POLYBORI - A Gröbner Basis Framework for Boolean Polynomials

Michael Brickenstein
Mathematisches Forschungsinstitut Oberwolfach

Alexander Dreyer*
Fraunhofer Institut für Techno- und Wirtschaftsmathematik (ITWM)

Oberseminar im Wintersemester 2007/2008
Kryptographie und Computeralgebra, TU Darmstadt, 31.1.2008

*Presentation
**Basic Data**

**POLYBORI**

*Poly*nomials over *Boolean* *Rings*

Website

http://polybori.sourceforge.net

DFG Project

“Development, implementation and application of mathematical-algebraic algorithms for formal verification of digital systems with arithmetic blocks”

Fraunhofer ITWM

Alexander Dreyer (Department Adaptive Systems)

University of Kaiserslautern

Algebra, Geometry and Computer Algebra Group (Prof. Greuel, Department of Mathematics)

Doctoral thesis

Michael Brickenstein (Mathematisches Forschungsinstitut Oberwolfach)
POLYBORI - A Gröbner Basis Framework for Boolean Polynomials

Overview

Optimization on many levels

Model
- topology of digital circuits
- translation into Boolean polynomials

Algorithm
- premade, optimized components
- one or several public/private key pairs for crypto
- suitable block orderings

Data Structure
- variable orderings
- specialized scripts
- F4
- Python
- C++

Polynomial
- Buchberger

ZDD
- Matrices
- PolyBoRi

Fraunhofer ITWM
Mathematisches Forschungsinstitut Oberwolfach
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Identification of fields

$\mathbb{GF}(2^r) \leftrightarrow \mathbb{GF}(2)[a]/\langle m \rangle =: F,$

irreducible $m \in \mathbb{GF}(2)[a]$ with $\deg m = r$
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Identification of fields

$$\mathbb{GF}(2^r) \leftrightarrow \mathbb{GF}(2)[a]/\langle m \rangle =: F,$$

irreducible $m \in \mathbb{GF}(2)[a]$ with $\deg m = r$

Identification of rings

$$\mathbb{GF}(2^r)[x_1, \ldots x_n] \leftrightarrow \mathbb{GF}(2)[a][x_1, \ldots x_n]/\langle m \rangle =: P$$

Monomorphism of $F$-algebras

$$\phi : P \rightarrow P_2, \quad x_i \mapsto \sum_{j=0}^{r-1} y_{i,j} \cdot a^j$$

$$P_2 = F[y_{1,0}, \ldots, y_{1,r-1}, y_{2,0}, \ldots, y_{2,r-1}, \ldots, y_{n,0}, \ldots, y_{n,r-1}]$$
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Identification of fields

$\mathbb{GF}(2^r) \leftrightarrow \mathbb{GF}(2)[a]/\langle m \rangle =: F$,
irreducible $m \in \mathbb{GF}(2)[a]$ with $\deg m = r$

Identification of rings

$\mathbb{GF}(2^r)[x_1, \ldots, x_n] \leftrightarrow \mathbb{GF}(2)[a][x_1, \ldots, x_n]/\langle m \rangle =: P$

Monomorphism of $F$-algebras

$\phi : P \rightarrow P_2, \quad x_i \mapsto \sum_{j=0}^{r-1} y_{i,j} \cdot a^j$

$P_2 = F[y_1,0, \ldots, y_1,r-1, y_2,0, \ldots, y_2,r-1, \ldots, y_n,0, \ldots, y_n,r-1]$

Canonical components $f_i$

Let $f \in P \Rightarrow \exists f_i \in \mathbb{GF}(2)[y_1,0, \ldots, y_1,r-1, y_2,0, \ldots, y_n,r-1] =: Q$
s. th. $\phi(f) = \sum_{i=0}^{r-1} a^i \cdot f_i$
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Canonal components\[ \phi(f) = \sum_{i=0}^{r-1} a^i \cdot f_i \]

Map to canonical components\[ \tau : P \rightarrow \text{PowerSet}(Q), \quad f \mapsto \{f_0, \ldots, f_{r-1}\} \]
Translation from $GF(2^r)$ to $GF(2)$

Canonical components

$$\phi(f) = \sum_{i=0}^{r-1} a^i \cdot f_i$$

Map to canonical components

$$\tau : P \rightarrow \text{PowerSet}(Q), \quad f \mapsto \{f_0, \ldots, f_{r-1}\}$$

Correspondence

$$G = \{g_1, \ldots, g_m\} \subset P \leftrightarrow H = \bigcup_{g \in G} \tau(f) \subset Q$$

Isomorphism of $GF(2)$-vector spaces

$$\lambda : F^n \rightarrow GF(2)^{r \cdot n} \quad (F = GF(2)[a]/\langle m \rangle \cong GF(2^r))$$

$$a^j \cdot e_i \mapsto E_{(i-1) \cdot r+j+1}$$
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Canonical components

$$\phi(f) = \sum_{i=0}^{r-1} a^i \cdot f_i$$

Map to canonical components

$$\tau : P \rightarrow \text{PowerSet}(Q), \quad f \mapsto \{f_0, \ldots, f_{r-1}\}$$

Correspondence

$$G = \{g_1, \ldots, g_m\} \subset P \leftrightarrow H = \bigcup_{g \in G} \tau(f) \subset Q$$

Isomorphism of $\mathbb{GF}(2)$-vector spaces

$$\lambda : F^n \rightarrow \mathbb{GF}(2)^{r \cdot n} \quad \quad (F = \mathbb{GF}(2)[a]/\langle m \rangle \cong \mathbb{GF}(2^r))$$

$$a^j \cdot e_i \mapsto E_{(i-1) \cdot r+j+1}$$

Theorem

$$\lambda(V(G \cup \text{FP}_P)) = V(H \cup \text{FP}_Q)$$

Field polynomials

$$\text{FP}_P = \{x_1^{2^r} - x_1, \ldots, x_n^{2^r} - x_n\},$$

$$\text{FP}_Q = \{y_{i,j}^{2^r} - y_{i,j} \mid 1 \leq i \leq n, 0 \leq j < r\}$$

Describe vanishing points over $\mathbb{GF}(2^r)$ and $\mathbb{GF}(2)$, resp.
Translation from $\mathbb{GF}(2^r)$ to $\mathbb{GF}(2)$

