Network-driven Boolean Normal Forms*

[Preliminary Draft]†

Michael Brickenstein
Mathematisches Forschungsinstitut Oberwolfach
Schwarzwaldstr. 9-11
77709 Oberwolfach-Walke, Germany
brickenstein@mfo.de

Alexander Dreyer
Fraunhofer Institute for Industrial Mathematics
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
alexander.dreyer@itwm.fraunhofer.de

ABSTRACT

We apply the PolyBoRi framework for Gröbner bases computations with Boolean polynomials to bit-valued problems from algebraic cryptanalysis and formal verification.

First, we proposed zero-suppressed binary decision diagrams (ZDDs) as a suitable data structure for Boolean polynomials. Utilizing the advantages of ZDDs we develop new reduced normal form algorithms for linear lexicographical lead rewriting systems. The latter play an important role in modeling bit-valued components of digital systems.

Next, we reorder the variables in Boolean polynomial rings with respect to the topology of digital components. This brings computational algebra to digital circuits and small scale crypto systems in the first place. We additionally propose an optimized topological ordering, which tends to keep the intermediate results small. Thus, we successfully applied the linear lexicographical lead techniques to non-trivial examples from formal verification of digital systems.

Finally, we evaluate the performance using benchmark examples from formal verification and cryptanalysis including equivalence checking of a bit-level formulation of multiplier components. Before we introduced topological orderings in PolyBoRi, state of the art for the algebraic approach was a bit-width of 4 for each factor. By combining our techniques we raised this bound to 16, which is an important step towards real-world applications.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation, Algorithms

General Terms

Algorithms, Theory

Keywords

Gröbner, normal forms, Boolean polynomials, cryptanalysis, verification

*This work has been partly financed by the *Deutsche Forschungsgemeinschaft* (DFG) under Grant No. GR 640/13-2.

†Part of this work was done at the Dagstuhl Seminar 10271 – *Verification over discrete-continuous boundaries.*

1. INTRODUCTION

The central applications of this work are algebraic cryptanalysis and formal verification on the top of PolyBoRi. We developed the PolyBoRi framework for Gröbner bases computations with Boolean polynomials [8, 7]. The latter denote polynomials with coefficients in the field with two elements and a maximal degree of one per variable. They are in one-to-one correspondence with Boolean functions, which motivates the use of polynomial techniques for bit-valued problems from digital systems. In this context Bulygin and Brickenstein had improved the state of the art in algebraic cryptanalysis in [12]. They were able to attack small scale AES cipher [14] from 8 to 64 bits for two rounds. This article presents the optimized normal form functions, which form the basis of their algebraic attack.

First, we briefly describe zero-suppressed decision diagrams. They are suited for storing and manipulating Boolean polynomials, but yield a different complexity behavior of the basic polynomial routines. So, we had to reformulate central algorithms, like the reduced normal form in Section 3. This was already behind the scenes in [12].

Several research groups made considerable efforts to use Gröbner bases for formal verification and satisfiability. These past approaches focussed on various points: Tran and Vardi [29] applied ideal theory to symbolic model checking. Clegg, Edmonds, and Impagliazzo [15] combined Buchberger's algorithm [11] with backtracking. Condrat and Kalla [16] used it to preprocess small subsystems. In contrast, we start with decision diagrams from formal verification and continue with algorithmic research on Gröbner bases on the top of these. Finally, we present and analyze some timing results.

2. DECISION DIAGRAMS FOR BOOLEAN GRÖBNER BASES

In this section we fix the notation, which we will use in the algorithmic part of the paper. We give also a brief explanation of basics from computational algebra and a rough overview of zero-suppressed decision diagrams.

2.1 Algebraic Basics

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 see [3, 21] and the references therein.

Let $P = K[x_1, \ldots, x_n]$, K a field, and let its set of mono-

mials $\{x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n} | \alpha \in \mathbb{N}^n\}$ be equipped with a **monomial ordering** $>$. It is a well ordering, with the following additional property: if $x^\alpha > x^\beta$ then $x^{\alpha+\gamma} > x^{\beta+\gamma}$, for $\gamma \in \mathbb{N}^n$.

Let $f = \sum_\alpha c_\alpha \cdot x^\alpha$ ($c_\alpha \in \mathbb{Z}_2$) a polynomial. Then

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

is called the **support** of f.

If $f \neq 0$ then $\mathrm{lm}(f)$ denotes the **leading monomial** of f, the biggest monomial occurring in f with respect to ">". Moreover, we set $\mathrm{tail}(f) = f - \mathrm{lm}(f)$.

If $F \subset P$ is any subset, $\mathrm{L}(F)$ denotes the **leading ideal** of F, i. e. the ideal in P generated by $\{\mathrm{lm}(f) | f \in F \backslash \{0\}\}$.

We recall that a finite set $G \subset P$ is called a **Gröbner basis** of an ideal $I \subset P$, if $G \subset I$ and $\{\mathrm{lm}(g) | g \in G \backslash \{0\}\}$ generates $\mathrm{L}(I)$ in the ring P.

