most important standard benchmark problem for SAT-solvers, e.g. used in Hoos and Stützle (2000), and some small-scale AES examples again. The latter checks whether it is possible to place $n+1$ pigeons in n holes without two of them being in the same hole (obviously, it is unsatisfiable).

Although, the memory consumption of PolyBoRi is larger, Table 3 illustrates, the computation time of both approaches is comparable for this kind of practical examples. (Again the first part was computed using preprocessing.) In particular, it shows, that in our research area the algebraic approach is competitive with SAT-solvers: the fast Boolean multiplication can be seen in the pigeon-hole benchmarks. In contrast, the good performance of the cryptography examples is due to fast elimination of auxiliary variables.

Finally, we used some standard benchmark problems to compare PolyBoRi's performance with that achieved by other systems. For this purpose, HFE cryptosystems of degree 96 in 25, 30, 35, 40, and 45 variables were treated (Steel, 2004). Table 4 shows, that PolyBoRi is able to solve these problems in a reasonable time. Magma shows better performance, in particular if specialised routines for HFE are used. PolyBoRi cannot play out its strengths here, since it is optimised for Boolean systems with several hundreds of variables. In this case, the main bottleneck is the linear algebra approach

Table 3
Comparison of PolyBoRi and MiniSat.

	Vars./Eqs.		PolyBoRi		MiniSat	
			s	MB	s	MB
hole6	42	133	0.13	51.64	0.01	1.96
hole7	56	204	0.46	51.89	0.06	1.96
hole8	72	297	1.88	56.59	0.30	2.08
hole9	90	415	8.01	84.04	2.31	2.35
hole10	110	561	44.40	97.68	25.20	3.24
aes_2_2_2_4	128	128	3.93	64.91	12.04	4.73
aes_3_2_2_4	184	184	46.67	88.77	76.88	6.77
aes_10_1_1_8	328	328	2.26	132.74	58.30	38.13
aes_10_2_2_4	576	576	1193.19	150.70	190.58	15.03
aes_7_1_2_4	204	225	0.15	53.82	0.38	5.16
aes_10_1_1_4	164	184	0.02	52.70	0.03	3.49
aes_10_1_2_4	288	318	0.22	55.59	0.44	7.28
mult3 × 3	33	28	0.01	54.52	0.00	1.96
mult4 × 4	55	48	0.00	54.54	0.00	1.95
mult5 × 5	83	74	0.01	54.66	0.01	1.95
mult6 × 6	117	106	0.03	54.92	0.03	1.95
mult8 × 8	203	188	0.40	55.43	0.96	2.21
mult10 × 10	313	294	18.11	85.91	22.85	3.61
aes_1_2_2_4	72	72	3.53	54.92	0.08	2.51
aes_3_1_2_4	104	104	0.48	55.42	0.23	2.98
aes_4_1_1_4	68	68	0.14	54.94	0.01	2.38
aes_4_1_2_4	120	120	0.88	55.44	0.32	3.27
aes_4_2_1_4	136	136	0.63	55.68	1.36	3.91
aes_5_1_2_4	148	148	1.29	55.69	0.40	3.58
aes_6_1_1_4	100	100	0.17	55.42	0.04	2.73
aes_6_1_2_4	176	176	1.55	55.98	0.64	4.05

Table 4
Timings and memory usage for hidden field equation problems (ordering is dp).

	PolyBoRi		FGb		Magma		Magma: HFE	
	s	MB	s	MB	s	MB	s	MB
HFE 25-96	75.64	244.16	359.03	12.95	16.80	153.88	3.46	20.89
HFE 30-96	378.52	626.13	2 065.06	22.33	55.46	780.66	13.12	48.14
HFE 35-96	2 625.89	3318.59	7 687.16	36.60	211.06	2 439.00	44.47	109.76
HFE 40-96	5 705.39	4430.30	23 352.74	57.37	588.43	5 914.81	110.36	233.79
HFE 45-96	11 620.94	7014.04	(Failed after 16 h)		1527.72	12 681.66	256.45	464.90

proposed by Bard (2006), which can be used to solve HFE-like problems. Currently PolyBoRi's implementation is less advanced than the corresponding Magma procedures.

7. Conclusions

In this paper the PolyBoRi framework was presented. It utilises special properties of Boolean systems of equations for computing Gröbner-basis over Boolean polynomials in an efficient way. A special data structure based on binary decision diagrams, and optimised procedures for polynomial operations were proposed. In addition, novel theoretical results were given, which have been used to improve the algorithmic aspects in higher levels of the Gröbner-basis computations.

On the one hand, PolyBoRi offers fast Boolean operations and elimination of auxiliary variables. On the other, the algorithmic improvement of symmgbGF2 and the optimised pair handling make it possible to treat problems with a large number of generators, which might yield large intermediate results.