Disadvantages

– More equations ($r$ times as much)

Advantages

– Simpler base field $\mathbb{GF}(2)$

– Lower degree per variable (bound to 1, if FPQ is omitted)

– Boolean polynomial approach applicable (POLYBORI)
Digital systems verification and Boolean Polynomials

Interprete Boolean expressions as polynomials over $\mathbb{Z}_2$

Logical operations $\rightarrow$ arithmetical operations and
${\{\text{true, false}\}} \rightarrow \{0, 1\}$

$p \in \mathbb{Z}_2[x_1, \cdots, x_n]/\langle x_1^2 - x_1, \cdots, x_n^2 - x_n \rangle$
Digital systems verification and Boolean Polynomials

Interprete Boolean expressions as polynomials over $\mathbb{Z}_2$

logical operations $\rightarrow$ arithmetical operations and
\{true, false\} $\rightarrow$ \{0, 1\}

$p \in \mathbb{Z}_2[x_1, \cdots, x_n]/\langle x_1^2 - x_1, \cdots, x_n^2 - x_n \rangle$

Polynomials as sets

$p = a_1 \cdot x_1^{\nu_1} \cdots x_n^{\nu_n} + \cdots + a_2^n \cdot x_1^{\nu_{2n1}} \cdots x_n^{\nu_{2nn}}$

$= \sum_{s \in S_p} (\prod_{x_{\nu} \in s} x_{\nu})$, 
Digital systems verification and Boolean Polynomials

Interprete Boolean expressions as polynomials over $\mathbb{Z}_2$

Logical operations $\rightarrow$ arithmetical operations and \{true, false\} $\rightarrow$ \{0, 1\}

$$p \in \mathbb{Z}_2[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$$

Polynomials as sets

$$p = a_1 \cdot x_1^{\nu_1} \cdot \ldots \cdot x_n^{\nu_n} + \ldots + a_{2^n} \cdot x_1^{\nu_2^{n_1}} \cdot \ldots \cdot x_n^{\nu_2^{n_n}}$$

$$= \sum_{s \in S_p} (\prod_{x_\nu \in s} x_\nu),$$

with $S_p = \{\{x_{i_1}, \ldots, x_{i_n}\}, \ldots, \{x_{i_m}, \ldots, x_{i_{nm}}\}\}$

$$\subseteq \text{PowerSet}(x_1, \ldots, x_n)$$
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \( \{0, 1\} \), decision nodes (\( \equiv \) Boolean variables)
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \( \{0, 1\} \), decision nodes (\( \supseteq \) Boolean variables)

\[
\begin{array}{c}
X \\
\downarrow \\
y \\
\downarrow \\
z \\
\downarrow \\
0 \quad 1 \\
\end{array}
\]
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (\(\equiv\) Boolean variables)

Two descending edges per node (high/low or then/else)

\(\equiv\) assign variable to true/false
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (\equiv Boolean variables)

Two decending edges per node (high/low or then/else) 
\equiv assign variable to true/false

»Ordered« if the variable order is constant over all paths
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (\equiv Boolean variables)

Two descending edges per node (high/low or then/else)
\equiv assign variable to true/false

»Ordered« if the variable order is constant over all paths
**Zero-suppressed Binary Decision Diagrams**

**Binary Decision Diagram**

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (\Leftrightarrow\ Boolean variables)

Two descending edges per node (high/low or then/else)
\Leftrightarrow assign variable to true/false

»Ordered« if the variable order is constant over all paths

**Zero-suppressed BDD (ZDD)**

Node eliminated \iff then \rightarrow 0
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (⇔ Boolean variables)

Two descending edges per node (high/low or then/else)  
⇔ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated  ⇔ then → 0
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \{0, 1\}, decision nodes (⇔ Boolean variables)

Two descending edges per node (high/low or then/else)
⇔ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated ⇔ then → 0

Equal subgraphs merged
Zero-suppressed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes \( \{0, 1\} \), decision nodes (\( \cong \) Boolean variables)

Two decending edges per node (high/low or then/else)
\( \cong \) assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated \( \iff \) then \( \rightarrow 0 \)

Equal subgraphs merged
ZDDs for Sparse Sets

Example: \{"a, e\}
ZDDs for Sparse Sets

Example: \{\{a, e\}\}
ZDDs for Sparse Sets

Example: \{\{a, e\}\}
**Advantages of ZDDs**

**Idea**

ZDDs store term structure (not the Boolean function behind)
Advantages of ZDDs

Idea
ZDDs store term structure (not the Boolean function behind)

Reasons
– Compact data structure
Advantages of ZDDs

Idea

ZDDs store term structure (not the Boolean function behind)

Reasons

– Compact data structure

– Suitable for sparse and structured subsets of power sets
**Advantages of ZDDs**

**Idea**

ZDDs store term structure (not the Boolean function behind)

**Reasons**

- Compact data structure
- Suitable for sparse and structured subsets of power sets
- Polynomial structure recognizable
  - Paths $\cong$ polynomial terms
  - Natural path sequence $\cong$ lexicographical term ordering
Advantages of ZDDs

Idea
ZDDs store term structure (not the Boolean function behind)

Reasons
– Compact data structure
– Suitable for sparse and structured subsets of power sets
– Polynomial structure recognizable
  o Paths $\cong$ polynomial terms
  o Natural path sequence $\cong$ lexicographical term ordering
– Unique diagram nodes (less memory, reference counting)
– Caching of operations
Boolean Polynomials as ZDDs

Example

\[ xy + x + z \equiv \{ \{ x, y \} , \{ x \} , \{ z \} \} \]
Boolean Polynomials as ZDDs

Example

\[ xy + x + z = \{\{x, y\}, \{x\}, \{z\}\} \]
Boolean Polynomials as ZDDs

Example

\[ xy + x + z \equiv \{\{x, y\}, \{x\}, \{z\}\} \]
Boolean Polynomials as ZDDs

Example

\[ xy + x + z \equiv \{ \{x, y\}, \{x\}, \{z\} \]
**Polynomial Arithmetic**

Boolean polynomial operations correspond to set operations.
**Polynomial Arithmetic**

Boolean polynomial operations correspond to set operations.