Let $f, h, g_1, \ldots g_m \in P$ and $G = \{g_1, \ldots g_m\}$ a Gröbner basis. We say, that h is the reduced normal form of f against G, if $h \equiv f \mod \langle G \rangle$ and $\mathrm{supp}(h) \cap \mathrm{L}(G) = \emptyset$.

The reduced normal form is unique and we denote it as $\mathrm{REDNF}(f, G)$. It does not depend on G itself, but only on the ideal spanned by G.

Given a **block ordering** or **product ordering** $>$ on P with two blocks is given by $1 < i_0 \leq n$ and monomial orderings $>_1$ on $K[x_1, \ldots, x_{i_0-1}]$ and $>_2$ on $K[x_{i_0}, \ldots, x_n]$ satisfying for all monomials $m_1, m_2 \in K[x_1, \ldots, x_{i_0-1}]$ and $n_1, n_2 \in K[x_{i_0}, \ldots, x_n]$: $m_1 \cdot m_2 > m_2 \cdot n_2$ if and only if $m_1 >_1 m_2$ or ($m_1 = m_2$ and $n_1 >_2 n_2$). Orderings consisting of arbitrary many blocks are constructed recursively.

A **Boolean ring** is not a polynomial ring, but isomorphic to the factor ring $\mathbb{Z}_2[x_1, \ldots, x_n] / \langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$. Every class of it has a unique representation as a polynomial in $\mathbb{Z}_2[x_1, \ldots, x_n]$ with a maximal degree of one per variable, called a **Boolean polynomial**.

2.2 Zero-suppressed decision diagrams

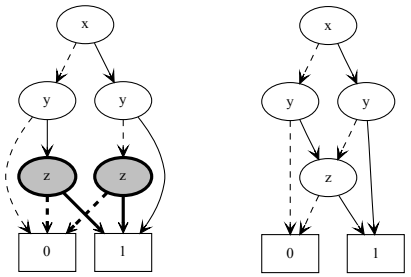
The idea of using binary decision diagrams for representing Boolean polynomials is straight-forward. Obviously, constant polynomials can be identified with the corresponding terminal node, respectively. Polynomials of positive degree contain at least one variable x. Since the degree per variable is most one, we can write a Boolean polynomial p in the form $p = x \cdot p_1 + p_0$. p_1 and p_0 are also Boolean polynomials, which do not depend on x anymore. Identifying x with a decision variable and assuming p_1 and p_0 to be representable as binary decision diagrams, we can recursively generate a diagram for p. We extensively elaborated the correspondence between decision diagrams, Boolean polynomials, and related objects in [8]. For decision diagram basics we refer to [10, 4].

Definition 2.1 (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 arcs (then and else), each of which corresponding to the assignment of true or false, respectively, to a given decision variable.*

*In case the variable order is constant over all paths starting at the root and ending at a terminal node, we speak of an **ordered** BDD.*

*Moreover, it is called **reduced**, if it contains no distinct subgraphs, that are isomorphic as ordered binary decision diagrams.*

*Furthermore, b^T (**then branch**) and b^E (**else branch**) indicate the (sub-)diagrams outgoing from the terminus of the then- and else-arc, respectively, of the root node of b (Figure 2(a)).*



(a) Not reduced (b) Reduced

Figure 1: Reduced and not reduced diagrams

A non-reduced binary decision diagram can be reduced as illustrated by Figure 1: The diagrams outgoing from the z-nodes in this Figure 1(a) are isomorphic. The correctly reduced version of this graph is the one in Figure 1(b).

The **decision variable** associated to a diagram node z is denoted by $\mathrm{var}(z)$ and the decision variable associated to the root node of a diagram b is denoted by **topvar(b)** .

Having the set $V = \{x_1, \ldots, x_n\}$ of all decision variables, we identify each variable x_i with its index i, and consequently, **top(b)** denotes the **index of the root node**. If the root node is a terminal node (0 or 1), we define $\mathrm{top}(b) = n + 1$.

For two BDDs b_1, b_0, which do not depend on the decision variable x, the **if-then-else operator** $\mathrm{ite}(x, b_1, b_0)$ denotes the BDD c, which is obtained by introducing a new node associated to the variable x, s. th. $c^T = b_1$, and $c^E = b_0$. Note, that the index of x has to be less than $\mathrm{top}(b_0)$, $\mathrm{top}(b_1)$.

Following, we describe a variant [27] which we use for representing Boolean polynomials.

Definition 2.2. *A reduced ordered decision diagram is called **zero-suppressed decision diagram** (ZDD), if no then arc leads to the zero terminal node.*

Definition 2.2 is illustrated in Figure 2(b). The introductory example from Figure 1(b) was extended by an artificial z-node, which is forbidden for ZDDs.

