Combinational Circuit Verification using Strong Nullstellensatz
Overcoming the Complexity of Gröbner Bases for Efficient Verification over $\mathbb{F}_{2^k}$

Priyank Kalla

Associate Professor
Electrical and Computer Engineering, University of Utah
kalla@ece.utah.edu
http://www.ece.utah.edu/~kalla

Nov 17-19, 2014
What we have learnt so far...

**Theorem (Weak Nullstellensatz)**

Let $\overline{F}$ be an algebraically closed field. Given ideal $J \subset \overline{F}[x_1, \ldots, x_n]$, $V_{\overline{F}}(J) = \emptyset \iff J = \overline{F}[x_1, \ldots, x_n] \iff 1 \in J \iff \text{reducedGB}(J) = \{1\}$.

**Theorem (Regular Nullstellensatz)**

Let $\overline{F}$ be an algebraically closed field. Let $J = \langle f_1, \ldots, f_s \rangle \subset \overline{F}[x_1, \ldots, x_n]$. Let another polynomial $f$ vanish on $V_{\overline{F}}(J)$, so $f \in I(V_{\overline{F}}(J))$. Then, $\exists m \in \mathbb{Z}_{\geq 1}$ s.t.

$$f^m \in J,$$

and conversely.

**Theorem (The Strong Nullstellensatz)**

Over an algebraically closed field $I(V(J)) = \sqrt{J}$
Nullstellensatz over $\mathbb{F}_q$

**Theorem (Weak Nullstellensatz over $\mathbb{F}_{2^k}$)**

Let ideal $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_n]$ be an ideal. Let $J_0 = \langle x_1^{2^k} - x_1, \ldots, x_n^{2^k} - x_n \rangle$ be the ideal of all vanishing polynomials. Then:

$$\text{V}_{\mathbb{F}_{2^k}}(J) = \emptyset \iff \text{V}_{\mathbb{F}_{2^k}}(J + J_0) = \emptyset \iff \text{reducedGB}(J + J_0) = \{1\}$$

**Theorem ($J + J_0$ is radical)**

Over Galois fields $\sqrt{J + J_0} = J + J_0$, i.e. $J + J_0$ is a radical ideal.

**Theorem (Strong Nullstellensatz over $\mathbb{F}_q$)**

$$I(V_{\mathbb{F}_q}(J)) = I(V_{\mathbb{F}_q}(J + J_0)) = \sqrt{J + J_0} = J + J_0$$
Radical Membership.

- Given $J$, we cannot easily find generators of $\sqrt{J}$
- But we can test for membership in $\sqrt{J}$
  - $f \in \sqrt{J} \iff \text{reducedGB}(J + \langle 1 - y \cdot f \rangle) = \{1\}$
Given specification polynomial: \( f : Z = A \cdot B \pmod{P(x)} \) over \( \mathbb{F}_{2^k} \), for given \( k \), and given \( P(x) \), s.t. \( P(\alpha) = 0 \)

Given circuit implementation \( C \)
- Primary inputs: \( A = \{a_0, \ldots, a_{k-1}\} \), \( B = \{b_0, \ldots, b_{k-1}\} \)
- Primary Output \( Z = \{z_0, \ldots, z_{k-1}\} \)
- \( A = a_0 + a_1 \alpha + a_2 \alpha^2 + \cdots + a_{k-1} \alpha^{k-1} \)
- \( B = b_0 + b_1 \alpha + \cdots + b_{k-1} \alpha^{k-1} \), \( Z = z_0 + z_1 \alpha + \cdots + z_{k-1} \alpha^{k-1} \)

Does the circuit \( C \) implement \( f \)?

Mathematically:
- Model the circuit (gates) as polynomials: \( f_1, \ldots, f_s \)
  \[ J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_n] \]
- Does \( f \) agree with solutions to \( f_1 = f_2 = \cdots = f_s = 0 \)?
- Does \( f \) vanish on the Variety \( V_{\mathbb{F}_q}(J) \)?
- Is \( f \in I(V_{\mathbb{F}_q}(J)) = J + J_0 \) or is \( f \xrightarrow{\text{GB}(J+J_0)} + 0 \)?
Example Formulation

Gates as polynomials

$F_2 \subset F_{2^k}$:

Ideal $J$:

\[ z_0 = s_0 + s_3; \quad \mapsto \quad f_1 : z_0 + s_0 + s_3 \]
\[ s_0 = a_0 \cdot b_0; \quad \mapsto \quad f_2 : s_0 + a_0 \cdot b_0 \]

\[ \vdots \]

\[ A + a_0 + a_1 \alpha; \quad B + b_0 + b_1 \alpha; \quad Z + z_0 + z_1 \alpha \]

Ideal $J_0$:

\[ z_0^2 - z_0, s_0^2 - s_0, \]
\[ \vdots \]
\[ A^{2^k} - A, B^{2^k} - B, \]
\[ Z^{2^k} - Z \]
Complexity of Gröbner Basis

- Complexity of Gröbner basis
  - Degree of polynomials in $G$ is bounded by $2(\frac{1}{2}d^2 + d)^{2^{n-1}}$ [1]
  - Doubly-exponential in $n$ and polynomial in the degree $d$
- This is the complexity of the GB problem, not of Buchberger’s algorithm – that’s still a mystery
- For $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, Complexity $GB(J + J_0) : q^{O(n)}$ (Single exponential)
- Improving Buchberger’s algorithm:
  - Improve term ordering (heuristics)
  - Get to all $S(f, g) \xrightarrow{G} 0$ quickly; i.e. arrive at a GB quickly (hard to predict)
  - Improve the implementation of polynomial division; ideas proposed by Faugére in the $F_4$ algorithm
Complexity of Gröbner Basis and Term Orderings

- For $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, Complexity $GB(J + J_0) : q^{O(n)}$
- GB complexity very sensitive to **term ordering**
- A term order has to be imposed for systematic polynomial computation

Let $f = 2x^2yz + 3xy^3 - 2x^3$

- **LEX** $x > y > z$: $f = -2x^3 + 2x^2yz + 3xy^3$
- **DEGLEX** $x > y > z$: $f = 2x^2yz + 3xy^3 - 2x^3$
- **DEGREVLEX** $x > y > z$: $f = 3xy^3 + 2x^2yz - 2x^3$

Recall, $S$-polynomial depends on term ordering:

$$S(f, g) = \frac{L}{\text{lt}(f)} \cdot f - \frac{L}{\text{lt}(g)} \cdot g; \quad L = \text{LCM}(\text{lm}(f), \text{lm}(g))$$
Effect of Term Orderings on Buchberger’s Algorithm

The Product Criteria

If $\text{lm}(f) \cdot \text{lm}(g) = \text{LCM}(\text{lm}(f), \text{lm}(g))$, then $S(f, g) \xrightarrow{G'} + 0$.

**LEX:** $x_0 > x_1 > x_2 > x_3$

- $f = x_0 x_1 + x_2, \ g = x_1 x_2 + x_3$
- $\text{lm}(f) = x_0 x_1; \ \text{lm}(g) = x_1 x_2$
- $S(f, g) \xrightarrow{G'} + x_0 x_3 + x_2^2$

**LEX:** $x_3 > x_2 > x_1 > x_0$

- $f = x_2 + x_0 x_1, \ g = x_3 + x_1 x_2$
- $\text{lm}(f) = x_2; \ \text{lm}(g) = x_3, \ S(f, g) \xrightarrow{G'} + 0$

“Obviated” Buchberger’s algorithm... really?

Find a “term order” that makes ALL $\{\text{lm}(f), \ \text{lm}(g)\}$ relatively prime.
Recall Buchberger’s theorem

The set $G = \{g_1, \ldots, g_t\}$ is a Gröbner basis iff for all pairs $(f, g) \in G$, $S(f, g) \xrightarrow{G} 0$

- If we can make leading monomials of all pairs $lm(f), lm(g)$ relatively prime, then all $Spoly(f, g)$ reduce to 0
- This would imply that the polynomials already constitute a Gröbner basis
- No need to compute a GB, may be able to circumvent the GB complexity issues
- Can a term order be derived that makes leading monomials of all polynomials relatively prime?
  - For an “acyclic” circuit, make the gate output variable $x_i$ greater than all variables $x_j$ that are inputs to the gate
For Circuits, such an order can be derived

Perform a Reverse Topological Traversal of the circuit, order the variables according to their reverse topological levels

LEX with $Z > \{A > B\} > \{z_0 > z_1\} > \{r_0 > s_0 > s_3\} > \{s_1 > s_2\} > \{a_0 > a_1 > b_0 > b_1\}$

This makes every gate output a leading term, and $\{f_1, \ldots, f_{10}\}$ is a Gröbner basis.
Using the Topological Term Order:

- \( F = \{f_1, \ldots, f_s\} \) is a Gröbner Basis of \( J = \langle f_1, \ldots, f_s \rangle \)

- \( F_0 = \{x_1^q - x_1, \ldots, x_n^q - x_n\} \) is also a Gröbner basis of \( J_0 \) (these polynomials also have relatively prime leading terms)

- But we have to compute a Gröbner Basis of \( J + J_0 = \langle f_1, f_2 \ldots, f_s, x_1^q - x_1, \ldots, x_n^q - x_n \rangle \)

- It turns out that \( \{f_1, f_2 \ldots, f_s, x_1^q - x_1, \ldots, x_n^q - x_n\} \) is a Gröbner basis!!

- From our circuit: \( f_i = x_i + \text{tail}(f_i) = x_i + P \)

- Vanishing polynomials \( x_i^q - x_i \) with same variable \( x_i \)

- Only pairs to consider: \( S(f_i, x_i^q - x_i) \) in Buchberger’s Algorithm

- All other pairs will have relatively prime leading terms, which will reduce to 0 modulo \( G \)
This term order renders a Gröbner basis by construction.

So, let us compute $S(f_i = x_i + P, \ x_i^q - x_i)$:

$$S(f_i = x_i + P, \ x_i^q - x_i) = x_i^{q-1}P + x_i$$

$$x_i^{q-1}P + x_i \xrightarrow{x_i + P} x_i^{q-2}P^2 + x_i \xrightarrow{x_i + P} \ldots \xrightarrow{x_i + P} P^q - P \xrightarrow{J_0} + 0$$

Since $P^q - P$ is a vanishing polynomial, $P^q - P \in J_0$ and $P^q - P \xrightarrow{J_0} + 0$

Conclusion: The set of polynomials $F \cup F_0 = \{f_1, \ldots, f_s, \ x_i^q - x_i, \ldots, x_n^q - x_n\}$ is itself a Gröbner basis due to the reverse topological term order derived from the circuit!
Our Minimal Gröbner Basis

Conclusion:

- Our term order makes \( G = \{ f_1, \ldots, f_s, x_1^q - x_1, \ldots, x_n^q - x_n \} \) a Gröbner Basis
- This \( \text{GB}(J + J_0) \) can be further simplified (made minimal)
  - Two types of polynomials: \( f_i = x_i + P, \quad g_i = x_i^q - x_i \)
  - Primary inputs bits are never a leading term of any polynomial
  - Primary inputs are not the output of any gate
- For \( x_i \notin \) primary inputs, \( f_i = x_i + P \) divides \( x_i^q - x_i \); remove \( x_i^q - x_i \)
- Keep \( J_0 = \langle x_i^2 - x_i : x_i \in \text{primary input bits} \rangle \)

Our term order makes \( G = \{ f_1, \ldots, f_s, \ x_P^2 - x_P \} \) a minimal Gröbner basis by construction!

Verify the circuit only by a reduction: \( f \xrightarrow{G} + 0? \)
Our Overall Approach

- Given the circuit, perform reverse topological traversal
- Derive the term order to represent the polynomials for every gate
- The set: \( \{ F, F_0 \} = \{ f_1, \ldots, f_s, \ x_i^2 - x_i : x_i \in X_{PI} \} \) is a minimal Gröbner Basis
- Obtain: \( f \xrightarrow{F;F_0} r \)
- If \( r = 0 \), the circuit is verified correct
- If \( r \neq 0 \), then \( r \) contains only the primary input variables
- Any SAT assignment to \( r \neq 0 \) generates a counter-example
- Counter-example found in no time as \( r \) is simplified by Gröbner basis reduction
Move the complexity to that of Polynomial Division

Is this Magic? Or have I told you the full story?