**Example**

\[(x + xy) + (xy + z) = x + 2xy + z \equiv x + z \iff \]

\[(S_1 \cup S_2) \setminus (S_1 \cap S_2) = \{\{x\}, \{z\}\} \]

(with \(S_1 = \{\{x\}, \{x, y\}\}, S_2 = \{\{x, y\}, \{z\}\}\))
**PolyBoRi** - A Gröbner Basis Framework for Boolean Polynomials

**Polynomial Arithmetic**

Boolean polynomial operations correspond to set operations:

Example:

\[(x + x y) + (x y + z) = x + 2 x y + z \equiv x + z \iff\]

\[(S_1 \cup S_2) \setminus (S_1 \cap S_2) = \{\{x\}, \{z\}\}\]

(with \(S_1 = \{\{x\}, \{x, y\}\}, S_2 = \{\{x, y\}, \{z\}\}\))

\[x \cdot (y + z) = x y + y z \iff\]

\[\{\{x\} \cup \{y\}, \{x\} \cup \{z\}\} = \{\{x, y\}, \{x, z\}\}\]
### Polynomial Arithmetic

Boolean polynomial operations correspond to set operations.

**Example**

\[(x + xy) + (xy + z) = x + 2xy + z \equiv x + z \iff \]

\[(S_1 \cup S_2) \setminus (S_1 \cap S_2) = \{\{x\}, \{z\}\} \]

(with \(S_1 = \{\{x\}, \{x, y\}\}, S_2 = \{\{x, y\}, \{z\}\}\))

\[x \cdot (y + z) = xy + yz \iff \]

\[\{\{x\} \cup \{y\}, \{x\} \cup \{z\}\} = \{\{x, y\}, \{x, z\}\}\]

Likewise (but more complicated)

Factors/multiples of monomials, degree of a polynomial.

**ZDD Implementation**

Free C/C++ Library Cudd: Fabio Somenzi (University of Colorado)
From ZDDs to POLYBORI

C++-Library

High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings
Implements polynomial operations and basic functionality
### From ZDDs to POLYBOРИ

| High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings |
| Implements polynomial operations and basic functionality |

| Internal handling of polynomial structure |
| (utilizing cache and uniqueness) |
**From ZDDs to POLYBORI**

| C++-Library     | High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings
|                 | Implements polynomial operations and basic functionality |
| ZDDs            | Internal handling of polynomial structure
|                 | (utilizing cache and uniqueness) |
| Ordering-dependend functions | Leading term computation, monomial comparisons, ... added |
### From ZDDs to POLYBORI

<table>
<thead>
<tr>
<th>C++-Library</th>
<th>High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings. Implements polynomial operations and basic functionality.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDDs</td>
<td>Internal handling of polynomial structure (utilizing cache and uniqueness).</td>
</tr>
<tr>
<td>Ordering-dependent functions</td>
<td>Leading term computation, monomial comparisons, … added</td>
</tr>
<tr>
<td>Non-trivial Monomial Orderings (run- &amp; compile-time selectable)</td>
<td>Lexicographic, degree-lexicographic, and degree-reverse-lexicographic (ascending variable order) orderings. Orderings consisting of blocks of degree-orderings.</td>
</tr>
</tbody>
</table>
### From ZDDs to POLYBORI

<table>
<thead>
<tr>
<th>C++-Library</th>
<th>High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings. Implements polynomial operations and basic functionality.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDDs</td>
<td>Internal handling of polynomial structure (utilizing cache and uniqueness).</td>
</tr>
<tr>
<td>Ordering-dependend functions</td>
<td>Leading term computation, monomial comparisons, ... added.</td>
</tr>
<tr>
<td>Non-trivial Monomial Orderings (run- &amp; compile-time selectable)</td>
<td>Lexicographic, degree-lexicographic, and degree-reverse-lexicographic (ascending variable order) orderings. Orderings consisting of blocks of degree-orderings.</td>
</tr>
<tr>
<td>Interfaces</td>
<td>SAGE (python code compatible; community-made) SINGULAR (Prototype)</td>
</tr>
</tbody>
</table>

---

**www.sagemath.org**

**singular.mathematik.uni-kl.de**
Data Structures

Polynomial Ring (tiny excerpt)

```cpp
class BoolePolyRing {
public:
    // ...
    BoolePolyRing(size_type nvars, ordercode_type order = lp);

    dd_type variable(idx_type i) const; // i-th variable
    dd_type zero() const;               // Constant polynomials
    dd_type one() const;

    size_type nVariables() const;

    void changeOrdering(ordercode_type);

protected:
    manager_ptr pMgr;
    // ...
};
```

Generates base components

Ordering (runtime changeable)

Active ring (shared pointer) based on decision diagram manager
**Data Structures**

**Example**
(tiny excerpt)

```cpp
class BoolePolynomial {
public:
    // ...
    BoolePolynomial(); // Default constructor
    BoolePolynomial(bool_type); // Construct from 0 or 1

    self& operator+=(const self&); // Arithmetical operations
    self& operator*=(const self&);

    bool operator==(const self&) const; // Logical operations
    bool operator!=(const self&) const;

    monom_type lead() const; // Get leading term
    size_type deg() const; // Degree of the polynomial
    monom_type usedVariables() const;
    // ... various term access functions ...

private:
    dd_type m_dd; // Actual decision diagram
};
```

**Object-oriented operations**

**High-level abstractions**

Likewise for monomials, variables, and exponent vector
Term-access Functions

Forward iterator over terms

```cpp
const_iterator begin() const;
const_iterator end() const;
in natural – lexicographical – term ordering
```
**Term-access Functions**

Forward iterator over terms

```cpp
const_iterator begin() const;
const_iterator end() const;
```

in natural – lexicographical – term ordering

Runtime-selected ordering

```cpp
ordered_iterator orderedBegin() const;
ordered_iterator orderedEnd() const;
```
**Term-access Functions**

Forward iterator over terms

```cpp
const_iterator begin() const;
const_iterator end() const;
```

in natural – lexicographical – term ordering

Runtime-selected ordering

```cpp
ordered_iterator orderedBegin() const;
ordered_iterator orderedEnd() const;
```