Definition 2.3. *Let b be a ZDD and let p be a directed path in b starting at the root node of b $n_1 = \mathrm{root}(b)$ ending with $n_k \in \{0, 1\}$.*

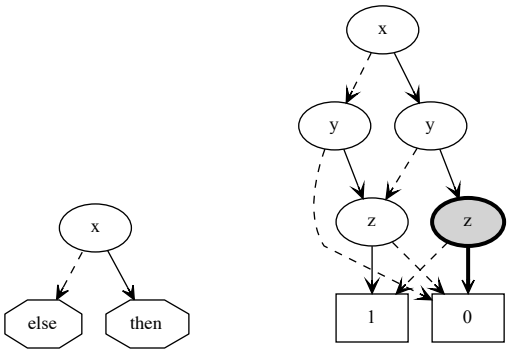
*Then the sequence p is called a **terminated path** of b. If $n_k = 1$, it is called a **valid path**.*

For a valid path p, we define

- *the **set of** p:* $\mathbf{set(p)} = \{\mathrm{var}(n_i) | a_i \in A_T\}$,
- *the **monomial** or **term** of p:* $\mathbf{term(p)} = \prod_{v \in \mathrm{set}(p)} v$.

We denote the set of all valid paths in b by $\mathrm{paths}(b)$. *We identify a ZDD with a **polynomial***

$$\mathbf{polynomial(b)} := \sum_{p \ valid \ path} \left(\prod_{v \in \mathrm{set}(p)} v \right).$$



(a) then- and else-branches (b) Then-arcs of ZDDs must not to point to 0

Figure 2: Basic decision diagram structures

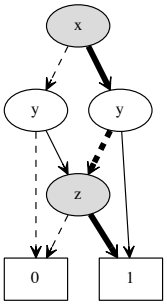


Figure 3: The monomial $x \cdot z$ is associated with the highlighted path. The complete diagram represents the polynomial $x \cdot y + x \cdot z + y \cdot z$.

Definition 2.3 means, that the set/monomial associated to a valid path only contains variables of nodes, where the path follows the outgoing **then edge**. From Figure 3 we see that the terms of a Boolean polynomials can be uniquely represented by the valid paths of a ZDD and vice versa.

It is also possible to refer to these diagrams as functional decision diagrams (FDDs, see [24]). While we store the term structure of a polynomial in a ZDD, it results in the same graph, when one stores the polynomial function in an FDD. From a computational algebra point of view, our description is more natural (and also used in [27, 13]). From a verification point of view the FDD is the more appropriate description.

3. LINEAR LEXICOGRAPHICAL LEAD REDUCTIONS

In applications like formal verification of digital circuits and cryptography it is important to calculate lexicographical normal forms against a polynomial system F. Often, the latter is of the form $F = \{f_i | i \in \{1, \ldots, m\}\}$, where the f_i are Boolean polynomials with pairwise different linear lexicographical leading terms $l_i = \mathrm{lm}(f_i)$, for some integer $m \leq n$. If $\deg(l_i) = 1 : \exists j_i$ with $l_i = x_{j_i}$, we call F a **linear lexicographical lead rewriting system**.

THEOREM 3.1. *A linear lexicographical lead rewriting system is a lexicographical Boolean Gröbner basis.*

The theorem can be proved analogously to the proof of [12, Theorem 3.1].

The normal form computation against F can be done in a recursive ZDD computation, while this is probably not the case for calculating general normal forms. We present algorithms for lexicographical normal forms against systems of Boolean polynomials with linear lexicographical leading terms. For the performance of the PolyBoRi framework, it was an essential step to find these algorithms.

The first problem we encounter is, that decision diagram based approaches usually deal with a fixed number of arguments, while F may contain arbitrary many arguments. So we need to encode all elements of F into a single ZDD. This can be done in the following way.

To simplify the presentation of the algorithm, we will assume, that $i_j < i_{j+1}$ for all $j \in \{1, \ldots, m-1\}$. As shown in Figure 4 we encode F by the decision diagram

$$\mathrm{ite}(x_{i_1}, \mathrm{ite}(x_{i_2}, \ldots \mathrm{ite}(x_{i_m}, 1, \mathrm{tail}(f_m)) \ldots, \mathrm{tail}(f_2)), \mathrm{tail}(f_1)) \,.$$

In this way, we can iterate over the leading terms of F by

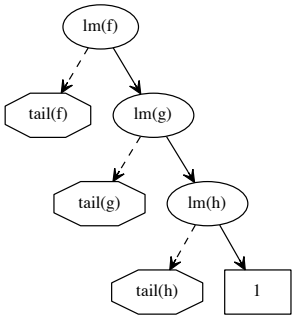


Figure 4: Encoding of system of polynomials with linear, lexicographical leading terms

always following the then-branch. The elements of F can be reconstructed by accessing the respective else-branch.

For this encoding of F we define the reduction algorithm for a Boolean polynomial against $F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$ in algorithm 1 (llnf). Note that $p \star q$ denotes the **Boolean**

Algorithm 1 llnf: lexicographical normal form against systems with linear leads

Input: p Boolean polynomial, F system of Boolean polynomials with pairwise different linear lexicographical leading terms encoded as described above
Output: result $\mathrm{REDNF}(p, F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\})$

```
if F = {1} or p ∈ {0,1} then
  return p
if top(p) > top(F) then
  return llnf(p, F^T)
if top(p) < top(F) then
  return topvar(F)⋆ llnf(p^T, F) + llnf(p^E, F)
/* now we know top(p) = top(F) */
return llnf(F^E, F^T)⋆ llnf(p^T, F^T) + llnf(p^E, F^T)
```

multiplication of two Boolean polynomials p and q:

$$p \star q = \mathrm{REDNF}(p \cdot q, \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}) \,.$$

It is again a Boolean polynomial.