It is also important to note, that various nontrivial monomial orderings were made available in an efficient way on a ZDD-based data structure. To our knowledge, this is a unique approach in area of Gröbner-basis computations, which has already paid off. Also, together with some valuable properties of binary decision diagrams it may lead to interesting new points of view in future research.

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 larger. We are very confident, that it will be possible to tackle some of these problems in the future by using more specialised approaches. Indeed, it was a key point during 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. Universität Bremen. Diplomarbeit.
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.
Bard, G.V., 2006. Accelerating cryptanalysis with the method of four Russians. Cryptology ePrint Archive, Report 2006/251 http://eprint.iacr.org/.
Becker, T., Weispfennig, V., 1993. Gröbner bases, a computational approach to commutative algebra. In: 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.
Brickenstein, M., Dreyer, A., Greuel, G.-M., Wedler, M., Wienand, O., New developments in the theory of Gröbner bases and applications to formal verification. In: Theoretical Effectivity and Practical Effectivity of Groebner Bases. Journal of Pure and Applied Algebra, in press (doi:10.1016/j.jpaa.2008.11.043) (special issue).
Bryant, R.E., 1986. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers 35 (8), 677–691.
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.
Bulygin, S., Small-scale variant (examples) of the Advanced Encryption Standard (AES), private communication.
Chai, F., Gao, X.-S., Yuan, C., 2008. A characteristic set method for solving Boolean equations and applications in cryptanalysis of stream ciphers. MM-Preprints 26. URL: http://www.mmrc.iss.ac.cn/pub/mm26.pdf/4-csz2-mm.pdf.
Cid, C., Murphy, S., Robshaw, M.J.B., 2005. Small scale variants of the AES. In: Gilbert, H., Handschuh, H. (Eds.), Fast Software Encryption 2005. In: Lecture Notes in Computer Science, vol. 3557. Springer-Verlag, pp. 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.
Eén, N., Sörensson, N., 2003. An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (Eds.), SAT. In: Lecture Notes in Computer Science, vol. 2919. Springer, pp. 502–518.
Faugère, J.-C., 2003. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In: Advances in Cryptology. CRYPTO 2003. In: Lecture Notes in Computer Science, vol. 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.
Ghasemzadeh, M., 2005. A new algorithm for the quantified satisfiability problem, based on zero-suppressed binary decision diagrams and memoization. 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., Pfister, G., 2002. A SINGULAR Introduction to Commutative Algebra. Springer Verlag.
Hachtel, G.D., Somenzi, F., 1996. Logic Synthesis and Verification Algorithms. Kluwer Academic.
Hansen, O.M., Michon, J.-F., 2006. Boolean Gröbner basis. In: Michon, J.-F., Valarcher, P., Yunès, J.-B. (Eds.). Proceedings of BFCA'06 Conference, March 13–15, 2006, Rouen, France. pp. 185–201.
Hoos, H.H., Stützle, T., 2000. SATLIB: An online resource for research on SAT. In: Gent, I.P., v. Maaren, H., Walsh, T. (Eds.), SAT 2000. IOS Press, pp. 283–292.
Kunz, W., Marques-Silva, J., Malik, S., 2002. SAT and ATPG: Algorithms for Boolean Decision Problems. p. 309–341.
McMillan, K.L., 1993. Symbolic Model Checking. Kluwer Academic Publishers, Norwell, MA, USA.
Michon, J.-F., Valarcher, P., Yunès, J.-B., 2004. HFE and BDDs: A practical attempt at cryptanalysis. In: Feng, K., Niederreiter, H., Xing, C. (Eds.), Coding Cryptography and Combinatorics. In: Progress in Computer Science and Applied Logic, vol. 23. Birkhäuser Verlag, pp. 237–246.

Minato, S., 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.
National Institute of Standards and Technology,, 2001. Advanced encryption standard (AES). FIPS-197, U.S. DoC/National Institute of Standards and Technology.
Rossum, G.V., Drake, F.L., 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/.
Steel, A., 2004. Allan Steel's Gröbner basis timings page.Website. URL: http://magma.maths.usyd.edu.au/users/allan/gb/.
Stepanov, A.A., Lee, M., 1994. The Standard Template Library. Tech. Rep. X3J16/940095, WG21/N0482.
Toli, I., Zanoni, A., 2004. An algebraic interpretation of AES-128. In: Dobbertin, H., Rijmen, V., Sowa, A. (Eds.), AES Conference. In: Lecture Notes in Computer Science, vol. 3373. Springer, pp. 84–97.
Wedler, M., Formal verification of multipliers (examples), private communication.