Compile-time selected ordering

```cpp
⟨order⟩_iterator genericBegin⟨⟨order⟩⟩_tag() const;
⟨order⟩_iterator genericEnd⟨⟨order⟩⟩_tag() const;
```

currently for lexicographic, degree-lexicographic, and
degree-reverse-lexicographic (ascending variable order) orderings
### Term-access Functions

<table>
<thead>
<tr>
<th>Term Access Description</th>
<th>C++ Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward iterator over terms</td>
<td><code>const_iterator begin() const;</code></td>
</tr>
<tr>
<td></td>
<td><code>const_iterator end() const;</code></td>
</tr>
<tr>
<td>in natural – lexicographical – term ordering</td>
<td></td>
</tr>
<tr>
<td>Runtime-selected ordering</td>
<td><code>ordered_iterator orderedBegin() const;</code></td>
</tr>
<tr>
<td></td>
<td><code>ordered_iterator orderedEnd() const;</code></td>
</tr>
<tr>
<td>Compile-time selected ordering</td>
<td><code>&lt;order&gt; iterator genericBegin(&lt;order&gt; tag) const;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;order&gt; iterator genericEnd(&lt;order&gt; tag) const;</code></td>
</tr>
<tr>
<td>currently for lexicographic, degree-lexicographic, and degree-reverse-lexicographic (ascending variable order) orderings</td>
<td></td>
</tr>
<tr>
<td>Iterator-like structure for navigation over ZDD nodes</td>
<td><code>navigator navigation() const;</code></td>
</tr>
<tr>
<td></td>
<td><code>incrementThen(), incrementElse() replacing operator++()</code></td>
</tr>
</tbody>
</table>
**POLYBORI’s Python Interface**

Python interface allows for

- Parsing of complex polynomial systems
  (Inner-domain specific language)

- Interactive use (via ipython)
**POLYBORI’s Python Interface**

Python interface allows for

- Parsing of complex polynomial systems
  (Inner-domain specific language)

- Interactive use (via ipython)

Extensive testsuite

- Mainly satisfiability examples; some from cryptography

Rapid Prototyping

- Many algorithms existed in python first
**POLYBORI’s Python Interface**

Python interface allows for

- Parsing of complex polynomial systems (Inner-domain specific language)
- Interactive use (via ipython)

Extensive testsuite

- Mainly satisfiability examples; some from cryptography

Rapid Prototyping

- Many algorithms existed in python first
- Sophisticated and easy extendable strategies for Gröbner base computation
Python Interface (Data Format)

Script level

Easy input and handling of complex polynomial systems

```
declarer_ring([  
    AlternatingBlock(["c","s"],56,start_index=1,reverse=True),  
    AlternatingBlock(["a","b"],8,reverse=True),  
],globals())

ideal = [  
    a(0)*b(1)*a(1)*b(0)+c(1),  
    a(0)*b(1)+a(1)*b(0)+s(1),  
    a(1)*b(1)*a(2)*b(0)+c(3)  
]

ideal = function(ideal);
```

Several variants for Gröber base computation
Comparison of Storage Types for Boolean Polynomials

Tune strategies and heuristics to strengths of underlying data structure!

<table>
<thead>
<tr>
<th>Operation</th>
<th>ZDD</th>
<th>linked list</th>
<th>buckets&lt;sup&gt;1&lt;/sup&gt;</th>
<th>factory&lt;sup&gt;2&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>leading term</td>
<td>++</td>
<td>0</td>
<td>–&lt;sup&gt;3&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td><code>p1 + p2</code></td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td><code>tail(p)</code></td>
<td>–</td>
<td>++</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td><code>redNF(p,m)</code></td>
<td>++</td>
<td>0</td>
<td>–</td>
<td>– –</td>
</tr>
<tr>
<td>canonicalize</td>
<td>++</td>
<td>+</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>memory use</td>
<td>++&lt;sup&gt;5&lt;/sup&gt;</td>
<td>+</td>
<td>–</td>
<td>– –</td>
</tr>
<tr>
<td>iteration</td>
<td>0&lt;sup&gt;6&lt;/sup&gt;</td>
<td>++</td>
<td>–</td>
<td>– (see ZDD)</td>
</tr>
<tr>
<td><code>p1 * p2</code></td>
<td>++&lt;sup&gt;7&lt;/sup&gt;</td>
<td>+</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>factorize</td>
<td>++</td>
<td>–</td>
<td>–</td>
<td>++</td>
</tr>
</tbody>
</table>

1) partly reduced container  
2) recursive expression  
3) lex | special cases | general case  
4) often during reduction with monomial  
5) every substructure stored once  
6) possible, but ordering-dependent  
7) automatically reduced
Caching and Recursion

ZDD normalform

Unique diagram root nodes

\[ a = b \iff \text{rootnode}(a) \text{ is rootnode}(b) \]
Caching and Recursion

<table>
<thead>
<tr>
<th>ZDD normalform</th>
<th>Unique diagram root nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a = b \iff \text{rootnode}(a) \text{ is rootnode}(b) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference counting</th>
<th>Lower memory usage (no deep copies)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Caching of operations</th>
<th>( a \triangleleft b ) never evaluated twice (also for sub-diagrams)</th>
</tr>
</thead>
</table>
Caching and Recursion

ZDD normalform

Unique diagram root nodes

\[ a = b \iff \text{rootnode}(a) \text{ is rootnode}(b) \]

Reference counting

Lower memory usage (no deep copies)

Caching of operations

\[ a \diamond b \text{ never evaluated twice (also for sub-diagrams)} \]

Advantage

Speed-up recursive procedures
Caching and Recursion

Example: \( \text{deg}(f) \)

\[
f = q + x \cdot p
\]

**Input:** \( f \) Boolean polynomial

**Output:** \( d = \text{deg}(f) \)

- if \( f \in \{0, 1\} \) then
  - set \( d := 0 \)
- else if isCached(\( \text{deg}, f \)) then
  - set \( d := \text{cache}(\text{deg}, f) \)
- else
  - set \( d := \max(\text{deg(thenBranch}(f)) + 1, \text{deg(elseBranch}(f))) \)

**end if**
Caching and Recursion

Example: \( \deg(f) \)

\[
\text{Input: } f \text{ Boolean polynomial} \quad \text{Output: } d = \deg(f)
\]