There exists a variant of algorithm 1, which we call llnf*. It replaces the final return-statement, namely

$$\mathrm{llnf}(F^E, F^T) \star \mathrm{llnf}(p^T, F^T) + \mathrm{llnf}(p^E, F^T),$$

by $\mathrm{llnf}^*(F^E \star p^T + p^E, F^T)$. In the case of equal top variables this second variant recursively calls itself only once, instead of three times.

REMARK 3.2. *Algorithms operating directly on binary decision diagrams are usually recursive. Typically in each call* top *increases. Since it can only take $n+1$ distinct values, the recursion depth is bounded. So algorithms will always terminate (when using only recursion following this principle, conditions and terminating function call). Moreover, we can always prove correctness by induction on the minimal* top *of all function arguments.*

THEOREM 3.3. *Let F be as stated above, p a Boolean polynomial, then algorithm 1 terminates and returns a reduced normal form of p against $F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$.*

PROOF. For constant polynomials or the empty set the algorithm is obviously correct. Let $\deg(p) > 0$ and $F \neq \emptyset$. Since the recursive calls terminate (see Remark 3.2), the algorithm terminates by induction on $\min(\mathrm{top}(p), \mathrm{top}(F))$. We have to show: $\mathrm{llnf}(p, F)$ is reduced against $F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$ and lies in the same residue class like p modulo $F, \{x_1^2+x_1, \ldots, x_n^2+x_n\}$. Without loss of generality we assume $\mathrm{top}(p) = \mathrm{top}(F) = j_1$. We can further assume by induction, that the theorem holds for the recursive calls. Therefore,

$$\begin{aligned} \mathrm{llnf}(F^E, F^T) \star \mathrm{llnf}(p^T, F^T) + \mathrm{llnf}(p^E, F^T) &= \\ \mathrm{llnf}(\mathrm{tail}(f_1), \{f_2, \ldots, f_m\}) \star \mathrm{llnf}(p^T, \{f_2, \ldots, f_m\}) + & \\ \mathrm{llnf}(p^E, \{f_2, \ldots, f_m\}) &\equiv \\ x_{j_1} \cdot p^T + (p^E) &= p \end{aligned}$$

where the equivalence holds modulo $f_2, \ldots, f_m$ and the field equations $x_1^2 = x_1, \ldots, x_n^2 = x_n$. Hence, the result lies in the same residue class. Now, we show that it is reduced:

Since the recursive call returns polynomials which are reduced against $\{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$, they are Boolean polynomials and their addition and Boolean multiplication yields also Boolean polynomials (so reduced against $\{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$). It remains to show that the result of the algorithm is reduced against F. In a similar way, we use the fact that the recursive call returns normal forms of F^E, p^T, p^E against

$$\{f_2, \ldots, f_m, x_1^2 + x_1, \ldots, x_n^2 + x_n\}.$$

A polynomial p is reduced against $\{f_2, \ldots, f_m\}$, if and only if it does not involve any x_{j_k} for $k > 1$. So the sum and (Boolean) product of two such polynomials are again reduced against F. Since the arguments to the recursive calls do not involve any variable x_k with $k \leq j_1$, their reduced lexicographical normal form against $\{f_2, \ldots, f_m\} \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$ does not involve such variables either. In particular, it does not involve $x_{j_1} = \mathrm{lm}(f_1)$. Hence, these recursive results are actually reduced against

$$F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$$

and do not involve any x_{j_1} as well as their (Boolean) product and sum. □

In the case, that F is not only a (Boolean) Gröbner basis, but also reduced, we can simplify the algorithm by leaving out the reduction of the else branches of F. This idea is given in algorithm 2.

Algorithm 2 llnfredsb: lexicographical normal form against systems with linear leads

Input: p Boolean polynomial, F reduced Boolean Gröbner basis, encoded as above
Output: result $\mathrm{llnfredsb}(p, F \cup \{x_1^2 + x_1, \ldots, x_n^2 + x_n\})$

```
if F = {1} or p ∈ {0,1} then
  return p
if top(p) > top(F) then
  return llnfredsb(p, F^T)
if top(p) < top(F) then
  return topvar(F)⋆ llnfredsb(p^T, F) + llnfredsb(p^E, F)
/* now we know top(p) = top(F) */
return F^E⋆ llnfredsb(p^T, F^T) + llnfredsb(p^E, F^T)
```

4. TOPOLOGICAL ORDERINGS FOR DIGITAL SYSTEMS

We designed the PolyBoRi framework for computing Gröbner bases of systems derived from problems in verification of models for integrated circuits. Often, these systems feature a large number of variables and polynomial equations (several thousands and more). So it was a primary design goal to achieve a compact representation for this data. On the other hand, despite better properties of the case of Boolean polynomials [29], it is known that there exist systems with an even smaller number of variables (e. g. 50 variables), whose Gröbner bases are hard to compute [19].