- Reduce $x^n$ modulo $\langle x + P \rangle$, how many cancellations?
  - Requires raising $P$ to the $n^{th}$ power
  - $P$ is the $\text{tail}(f_i)$
  - Depending upon $n$, this can become complicated

- **Reduce** this **minimal** GB $G = \{ F, F_0 \}$, what does it look like?
  - $f_i = x_i + \text{tail}(f_i)$, where $\text{tail}(f_i) = P(x_j)$, $x_i > x_j$
  - There exists $f_j = x_j + \text{tail}(f_j)$, where $f_j \mid P(x_j)$
  - All non-PI variables $x_j$ can be canceled in this reduction
  - Reduction results in GB $G$ with only primary input variables, potentially explosive

This approach should work for specification polynomials $f$ with low degree terms
## Table: Verification Results of SAT, SMT, BDD, ABC.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Word size of the operands $k$-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>MiniSAT</td>
<td>22.55</td>
</tr>
<tr>
<td>CryptoMiniSAT</td>
<td>7.17</td>
</tr>
<tr>
<td>PrecoSAT</td>
<td>7.94</td>
</tr>
<tr>
<td>PicoSAT</td>
<td>14.85</td>
</tr>
<tr>
<td>Yices</td>
<td>10.48</td>
</tr>
<tr>
<td>Beaver</td>
<td>6.31</td>
</tr>
<tr>
<td>CVC</td>
<td>TO</td>
</tr>
<tr>
<td>Z3</td>
<td>85.46</td>
</tr>
<tr>
<td>Boolector</td>
<td>5.03</td>
</tr>
<tr>
<td>SimplifyingSTP</td>
<td>14.66</td>
</tr>
<tr>
<td>ABC</td>
<td>242.78</td>
</tr>
<tr>
<td>BDD</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Experimental Results: Correctness Proof

Verify a specification polynomial $f$ against a circuit $C$ by performing the test $f \xrightarrow{J+J_0} 0$?

Table: Verify bug-free and buggy Mastrovito multipliers. SINGULAR computer algebra tool used for division.

<table>
<thead>
<tr>
<th>Size k-bits</th>
<th>32</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>160</th>
<th>163</th>
</tr>
</thead>
<tbody>
<tr>
<td>#variables</td>
<td>1155</td>
<td>4355</td>
<td>9603</td>
<td>16899</td>
<td>26243</td>
<td>27224</td>
</tr>
<tr>
<td>#polynomials</td>
<td>1091</td>
<td>4227</td>
<td>9411</td>
<td>16643</td>
<td>25923</td>
<td>26989</td>
</tr>
<tr>
<td>#terms</td>
<td>7169</td>
<td>28673</td>
<td>64513</td>
<td>114689</td>
<td>179201</td>
<td>185984</td>
</tr>
<tr>
<td>Compute-GB:</td>
<td>93.80</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
</tr>
<tr>
<td>Ours: Bug-free</td>
<td>1.41</td>
<td>112.13</td>
<td>758.82</td>
<td>3054</td>
<td>9361</td>
<td>16170</td>
</tr>
<tr>
<td>Ours: Bugs</td>
<td>1.43</td>
<td>114.86</td>
<td>788.65</td>
<td>3061</td>
<td>9384</td>
<td>16368</td>
</tr>
</tbody>
</table>

Why does Compute-GB (SINGULAR) run out of memory?
Improve GB-reduction: $F_4$-style reduction

New algorithm to compute a Gröbner basis by J.C. Faugère: $F_4$

- Buchberger’s algorithm $S(f, g) \xrightarrow{G} + r$
- Instead, compute a “set” of $S(f, g)$ in one-go
- Reduces them “simultaneously”
- Significant speed-up in computing a Gröbner basis
- Models the problem using sparse linear algebra
- Gaussian elimination on a matrix representation of the problem

Our term order: already a Gröbner basis. We only need $F_4$-style reduction:

\[ f \xrightarrow{F, F_0} + r \]
Objective: $f : Z + A \cdot B$, compute $f \xrightarrow{f_1, \ldots, f_s} + r$

Find a polynomial $f_i$ that divides $f$, or “cancels” $LT(f)$

Construct a matrix: rows = polynomials, columns = monomials, entries = coefficient of monomial present in the polynomial

- This matrix is constructed iteratively
- The specification polynomial $f$ is inserted into the first row
- Maintain the specified term order in the matrix
- Iterate over $i$, the list of monomials generated/utilized in the division process
- Find a polynomial $f_j$ s.t. $lt(f_j)$ cancels the $i^{th}$ monomial (column) of the matrix
- Insert $\frac{X_i}{lm(f_j)} \cdot f_j$ as the new row in the matrix
- Update the entries in the matrix subject to the term order
Matrix Construction with an Example

Given \( f : Z + AB \) over \( \mathbb{F}_2 \) with \( P(\alpha) = \alpha^2 + \alpha + 1 = 0 \).