- if \( f \in \{0, 1\} \) then
  - set \( d := 0 \)
- else if isCached(deg, f) then
  - set \( d := \text{cache}(\deg, f) \)
- else
  - set \( d := \max(\deg(\text{thenBranch}(f)) + 1, \deg(\text{elseBranch}(f))) \)
  - insert \( \text{cache}(\deg, f) := d \)
end if
### Caching and Recursion

**Example:** \( \text{deg}(f) \)

**Input:** \( f \) Boolean polynomial

**Output:** \( d = \text{deg}(f) \)

```
if \( f \in \{0, 1\} \) then
    set \( d := 0 \)
else if \( \text{isCached}(\text{deg}, f) \) then
    set \( d := \text{cache}(\text{deg}, f) \)
else
    set \( d := \max(\text{deg(thenBranch}(f)) + 1, \text{deg(elseBranch}(f))) \)
insert \( \text{cache}(\text{deg}, f) := d \)
end if
```
Caching and Recursion

Example: \( \text{deg}(f) \)

**Input:** \( f \) Boolean polynomial

**Output:** \( d = \text{deg}(f) \)

```
if \( f \in \{0, 1\} \) then
  set \( d := 0 \)
else if isCached(deg, \( f \)) then
  set \( d := \text{cache}(\text{deg}, \( f \)) \)
else
  set \( d := \max(\text{deg(thenBranch}(\( f \)) + 1, \text{deg(elseBranch}(\( f \))) \)
insert \( \text{cache}(\text{deg}, \( f \)) := d \)
end if
```
Caching and Recursion

Input: $f$ Boolean polynomial

Output: $d = \deg(f)$

\[
\begin{align*}
\text{if} & \quad f \in \{0, 1\} \quad \text{then} \\
& \quad \text{set } d := 0 \\
\text{else if} & \quad \text{isCached(deg, } f) \quad \text{then} \\
& \quad \text{set } d := \text{cache(deg, } f) \\
\text{else} & \\
& \quad \text{set } d := \max(\deg(\text{thenBranch}(f)) + 1, \\
& \quad \quad \quad \deg(\text{elseBranch}(f))) \\
& \quad \text{insert } \text{cache(deg, } f) := d \\
\text{end if}
\end{align*}
\]

- Reuse for polynomials with likewise structure.
- Fast access to intermediate results by other functions.
Caching and Recursion

Example: $\text{lead}(f)$

First (lexicographical) term $t$ in $f$ with $\deg t = \deg f$.

**Input:** $f$ Boolean polynomial  
**Output:** $t = \text{lead}(f)$ (deg-lex)

\[
\text{if } f \in \{0, 1\} \text{ then} \\
\text{set } t := 1 \\
\text{else if isCached(lead, } f) \text{ then} \\
\text{set } t := \text{cache(lead, } f) \\
\text{else} \\
\text{set } x := \text{root variable of } f \\
\text{if } \deg(f) = \deg(\text{thenBranch}(f)) + 1 \text{ then} \\
\text{set } t := x \cdot \text{lead(thenBranch}(f)) \\
\text{else} \\
\text{set } t := \text{lead(elseBranch}(f)) \\
\text{end if} \\
\text{insert cache(lead, } f) := t \\
\text{end if}
\]
Caching and Recursion

Example: lead(f)

First (lexicographical) term \( t \) in \( f \) with \( \deg t = \deg f \).

**Input:** \( f \) Boolean polynomial  
**Output:** \( t = \text{lead}(f) \) (deg-lex)

\[
\begin{align*}
\text{if} & \; f \in \{0, 1\} \; \text{then} \\
& \quad \text{set} \; t := 1 \\
\text{else if} & \; \text{isCached(lead, } f) \; \text{then} \\
& \quad \text{set} \; t := \text{cache(lead, } f) \\
\text{else} & \\
& \quad \text{set} \; x := \text{root variable of } f \\
& \quad \text{if} \; \deg(f) = \deg(\text{thenBranch}(f)) + 1 \; \text{then} \\
& \quad \quad \text{set} \; t := x \cdot \text{lead(thenBranch}(f)) \\
& \quad \text{else} \\
& \quad \quad \text{set} \; t := \text{lead(elseBranch}(f)) \\
\text{end if} \\
\text{insert} & \; \text{cache(lead, } f) := t \\
\text{end if}
\]
Caching and Recursion

Example: `lead(f)`

First (lexicographical) term $t$ in $f$ with $\deg t = \deg f$.

**Input:** $f$ Boolean polynomial  
**Output:** $t = \text{lead}(f)$ (deg-lex)

```
if $f \in \{0, 1\}$ then
    set $t := 1$
else if isCached(lead, $f$) then
    set $t := \text{cache}(\text{lead}, f)$
else
    set $x$ := root variable of $f$
    if $\deg(f) = \deg(\text{thenBranch}(f)) + 1$ then
        set $t := x \cdot \text{lead}(\text{thenBranch}(f))$
    else
        set $t := \text{lead}(\text{elseBranch}(f))$
    end if
    insert $\text{cache}(\text{lead}, f) := t$
end if
```
Caching and Recursion

Example: lead(f)

First (lexicographical) term \( t \) in \( f \) with \( \deg t = \deg f \).

**Input:** \( f \) Boolean polynomial  
**Output:** \( t = \text{lead}(f) \) (deg-lex)

\[
\text{if } f \in \{0, 1\} \text{ then} \\
\quad \text{set } t := 1 \\
\text{else if } \text{isCached(lead, } f) \text{ then} \\
\quad \text{set } t := \text{cache(lead, } f) \\
\text{else} \\
\quad \text{set } x := \text{root variable of } f \\
\quad \text{if } \deg(f) = \deg(\text{thenBranch}(f)) + 1 \text{ then} \\
\quad \quad \text{set } t := x \cdot \text{lead(thenBranch}(f)) \\
\quad \text{else} \\
\quad \quad \text{set } t := \text{lead(elseBranch}(f)) \\
\text{end if} \\
\text{insert } \text{cache(lead, } f) := t \\
\text{end if}
\]
**Formal Verification**