First, we have a look at Boolean formulas, the kind of formulas defining Boolean functions.

DEFINITION 4.1. *We define a map ϕ from formulas in propositional logic to Boolean functions by providing a translation from the basis system* `not` *(¬),* `or` *(∨),* `true` *(*True*). For any formulas p, q we define the following rules*

$$\begin{aligned} \phi(p \vee q) &:= \phi(p) \cdot \phi(q) \\ \phi(\neg p) &:= 1 - \phi(p) \\ \phi(\mathrm{True}) &:= 0 \end{aligned} \tag{1}$$

Recursively, every formula in propositional logic can be translated into Boolean functions, as $\{\vee, \neg, \mathrm{True}\}$ forms a basis system in propositional logic (it generates the Boolean algebra).

4.1 Integrated circuits topology

We start applying computational algebra to combinatorial networks, i. e. networks containing no memory of previous states (flip-flops). Combinatorial networks consist of signals, which can be modeled as Boolean variables and logic gates defining a functional relation between these variables.

Each logic gate takes several input signals and transforms them into exactly one output signal. For example an OR-gate transforms two signals y, z into a third signal x by $x := y \vee z$. So the variable x depends on y and z. In some sense x is used to replace larger expressions. Using the algebraic mapping (see Definition 4.1) from propositional logic this dependency is equivalent to the Boolean polynomial

$f := x - y \cdot z$. We would like to have a monomial ordering, where the application of the reduced normal form of a Boolean polynomial g against the system consisting of f and the field polynomials replaces all occurrences of x by $y \cdot z$. So, we need to have $\mathrm{lm}(f) = x$ (or $x > y \cdot z$). For a lexicographical ordering this is equivalent to $x > y$ and $x > z$. In this way, each logic gate gives a condition for the monomial ordering. We call a monomial ordering **topological**, if it matches these conditions. A variable ordering is called topological, if the lexicographical ordering with respect to this variable ordering is topological. A topological ordering has not necessarily to be a lexicographical or elimination ordering. It can even be realized by a weighted degree ordering.

Usually there exist several topological variable orderings. Since every combinatorial network forms a directed **acyclic** graph, there always exists such a variable ordering. It can be determined by algorithm 3.

Algorithm 3 Calculating topological ordering on G,V

Input: G set of logic gates, V set of variables
Output: return tuple of variables in V topologically ordered w. r. t. G
  if $V = \emptyset$ **then**
    return V
  choose $v \in V$, where for all $g \in G$: v is not an input of g
  if $\exists g \in G$: v is the output of g **then**
    $G = G \backslash \{g\}$
  return (v) + topological_ordering$(G, V \backslash \{v\})$ /* concatenation of tuples */

THEOREM 4.2. *Algorithm 3 is correct and terminates*

PROOF. The only critical point of the algorithm is the validity of the "choose" statement. Initially we can choose just any signal and then follow the flow of signal until we reach some end point (which is not an input to a gate). This is a valid candidate for v. Here we use the fact, that the combinatorial network forms a directed acyclic graph, i. e. it contains no loops. It is easy to see, that taking v as biggest variable does not contradict any topology condition from the gates: If there exists a g with output v, then making v the biggest variable of all satisfies the condition that v is bigger than the gates input variables variables. The other gates/equations/polynomials do not involve v, so correctness can be satisfied recursively. Since in each recursion step $|V|$ decreases, there can only be finitely many recursion steps and hence the algorithm terminates. □

COROLLARY 4.3. *For each combinatorial network there exists a topological variable ordering.*

REMARK 4.4. *Topological orderings form the precondition for formal verification with computational algebra. For most other orderings like degree orderings or lexicographical orderings with a generic ordering of variables, it takes already seconds to do an equivalence check on a 4-bit multiplier and 6-bits turn out to be unfeasible. Using advanced techniques and (even optimized) topological orderings, we were able to verify a* 16×16*-bit multiplier in 24 seconds, as it is shown in Table 2. This is remarkable, as the the case of multipliers is very difficult. As stated by our project partners [30], using the industrial standard SAT-solvers, problems larger than* 10×10 *can hardly be handled. We want to remark at this point, that we indeed provide some extra information to the Gröbner basis/normal form algorithm in form of a topological ordering.*

The results on combinatorial networks can be generalized to sequential circuits (integrated circuits containing state information) in the following way: Sequential circuits problems can be mapped to propositional logic by using a bound t for the time steps (bounded model checking). A signal x is identified with t distinct Boolean variables. In this way we can construct a directed, acyclic graph and apply the techniques presented for combinatorial circuit in this section.

4.2 Optimized topological ordering

Algorithm 3 contains many choices. In general the topological ordering is not unique. Moreover, the choices can influence the strategy of the algorithms. Hence, the question arises, whether some topological orderings have better properties than others.