Polynomials of the circuit, corresponding to ideal \( J \):
\[
\begin{align*}
    f_1 : A + a_0 + a_1 \alpha, & \quad f_2 : B + b_0 + b_1 \alpha, & \quad f_3 : Z + z_0 + z_1 \alpha, \\
    f_4 : r_0 + a_0 b_1 + a_1 b_0, & \quad f_5 : z_0 + a_0 b_0 + a_1 b_1, & \quad f_6 : z_1 + r_0 + a_1 b_1
\end{align*}
\]

Compute \( f \xrightarrow{f_1, \ldots, f_6} + r \)

Term order: LEX with \( Z > A > B > z_0 > z_1 > r_0 > a_0 > a_1 > b_0 > b_1 \)
Problem setup:

- Insert $f : Z + AB$ as the first row of the matrix $M$
- Note that $Z > AB$ in our monomial order
- Let $M_L$ denote the list of monomials; these will correspond to the columns of the matrix $M$
- Matrix $M$ at the first step:

\[
\begin{pmatrix}
Z & AB \\
f & 1 & 1
\end{pmatrix}
\]
Matrix Construction (Contd.)

- Set $i = 1$
- Find a polynomial $f_j$ from $f_1, \ldots, f_6$ s.t. $lm(f_j) \mid \text{monomial}[i]$ represented in the $i^{th}$ column
- Clearly, $f_j = f_3 = Z + z_0 + z_1\alpha$
- Division: $f \xrightarrow{f_j} r = f - \frac{lt(f)}{lt(f_j)} \cdot f_j = f - \frac{lc(f)}{lc(f_j)} \frac{lm(f)}{lm(f_j)} \cdot f_j$
- Ignore the coefficients, they will be resolved/computed as coefficients in the matrix $M$
- Compute: $f \xrightarrow{f_j} r = f - \frac{lm(f)}{lm(f_j)} \cdot f_j$
  - The computation $(Z + AB) - \frac{Z}{Z} \cdot (Z + z_0 + z_1\alpha)$ gives the monomial list as $AB, z_0, z_1$
  - These monomials will correspond to the columns of the matrix
- List of monomials $M_L = M_L \cup \frac{lm(f)}{lm(f_j)} \cdot f_j$

\[
\begin{array}{cccc}
Z & AB & z_0 & z_1 \\
1 & 1 & 0 & 0 \\
1 & 0 & 1 & \alpha
\end{array}
\]
Set \( i = i + 1 = 2 \)

Find a polynomial \( f_j \) from \( f_1, \ldots, f_6 \) s.t. \( \text{lm}(f_j) \mid \text{monomial}[i] \) represented in the \( i^{th} \) column

\( \text{monomial}[2] = AB \)

Clearly, \( f_j = f_1 = A + a_0 + a_1\alpha \)

Monomials required in cancellation (division) of \( AB \)

\[
\frac{AB}{\text{lm}(f_j)} \cdot f_j = (AB) - B(A + a_0 + a_1\alpha)
\]

Only interested in the monomials utilized in this division process

Update \( M_L = M_L \cup \{ \text{monomials of } \frac{AB}{A} \cdot f_1 \} \)

\[
\begin{bmatrix}
Z & AB & B a_0 & B a_1 & z_0 & z_1 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & \alpha \\
0 & 1 & 1 & \alpha & 0 & 0
\end{bmatrix}
\]
Continue in this fashion: $F_4$-style reduction

- Construct the whole matrix $M$
- $M$ is completed when monomial ordering reaches primary inputs
- Rows $= \frac{\text{monomials}}{\text{lm}(f_j)} \cdot f_j$; Columns $= M_L$, $M(i, j) =$ coefficients

\[
\begin{bmatrix}
Z & AB & B_{a0} & B_{a1} & z_0 & z_1 & r_0 & a_0b_0 & a_0b_1 & a_1b_0 & a_1b_1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & \alpha & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \alpha & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \alpha & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\end{bmatrix}
\]
$F_4$-style reduction

- Construct the matrix $M$ for polynomial reduction
- Apply Gaussian elimination on $M$
- Last row = remainder $r$ = result of reduction $= \alpha^2 + \alpha + 1 = 0$

\[
\begin{pmatrix}
Z & AB & Ba_0 & Ba_1 & z_0 & z_1 & r_0 & a_0b_0 & a_0b_1 & a_1b_0 & a_1b_1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \alpha & 1 & \alpha & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \alpha & 1 & \alpha & 0 & 1 & \alpha & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \alpha & 0 & 1 & \alpha & \alpha & \alpha^2 \\
0 & 0 & 0 & 0 & 0 & \alpha & 0 & 0 & \alpha & \alpha & \alpha^2 + 1 \\
0 & 0 & 0 & 0 & 0 & 0 & \alpha & 0 & \alpha & \alpha & \alpha^2 + \alpha + 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 + \alpha + 1 \\
\end{pmatrix}
\]
Algorithm for this reduction [2]