**Boolean Polynomials and Logical Expressions**

- Propositional logic $\rightarrow$ Boolean polynomial
- Each formula from propositional logic can be stated using Or, Not, True, False
- Choose a mapping: True $\rightarrow$ 0, False $\rightarrow$ 1
  $\text{Or}(x, y) \rightarrow x \cdot y$, $\text{Not}(x) \rightarrow 1 + x$
Formal Verification

Boolean Polynomials and Logical Expressions

- Propositional logic $\rightarrow$ Boolean polynomial
- Each formula from propositional logic can be stated using Or, Not, True, False
- Choose a mapping: True $\rightarrow$ 0, False $\rightarrow$ 1
  $\text{Or}(x, y) \rightarrow x \cdot y$, $\text{Not}(x) \rightarrow 1 + x$

Property Checking

- Digital system given as set $R$ of Boolean polynomials
- Property $p$ (Boolean expression) to be verified w. r. t. $R$
- Check via normal form computation: $\text{NF}(p, R) = 0$?
Gröbner Basis

- Let $G = \{g_1, \ldots, g_m\}$ be (multivariate) polynomials
- Define an ordering $>$ on the monomials of $K[x_1, \ldots, x_m]$
- For a polynomial $p$ define $\text{lead}(p)$ the biggest monomial occurring in $p$
Gröbner Basis

- Let $G = \{g_1, \ldots, g_m\}$ be (multivariate) polynomials

- Define an ordering $>$ on the monomials of $K[x_1, \ldots, x_m]$

- For a polynomial $p$ define $\text{lead}(p)$ the biggest monomial occurring in $p$

$G$ is a Gröbner basis $\iff$ for every $0 \neq p$ in the ideal generated by $G$ there exists $g \in G$: $\text{lead}(g)$ divides $\text{lead}(p)$
Gröbner Basis

- Let \( G = \{g_1, \ldots, g_m\} \) be (multivariate) polynomials
- Define an ordering \( > \) on the monomials of \( K[x_1, \ldots, x_m] \)
- For a polynomial \( p \) define \( \text{lead}(p) \) the biggest monomial occurring in \( p \)

\[ G \text{ is a Gröbner basis} \iff \text{ for every } 0 \neq p \text{ in the ideal generated by } G \text{ there exists } g \in G: \text{lead}(g) \text{ divides } \text{lead}(p) \]

Examples
- \( \langle G \rangle \) is the whole ring, then \( G \ni 1 \)
- \( \{x_1 - c_1, \ldots, x_n - c_n\} \) for a (radical) ideal with unique solution \( c_1, \ldots, c_n \)
Linear Lead-Rewriting System

System of Boolean polynomials

- \( G = \{g_1, \ldots, g_m\} \) in variables \( x_1, \ldots, x_n \)

Suppose, the following holds:

- For each \( i \): lead\( (g_i) \) is a variable \( x_{g_i} \) (\( \rightarrow x_{g_i} \) not in \( g_i - \text{lead}(g_i) \))
- For \( i \neq j \): lead\( (g_i) \neq \text{lead}(g_j) \)

\( \Rightarrow \) Then \( G \) is a Gröbner basis and
\( R := G \cup \{x_i^2 + x_i | i = 1, \ldots, n\} \) is a Gröbner basis
Linear Lead-Rewriting System

System of Boolean polynomials

- $G = \{g_1, \ldots, g_m\}$ in variables $x_1, \ldots, x_n$

Suppose, the following holds:

- For each $i$: lead($g_i$) is a variable $x_{g_i}$ ($\rightarrow x_{g_i}$ not in $g_i - \text{lead}(g_i)$)
- For $i \neq j$: lead($g_i$) $\neq$ lead($g_j$)

$\Rightarrow$ Then $G$ is a Gröbner basis and

$R := G \cup \{x_i^2 + x_i | i = 1, \ldots, n\}$ is a Gröbner basis

For each Boolean polynomial $p$ there exists exactly one Boolean polynomial $q$ in $\{x_1, \ldots, x_n\}\{\text{lead}(g_1), \ldots, \text{lead}(g_m)\}$: $p - q \in \langle R \rangle$

There exists an algorithm to calculate $q =: \text{NF}(p, R)$
Linear Lead-Rewriting System

System of Boolean polynomials

\[ G = \{g_1, \ldots, g_m\} \text{ in variables } x_1, \ldots, x_n \]

Suppose, the following holds:

- For each \( i \): lead\((g_i)\) is a variable \( x_{g_i} \) (\( \rightarrow x_{g_i} \) not in \( g_i \) – lead\((g_i)\))
- For \( i \neq j \): lead\((g_i)\) \( \neq \) lead\((g_j)\)

\[ \Rightarrow \text{ Then } G \text{ is a Gröbner basis and } R := G \cup \{x_{g_i}^2 + x_i | i = 1, \ldots, n\} \text{ is a Gröbner basis} \]

For each Boolean polynomial \( p \) there exists exactly one Boolean polynomial \( q \) in \( \{x_1, \ldots, x_n\} \setminus \{\text{lead}(g_1), \ldots, \text{lead}(g_m)\} \): \( p - q \in \langle R \rangle \)

There exists an algorithm to calculate \( q =: \text{NF}(p, R) \)

Example:

\[ G = \{x - 1, y - 1\}, \quad p = x \cdot y \cdot z \rightarrow \text{NF}(p, R) = z \]
Special Situation in Formal Verification

- A digital circuit is described by a linear lead rewriting system

- Each output signal $o_j$ is described by a Boolean function $f(i_1, \ldots i_s)$ with respect to its input signals $i_1, \ldots, i_s$
Special Situation in Formal Verification

- A digital circuit is described by a linear lead rewriting system
- Each output signal $o_j$ is described by a Boolean function $f(i_1, \ldots, i_s)$ with respect to its input signals $i_1, \ldots, i_s$
- Take $e_j := o_j - f(i_1, \ldots, i_s)$ as equation
- $R := \{ e_j | o_j \text{ output signal} \} \cup \{ x_i^2 + x_i | i = 1, \ldots, n \}$
Special Situation in Formal Verification

- A digital circuit is described by a linear lead rewriting system

- Each output signal $o_j$ is described by a Boolean function $f(i_1, \ldots, i_s)$ with respect to its input signals $i_1, \ldots, i_s$