We recursively define the rank of a signal:

$$\mathrm{rank}(v) = \max(\{\mathrm{rank}(w) + 1 \,|\, w \in \mathrm{dep}(v)\} \cup \{0\}),$$

where $\mathrm{dep}(v)$ denotes the set of outputs depending on v:

$$\mathrm{dep}(v) = \{w \,|\, \exists \text{ gate } g : v \text{ is input of } g, w \text{ is output of } g\}.$$

There exists a wide variety of rank concepts, e. g. in [20] and [23]. Sometimes also the term "level" is used synonymously.

This reflects the usual behaviour, that term substitution blows up with every substitution step and we want to keep the intermediate results during the normal form computation as slim as possible.

It seems preferable to sort the variables in ascending order with respect to their rank.

THEOREM 4.5. *Let* $>$ *be an ordering on the variables, which is compatible with the* rank*-function:*

$$\forall i, j : \mathrm{rank}(x_i) > \mathrm{rank}(x_j) \Rightarrow x_i < x_j$$

Then $>$ *is a topological variable ordering.*

PROOF. If a variable or signal v forms an input value on the gate determining w as output-function, then we have

$$\mathrm{rank}(v) \geq \mathrm{rank}(w) + 1 \Rightarrow \mathrm{rank}(v) > \mathrm{rank}(w) \Rightarrow v < w,$$

which proves the claim by the definition of the topological variable ordering. □

THEOREM 4.6. *For each* i *from* 0 *to* d*, where* d *is given as* $d := max(\{\mathrm{rank}(x_i) | i \in \{1, \ldots, n\}\})$*, we set*

$$X_i = \{x_j | \, \mathrm{rank}(x_j) = i\}.$$

We define the following orderings:

- *On each set of variables* X_i *we define* $>_i$ *a monomial ordering for* $\mathbb{Z}_2[X_i]$*. On* $\mathbb{Z}_2[x_1, \ldots, x_n]$ *we define* $>_p$ *to be the product ordering of* $>_0, >_1, \ldots, >_d$*.*
- *Let* $>_h$ *be an arbitrary monomial ordering on the polynomial ring* $\mathbb{Z}_2[x_1, \ldots, x_n]$*. Let* $a, b \in \mathbb{Z}_2[x_1, \ldots, x_n]$*, and let* $\deg_{X_i}$ *denote the total degree in the variables contained in* X_i*. Then we define* $>_q$ *as follows:*

  In case $\deg_{X_i}(a) = \deg_{X_i}(b) \, \forall i$*,* $a >_q b :\iff a >_h b$*, otherwise* $a >_q b$ *if and only if* $\exists j \in \{0, \ldots, d\}$ *with* $\deg_{X_j}(a) > \deg_{X_j}(b)$ *and* $\deg_{X_i}(a) = \deg_{X_i}(b) \, \forall i < j$*.*

Then the orderings $>_p$, $>_q$ are topological.

PROOF. Let $> \in \{>_q, >_p\}$, g a gate, x_o the output of g, and $x_{k_1}, \ldots, x_{k_r}$ the inputs of g (for some r). Then for all $j \in \{1, \ldots, r\}$ we have, that

$$\operatorname{rank}(x_{k_j}) \geq \operatorname{rank}(x_o) + 1 > \operatorname{rank}(x_o).$$

Since $>$ has the elimination property for $X_{\operatorname{rank}(x_o)}$ restricted to $\mathbb{Z}_2[X_i | i \geq \operatorname{rank}(x_o)]$, x_o is bigger than every monomial and in $x_{k_1}, \ldots, x_{k_n}$. So the topological condition for g is fulfilled. □

Section 5 will show the practical meaning of the rank function and the optimized topological ordering.

4.3 Formal equivalence checking

Formal equivalence checking is a quite important part of the verification process. Given two integrated circuits, one has to prove, that they yield the same output on the same input signals [26].

This technique does not prove, whether the circuit actually does, what it is supposed to do. It just checks, that the output signals form the same Boolean function. Equivalence checking usually takes place after small design steps: For instance, after applying an optimization to the integrated circuit it is necessary to check, that the optimization did not alter the behaviour of the integrated circuits. Usually these modifications are quite subtle, s. th. major parts of the original design structure are preserved. In this way typical tools for equivalence checking use exactly these structural similarity to make this comparison very fast. On the other hand, usually these tools are not suited for comparing two circuits, that were constructed independently.

Algebraic methods can combine the advantages of the two methods to some extent. The circuits are usually given only with identified input variables. Of course, if we have polynomials $x + f$ and $y + f$ with leading monomials x and y, then we can replace all occurrences of x in the equations by y. This reduces the number of variables (using the similar structure) and makes further optimizations possible. In PolyBoRi, the `eliminate_identical_variables` option exists for applying this technique.

Usually equivalence checking is combined with other verification methods, that really verify the behaviour or properties of the respective circuits. This can consume a lot of time. Usually having proven these properties once, designers only need to check the equivalence of their work with the previous design. It can be done in a quite decent time by these specialized tools.

5. RESULTS

Following, we present some computational results for digital systems. We used is a Macbook Pro with a 2.5 GHz Intel Core 2 Duo and 4 GB RAM (utilizing only one core).