**Input:** $f, F = \{f_1, \ldots, f_s\}$, term order $>$

**Output:** A matrix $M$ representing $f \xrightarrow{f_1, \ldots, f_s}_+ r$

/*$L = \text{set of polynomials, rows of } M$/;\n$L := \{f\}; i := 1;$\n$M_L := \{\text{monomials of } f\}; // M_L = \text{the set of monomials, columns of } M;$\n$\text{mon} := \text{the } i^{th} \text{ monomial of } M_L;$

**while** $\text{mon} \notin \text{PrimaryInputs} \text{ do}$

- Identify $f_k \in F$ satisfying: $\text{lm}(f_k)$ can divide $\text{mon}$
  /*add polynomial $f_k$ to L as a new row in $M$*/
  $L := L \cup \{\text{mon}\text{lm}(f_k) \cdot f_k\}$

- /*Add monomials to $M_L$ as new columns in $M$*/
  $M_L := M_L \cup \{\text{monomials of } \text{mon}\text{lm}(f_k) \cdot f_k\}$
  $i := i + 1;$
  $\text{mon} := \text{the } i^{th} \text{ monomial of } M_L;$

**end**

Gaussian Elimination on $M$;

**return** $r = \text{last row of } M$;

**Algorithm 1:** Generating the Matrix for Polynomial Reduction
### Results

**Table:** Runtime for verifying bug-free and buggy Montgomery multipliers. TO = timeout of 10hrs. Time is given in seconds. * denotes SINGULAR’s capacity exceeded.

<table>
<thead>
<tr>
<th>Operand size $k$</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>96</th>
<th>128</th>
<th>163</th>
</tr>
</thead>
<tbody>
<tr>
<td>#variables</td>
<td>1194</td>
<td>2280</td>
<td>4395</td>
<td>6562</td>
<td>14122</td>
<td>91246</td>
</tr>
<tr>
<td>#polynomials</td>
<td>1130</td>
<td>2184</td>
<td>4267</td>
<td>6370</td>
<td>13866</td>
<td>89917</td>
</tr>
<tr>
<td>#terms</td>
<td>10741</td>
<td>18199</td>
<td>40021</td>
<td>55512</td>
<td>134887</td>
<td>484738</td>
</tr>
<tr>
<td>Bug-free (Singular)</td>
<td>1.50</td>
<td>11.03</td>
<td>27.70</td>
<td>1802.75</td>
<td>10919</td>
<td>*</td>
</tr>
<tr>
<td>Bug-free ($F_4$)</td>
<td>0.86</td>
<td>4.47</td>
<td>10.11</td>
<td>700.59</td>
<td>4539</td>
<td>18374</td>
</tr>
<tr>
<td>Bugs (Singular)</td>
<td>1.52</td>
<td>11.10</td>
<td>28.18</td>
<td>1812.15</td>
<td>11047</td>
<td>*</td>
</tr>
<tr>
<td>Bugs ($F_4$)</td>
<td>0.88</td>
<td>4.49</td>
<td>10.12</td>
<td>709.03</td>
<td>4564</td>
<td>17803</td>
</tr>
</tbody>
</table>

$F_4$-style reduction 2.5X faster than use of Singular
Faugère’s motivation

- In practical GB computation, problems have sparsity
  - Look at our matrix $M$, it is full of 0s
- Such matrices usually have block-triangularity
- Rows of $M$ are often monomial multiples of the same polynomials
- Use “sparse linear algebra”

Further improvements possible: Certainly a MS thesis project

- Matrix based reduction can be parallelized: General Purpose GPU (GP-GPU) computing
- Complexity = construction of $M$, use of a symbol/hash table to search for $f_j$ s.t. $lm(f_j) | \text{monomial}[i]$
In Conclusion

The Key to Success in Design Automation

- Build algorithms and techniques on solid theoretical foundations
- Use all of the mathematical tools at your disposal
- Make sure to exploit circuit structure
- Develop domain-specific implementations
- That’s what SAT, BDDs, AIGs do too!