- Take $e_j := o_j - f(i_1, \ldots, i_s)$ as equation

- $R := \{e_j| o_j \text{ output signal}\} \cup \{x_i^2 + x_i | i = 1, \ldots, n\}$

- For the system described by this equations, we would like to check a property $p$, which is given as Boolean expression

- Check via normal form computation: $\text{NF}(p, R) = 0$?
Representation of Adder Blocks with Carry Bit

Variables: $s(0), \ldots, s(n - 1)$ (sums), $c(0), \ldots, c(n - 1)$ (carry bits), $a(0), \ldots, a(n - 1), b(0), \ldots, b(n - 1)$ (inputs)

Topological Variable Order: $s(n - 1), c(n - 1), a(n - 1), b(n - 1)$, $s(n - 2), c(n - 2), a(n - 2), b(n - 2), \ldots$

Outputs before inputs, $a, b$ reverse alternating (fast ZDD handling)
Representation of Adder Blocks with Carry Bit

Variables

\[ s(0), \ldots, s(n - 1) \text{ (sums), } c(0), \ldots, c(n - 1) \text{ (carry bits), } a(0), \ldots, a(n - 1), b(0), \ldots, b(n - 1) \text{ (inputs)} \]

Topological Variable Order

\[ s(n - 1), c(n - 1), a(n - 1), b(n - 1), \]
\[ s(n - 2), c(n - 2), a(n - 2), b(n - 2), \ldots \]

Outputs before inputs, \( a, b \) reverse alternating (fast ZDD handling)

Equations

\[ c(i) + \text{carry}_i \]
\[ s(i) + a(i) + b(i) + \text{carry}_i \]

with \( \text{carry}_{i-1} = 0 \)

and \( \text{carry}_i = a(i) \cdot (b(i) + \text{carry}_{i-1}) + b(i) \cdot \text{carry}_{i-1} \)
Carry-Bit Graph

\( a_i \)

\( b_i \)

\( b_i \)

0

carry \( i-1 \)

1
POLYBORI - A Gröbner Basis Framework for Boolean Polynomials

**Carry-Bit Graph**

![Carry-Bit Graph Diagram]

- $a_i$
- $b_i$
- $a_{i-1}$
- $b_{i-1}$
- Carry $i-2$

Optimization on many levels

Model

Algorithm

Data Structure

translation into Boolean polynomials

one or several public/private key pairs for crypto

topology of digital circuits

premade, optimized components

variable orderings

suitable block orderings

Polynomial

ZDD

Matrices

Buchberger specialized scripts

F4

Python

C++
**Polybori - A Gröbner Basis Framework for Boolean Polynomials**

**Carry-Bit Graph**

<table>
<thead>
<tr>
<th>i-th bit</th>
<th>terms</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>2047</td>
<td>32</td>
</tr>
</tbody>
</table>

- Sophisticated ordering of variables
- Linear growth of ZDDs (plaited structure)
- Fast generation and arithmetic operations

\[ n \quad 2^n - 1 \quad 3n - 1 \]
**Polybori** - A Gröbner Basis Framework for Boolean Polynomials

### Carry-Bit Graph

<table>
<thead>
<tr>
<th>i-th bit</th>
<th>terms</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>2047</td>
<td>32</td>
</tr>
<tr>
<td>n</td>
<td>$2^n - 1$</td>
<td>$3n - 1$</td>
</tr>
</tbody>
</table>

- Sophisticated ordering of variables
- Linear growth of ZDDs (plaited structure)
- Fast generation and arithmetic operations

**Demo:** $n = 4096$
**Application: Abstract 4096-Bit Adder and Logic**

```python
adder_bits=4096
adder_block=AdderBlock(sums="s", carries="c",
    input1="a", input2="b",
    adder_bits=adder_bits, start_index=1)

declare_ring([Block("x",100), adder_block, Block("y",100)])
ideal=[
    y(0)+y(1)+1,
    a(adder_bits)+1, b(adder_bits)+y(0)+y(1),
    x(0)+(c(adder_bits))*x(1)
]
adder_block.implement(ideal)

claims=[
    c(adder_bits)+1,
    if_then([x(1)], [x(0)]),
    if_then([a(1), b(1)], [c(1), s(1)]),
    if_then([a(1)+1], [c(1)+b(1), s(1)+b(1)+1]),
    s(adder_bits)+c(adder_bits-1),
    x(33)
]
```

---

**Mathematisches Forschungsinstitut Oberwolfach**

**Fraunhofer ITWM**

**Institut Techno- und Wirtschaftsmathematik**
Counter Example for Failed Claims

- A claim $c$ fails $\iff q := NF(c, R) \neq 0$
- A Boolean polynomial is 0 $\iff$ it is 0 as a function

Algorithm

Boolean polynomial $q$ reduced w. r. t. $R$, $R$ red. Gröbner basis

Variables($q$) $\subseteq V := \{x_1, \ldots, x_n\} \setminus \{\text{Leading Terms}(R)\}$
Counter Example for Failed Claims

- A claim $c$ fails $\iff q := \text{NF}(c, R) \neq 0$
- A Boolean polynomial is $0$ $\iff$ it is $0$ as a function

Algorithm

Boolean polynomial $q$ reduced w. r. t. $R$, $R$ red. Gröbner basis

$$\text{Variables}(q) \subseteq V := \{x_1, \ldots, x_n\} \setminus \{\text{Leading Terms}(R)\}$$

Step 1  Find $v_1, \ldots, v_n \in \{0, 1\}$, where $q(v_1, ..., v_n) = 1$

Now we have to make it compatible with relations in $R$:
Counter Example for Failed Claims

- A claim $c$ fails $\iff q := \text{NF}(c, R) \neq 0$
- A Boolean polynomial is 0 $\iff$ it is 0 as a function

Algorithm

Boolean polynomial $q$ reduced w. r. t. $R$, $R$ red. Gröbner basis

$\text{Variables}(q) \subseteq V := \{x_1, \ldots, x_n\} \setminus \{\text{Leading Terms}(R)\}$

**Step 1** Find $v_1, \ldots, v_n \in \{0, 1\}$, where $q(v_1, \ldots, v_n) = 1$

Now we have to make it compatible with relations in $R$:

**Step 2** For every $x_i \in \text{lead}(R)$, we have $p \in R$, with $x_i = \text{lead}(x)$ and $\text{tail}(p) \in K[V]$