First, we apply the methods from Section 3 to linear lexicographical lead rewriting systems from formal verification and cryptanalysis. Then we repeat this using the optimized variable ordering of Section 4. We start with equivalence checking of classical multiplier designs as given in the text books [28] and [25]. Wedler [30] provided us with bit-level formulations of those components. In the following we call them *multiplier* $n \times n$, where n is the input bit-widths. The *viscoherencep* example is part of the AIGER distribution [5]. The remaining two examples are based on cryptanalysis problems: The *CTC* [17] example is due to Albrecht [1], while the *AES* example was provided by Stanislav Bulygin. Bulygin and Brickenstein [12] have formed several series of algebraic attacks on small scale AES/SR [14]. The reductions of Section 3 form an essential algorithmic ingredient thereof.

The rows of Table 1 correspond to the results for these examples. The columns of the table's left part present the computing time (in seconds) for Gröbner bases in PolyBoRi utilizing the proposed normal forms as well as the ordinary Buchberger algorithm. The latter essentially eliminates variables and computes the Gröbner basis in the remaining ones with little extra effort. The timings for algorithm 2, llnfredsb, also include the time needed to reduce the system F using the same reduction method.

The results show that there is no unique winner. Hence it is difficult to choose the optimal reduction strategy. Moreover, these big differences provide insight in the enormous importance of heuristics and a wide variety of functions specialized on different kinds of polynomial systems. In this sense, the PolyBoRi system integrates these optimized algorithms in the regular Gröbner bases functions as much as possible on a heuristic level.

On the one hand, we had showed in [9], that classical computer algebra systems like Magma [6] and Singular [22] cannot tackle even the 8-bit multiplier in reasonable time. On the other hand, PolyBoRi is competitive with the SAT solver MiniSat [18] for these applications [9]. In [8], we have demonstrated, that the situation is similar for small scale AES examples.

Furthermore we reordered the ring variables and computed all three linear lexicographical lead reduction algorithms a second time (see the right part of Table 1). Here an optimized variable ordering was generated, which preserves the leading terms of the system. This is described in more detail in Section 4.2. The time for optimizing the monomial ordering is included in the results.

We can see in both cryptographic examples (AES, CTC), that the optimized variable ordering does not seem to improve the situation. This was expected, since the examples are very regularly structured, the natural ordering of variables is nearly optimal. It is not clear, whether to prefer the variant computing a reduced Gröbner basis of F first. But clearly, llnf* performs usually much better using an optimized topological ordering.

For the equivalence checking examples from formal verification our experiments indicate, that we can prove the equivalence of two multipliers in a time, that is better than exponential in the number of input bits of the integrated circuit. Note that the total number of variables is still much higher, e. g. 768 variables for 2×16 input bits of the multiplier problem. Handling so many variables was only possible using an optimized topological monomial ordering and the llnf*-algorithm for normal form computation (Section 3).

Finally, we improved the performance of the llnf* approach for the multiplier examples using structural optimization. For this purpose we identify identical variables in both integrated circuits. Table 2 compares the runtime (in seconds) and the approximate input vector rate (per second) of this approach with the best previous results. This shows, that we improved the performance even more. It is highly probable, that we are able to handle even larger circuits.

Table 1: Linear lexicographical lead reduction

	simple topological ordering				optimized variable ordering		
	Buchberger	llnf	llnf*	llnfredsb	llnf	llnf*	llnfredsb
multiplier 8 × 8	0.65	3.27	12.19	3.19	0.66	0.09	0.53
multiplier 10 × 10	15	891.92	>7200	627.46	891	0.32	907
multiplier 16 × 16						76.05	
viscoherencep5	1.5	0.075	0.076	0.18	0.29	0.29	0.32
aes 10 1 2 4	4.3	1022.85	813	0.17	1112	3636	0.27
ctc Nr3 B5 n20	73	13.05	6.2	13.03	45	9.26	22.51

Table 2: Equivalence checking using llnf* in optimized ordering, without and with structural optimization

			original modeling		structurally optimized	
Example	input bits	variables	time	input vectors / time	time	input vectors / time
multiplier 8 × 8	16	204	0.09	730 000	0.07	940 000
multiplier 10 × 10	20	314	0.32	3 300 000	0.14	7 500 000
multiplier 16 × 16	32	768	76.05	56 000 000	23.65	180 000 000

Unfortunately, larger benchmark examples are not available to us in a suitable input format up to now. This is due to the fact, that bit-widths around 4 bits were state of the art for algebraic methods before we introduced topological orderings. In [9] we have experienced, that even using these orderings 6×6 is the bound for classical computer algebra systems, while we were still able to go up to 10×10 with the general routines of PolyBoRi. For this reason, equivalence checking of multipliers with 16 bits or more had not been considered as a meaningful benchmark then.

By applying the normal form of Section 3 against ZDD-encoded digital systems we lifted the bound to 16×16. Together with the optimizations from Sections 4.2 and 4.3, we got a time of 24 seconds on a our test system.

In Table 2 we also compared the number of possible input vectors, for which we verified the integrated circuit, with the number of seconds needed for the calculation. An almost constant quotient between the number of input vectors and computing time would be typical for a simulator and result in an exponential complexity of the algorithm. While we do not have any theoretical results about the asymptotic complexity of our method, it is a promising sign, that this quotient growths with a factor of 200 between the 8×8 and the 16×16 examples.

It is well-known, that FDDs for multipliers have an exponential number of nodes [2]. Our experiments, however, do not indicate exponential behavior. This can be interpreted as follows: Using the optimized topological ordering in connection with the llnf* algorithm seems to lead to the following desired effect. The proposed result from equivalence checking is the zero polynomial. At the same time, both circuits are expanded at similar signals and the terms of the polynomial can be extinguished early in the computation. Hence, our approach avoids to compute those large intermediate results.

6. CONCLUSIONS

Starting with a suitable data structure for Boolean polynomials we proposed specialized algorithms for computing reduced normal forms and topological variable orderings. We have implemented this in our framework PolyBoRi. Our implementation was used to evaluate the performance of various approaches on benchmark examples from digital systems. It shows, that we successfully applied algebraic methods to selected applications from formal verification and cryptanalysis.

In the case of equivalence checking of multiplier components, we were able to raise the bound from a bit-width of 4 to 16. Note, that for a 16×16 multiplier the symbolic computations verify the behavior for all $2^{32} = 4\,294\,967\,296$ input vectors, which marks a step towards real-world applications. Although we do not have a formal proof, our experiments indicate that we avoid the growth of intermediate results and do not lead to decision diagrams of exponential size for these problems. Similar improvements of the state of the art were achieved for the cryptanalysis problems.

Finally, we would like to remark, that this work is in no way specific for multipliers, but applicable to the bit-level verification of general integrated circuits and digital systems of likewise structure.

References

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

[2] B. Becker, R. Drechsler, and R. Werchner. On the relation between BDDs and FDDs. *Inf. Comput.*, 123(2): 185–197, 1995.

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

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

[5] A. Biere. AIGER, 2007. URL `http://fmv.jku.at/aiger/`. AIGER is a format, library and set of utilities for And-Inverter Graphs (AIGs).

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

[7] M. Brickenstein. *Boolean Gröbner bases – Theory, Algorithms and Applications*. PhD thesis, University of Kaiserslautern, Germany, 2010. to appear.

[8] M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, 44(9): 1326–1345, 2009. Effective Methods in Algebraic Geometry.

[9] M. Brickenstein, A. Dreyer, G.-M. Greuel, M. Wedler, and O. Wienand. New developments in the theory of Gröbner bases and applications to formal verification. *Journal of Pure and Applied Algebra*, 213(8):1612–1635, Aug. 2009. Theoretical Effectivity and Practical Effectivity of Gröbner Bases.

[10] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[11] B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Company, 1985.

[12] S. Bulygin and M. Brickenstein. Obtaining and solving systems of equations in key variables only for the small variants of AES. *Mathematics in Computer Science*, 2010.

[13] F. Chai, X.-S. Gao, and C. Yuan. A characteristic set method for solving boolean equations and applications in cryptanalysis of stream ciphers. *MM-Preprints*, 26, 2008.

[14] C. Cid, S. Murphy, and M. J. B. Robshaw. Small scale variants of the AES. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer-Verlag, 2005.

[15] M. Clegg, J. Edmonds, and R. Impagliazzo. Using the groebner basis algorithm to find proofs of unsatisfiability. *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing*, pages 174–183, 1996.

[16] C. Condrat and P. Kalla. A Gröbner basis approach to CNF-formulae preprocessing. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 618–631. Springer, 2007.

[17] N. Courtois. How fast can be algebraic attacks on block ciphers? *Cryptology ePrint Archive, Report 2006/168*, 2006.

[18] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

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

[20] E. R. Gansner, E. Koutsofios, and S. North. Drawing graphs with Dot. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, Jan. 2006. URL `http://www.graphviz.org/pdf/dotguide.pdf`.

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

[22] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 3-1-0 — A computer algebra system for polynomial computations, 2009. URL `http://www.singular.uni-kl.de`.

[23] D. Ilsen, E. Roebbers, and G.-M. Greuel. *Algebraic and Combinatorial Algorithms for Translinear Network Synthesis*, volume 55, pages 3131–3144. IEEE Circuits and Systems Society, 2008.

[24] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. *Design Automation, 1992. Proceedings., 3rd European Conference on*, pages 43–47, Mar. 1992.

[25] I. Koren. *Computer Arithmetic Algorithms*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[26] U. Krautz, M. Wedler, W. Kunz, K. Weber, C. Jacobi, and M. Pflanz. Verifying full-custom multipliers by Boolean equivalence checking and an arithmetic bit level proof. In *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, pages 398–403, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

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

[28] D. Patterson and J. Hennessy. *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2009.

[29] Q. Tran and M. Y. Vardi. Groebner bases computation in Boolean rings for symbolic model checking. In *MOAS'07: Proceedings of the 18th conference on Proceedings of the 18th IASTED International Conference*, pages 440–445, Anaheim, CA, USA, 2007. ACTA Press.

[30] M. Wedler. private communication, 2007.