Redefine $v_i := \text{tail}(p)(v_1, \ldots, v_n)$
**Counter Example for Failed Claims**

- A claim $c$ fails $\iff q := \text{NF}(c, R) \neq 0$
- A Boolean polynomial is 0 $\iff$ it is 0 as a function

**Algorithm**

Boolean polynomial $q$ reduced w. r. t. $R$, $R$ red. Gröbner basis

$$\text{Variables}(q) \subseteq V := \{x_1, \ldots, x_n\} \setminus \{\text{Leading Terms}(R)\}$$

**Step 1** Find $v_1, \ldots, v_n \in \{0, 1\}$, where $q(v_1, \ldots, v_n) = 1$

Now we have to make it compatible with relations in $R$:

**Step 2** For every $x_i \in \text{lead}(R)$, we have $p \in R$, with $x_i = \text{lead}(x)$ and $\text{tail}(p) \in K[V]$

Redefine $v_i := \text{tail}(p)(v_1, \ldots, v_n)$

**Theorem**

$v_1, \ldots, v_n$ provides a valid counter example
Algorithm: Find a One of a Boolean Polynomial

Input: Boolean polynomial \( p \neq 0 \)

Out: \( \{x_i \rightarrow v_i\}_i : p(v_1, \ldots, v_k) = 1 \)

if \( p = 1 \) then
    return \( \emptyset \) // empty tuple
endif

Set \( r := \text{rootnode}(p) \)
Set \( x := \text{Variable corresponding to } r \)
if elseBranch(\( r \)) is 0 then
    Set \( v = 1 \)
    Set \( q = \text{thenBranch}(r) \) // plug in 1
else
    Set \( v = 0 \)
    Set \( q = \text{elseBranch}(r) \) // plug in 0
endif
return \( \{x \rightarrow v\} \cup \text{find\_one}(q) \)
### Benchmarks: Pigeon Hole Problems

<table>
<thead>
<tr>
<th>Hole</th>
<th>Vars./Eqs.</th>
<th>PolyBoRi</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hole6</td>
<td>42/133</td>
<td>0.13 s</td>
<td>0.01 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>51.64 MB</td>
<td>1.96 MB</td>
</tr>
<tr>
<td>hole7</td>
<td>56/204</td>
<td>0.46 s</td>
<td>0.06 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>51.89 MB</td>
<td>1.96 MB</td>
</tr>
<tr>
<td>hole8</td>
<td>72/297</td>
<td>1.88 s</td>
<td>0.30 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>56.59 MB</td>
<td>2.08 MB</td>
</tr>
<tr>
<td>hole9</td>
<td>90/415</td>
<td>8.01 s</td>
<td>2.31 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>84.04 MB</td>
<td>2.35 MB</td>
</tr>
<tr>
<td>hole10</td>
<td>110/561</td>
<td>44.40 s</td>
<td>25.20 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>97.68 MB</td>
<td>3.24 MB</td>
</tr>
<tr>
<td>hole11</td>
<td>132/738</td>
<td>643.14 s</td>
<td>782.65 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>130.83 MB</td>
<td>7.19 MB</td>
</tr>
<tr>
<td>hole12</td>
<td>156/949</td>
<td>10264.92 s</td>
<td>22920.20 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>338.66 MB</td>
<td>17.13 MB</td>
</tr>
</tbody>
</table>

**Remark**

Good performance mainly due to fast Boolean multiplication
Benchmarks: Lex. Normalform Computations without Gröbner basis

<table>
<thead>
<tr>
<th># points</th>
<th>smallest lex.</th>
<th>interpolate simple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time/1 s</td>
<td>length</td>
</tr>
<tr>
<td>100</td>
<td>0.01</td>
<td>42</td>
</tr>
<tr>
<td>500</td>
<td>0.06</td>
<td>249</td>
</tr>
<tr>
<td>1000</td>
<td>0.29</td>
<td>508</td>
</tr>
<tr>
<td>5000</td>
<td>5.53</td>
<td>2552</td>
</tr>
<tr>
<td>10000</td>
<td>19.78</td>
<td>5020</td>
</tr>
<tr>
<td>50000</td>
<td>250.95</td>
<td>25012</td>
</tr>
<tr>
<td>100000</td>
<td>897.85</td>
<td>50093</td>
</tr>
<tr>
<td>200000</td>
<td>3488.61</td>
<td>99868</td>
</tr>
<tr>
<td>500000</td>
<td>20336.02</td>
<td>249675</td>
</tr>
</tbody>
</table>

Interpolation vs. Gröbner approaches

Remark
Normalform w. r. t. Boolean functions, given by points in \( \mathbb{Z}_2^n \)
Gröbner basis sizes calculated without explicit basis computation
Summary

- Boolean polynomials as subsets of the power set of the ring variables, which can be represented efficiently as ZDDs
Summary

- Boolean polynomials as subsets of the power set of the ring variables, which can be represented efficiently as ZDDs
- High-level C++ library: internal ZDD structure in polynomials encapsulated, as well as transparent access to ZDD structure
Summary

- Boolean polynomials as subsets of the power set of the ring variables, which can be represented efficiently as ZDDs.
- High-level C++ library: internal ZDD structure in polynomials encapsulated, as well as transparent access to ZDD structure.
- Easy usable Python interface for rapid prototyping of sophisticated heuristics for Gröbner base computations.
Summary

- Boolean polynomials as subsets of the power set of the ring variables, which can be represented efficiently as ZDDs
- High-level C++ library: internal ZDD structure in polynomials encapsulated, as well as transparent access to ZDD structure
- Easy usable Python interface for rapid prototyping of sophisticated heuristics for Gröbner base computations
- Gröbner property checker for formal verification of given properties. Generation of counter examples, if claims fail
Summary

- Boolean polynomials as subsets of the power set of the ring variables, which can be represented efficiently as ZDDs
- High-level C++ library: internal ZDD structure in polynomials encapsulated, as well as transparent access to ZDD structure
- Easy usable Python interface for rapid prototyping of sophisticated heuristics for Gröbner base computations
- Gröbner property checker for formal verification of given properties. Generation of counter examples, if claims fail
- Optimized setup of arithmetic components (adder)