SAT, COMPUTER ALGEBRA, MULTIPLIERS

Armin Biere joint work with Daniela Kaufmann and Manuel Kauers

6th Vampire Workshop
Lisbon, Portugal, July 7, 2019
Multiplication

\[
1 \quad 3 \quad \cdot \quad 1 \quad 5
\]

\[
\underline{15}
\]
Multiplication

\[
\begin{array}{cc}
1 & 3 \\
\cdot & 1 5 \\
\hline
6 & 5
\end{array}
\]
Multiplication

\[
\begin{array}{c c}
1 & 3 \\
\cdot & 1 & 5 \\
\hline
& 6 & 5 \\
& 1 & 3 \\
\hline
\end{array}
\]
Multiplication

\[
\begin{array}{ccc}
1 & 3 & \cdot \\
\hline
& 6 & 5 \\
1 & 3 & 0 \\
\hline
& 0 & 0 \\
\end{array}
\]

5
Multiplication

\[
\begin{array}{c}
1 & 3 \\
\cdot & 1 & 5 \\
\hline
 & 6 & 5 \\
 & 1 & 3 \\
\hline
1 & 3 & 0 & 0 \\
\hline
9 & 5 \\
\end{array}
\]
Multiplication

\[
\begin{array}{ccc}
1 & 3 & \cdot \\ & & \\
1 & 5 & \\
\hline
& & 6 & 5 \\
& & 1 & 3 \\
& & 0 & 0 & 0 \\
\hline
& & 1 & 9 & 5
\end{array}
\]
Binary multiplication

\[ \begin{array}{cccccc}
1 & 1 & 0 & 1 & . & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

\[ \begin{array}{cccccc}
13 & \cdot & 15 & = & 195 \\
\end{array} \]
Binary multiplication

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\times & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
\end{array}
\]

13 \cdot 15 = 195
Binary multiplication

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\cdot & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
13 \cdot 15 &=& 195
\end{array}
\]
Binary multiplication

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
13 \cdot 15 = 195
\end{array}
\]
Binary multiplication

\[
\begin{array}{c}
1 & 1 & 0 & 1 \\
\times & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
13 \cdot 15 = 195
\end{array}
\]
Binary multiplication

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 & \\
1 & 1 & 0 & 1 & \\
1 & 1 & 0 & 1 & \\
1 & 1 & 0 & 1 & \\
\end{array}
\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
0 & 0 \\
\hline
1 & 1 & 1
\end{array}
\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\hline
0 & 1 & 1 & 1 \\
\end{array}
\]

\[
13 \cdot 15 = 195
\]
Binary multiplication

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
0 & 0 & 1 & 1
\end{array}
\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\cdot & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

\[= 0 0 0 1 1\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
& & & & & 1 & 1 & 0 & 1 \\
& & & & 1 & 1 & 0 & 1 \\
& & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
& & & & & 2 & 2 & 2 & 1 & 0 & 0 \\
\hline
& & & & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\cdot & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 2 & 2 & 2 \\
1 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[13 \cdot 15 = 195\]
Binary multiplication

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 & & & & & & \\
1 & 1 & 0 & 1 & & & & & & \\
1 & 1 & 0 & 1 & & & & & & \\
1 & 1 & 0 & 1 & & & & & & \\
\hline
1 & 2 & 2 & 2 & 1 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[13 \cdot 15 = 195\]
Example: 2 Bit - Binary Multiplication

\[
\begin{array}{ccc}
1 & 1 & \cdot \\
1 & 1 & 1 \\
\hline
1 & 1 & 1 \\
1 & 1 & 1 \\
\hline
1 & 0 & 0 & 1
\end{array}
\]

3 \cdot 3 = 9
Example: 2 Bit - Binary Multiplication

**AND-Gate**

<table>
<thead>
<tr>
<th>f</th>
<th>g</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**XOR-Gate**

<table>
<thead>
<tr>
<th>f</th>
<th>g</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Motivation & Solving Techniques

**Given:** Gate-level multiplier for fixed bit-width $n$.

**Question:** For all possible $a_i, b_i \in \mathbb{B}$:

$$(2a_1 + a_0) \times (2b_1 + b_0) = 8s_3 + 4s_2 + 2s_1 + s_0?$$

**Solving Techniques**

- SAT using CNF encoding
- Binary Moment Diagrams (BMD)
- Algebraic reasoning
Previous Work

- **SAT using CNF encoding**

- **Binary moment diagrams**

- **Algebraic reasoning**
  - J. Lv, P. Kalla, and F. Enescu. Efficient Gröbner basis reductions for formal verification of *Galois field arithmetic circuits*. In IEEE TCAD, 2013.
  - C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski. Formal verification of arithmetic circuits by *function extraction*. In IEEE TCAD, 2016.
Recent Work


- D. Kaufmann, A. Biere, and M. Kauers. Incremental Column-wise verification of arithmetic circuits using computer algebra. FMSD, Feb 2019


Basic Idea of Algebraic Approach

Multiplier

\[ a_0 b_0, a_1 b_0, a_0 b_1, a_1 b_1 \]

Polynomials

\[ B = \{ x - a_0 \cdot b_0, \ y - a_1 \cdot b_1, \ s_0 - x \cdot y, \ \ldots \} \]

Specification

\[ \sum_{i=0}^{2n-1} 2^i s_i - (\sum_{i=0}^{n-1} 2^i a_i)(\sum_{i=0}^{n-1} 2^i b_i) \]

Ideal Membership Test

\[ = 0 \text{ ✓} \]
\[ \neq 0 \times \]
Polynomials

\[ p = c_1 \tau_1 + \ldots + c_m \tau_m \in \mathbb{Q}[X] = \mathbb{Q}[x_1, \ldots, x_n] \]

- \( \mathbb{Q}[X] \) is the **ring of polynomials** with variables \( X = x_1, \ldots, x_n \) and coefficients in \( \mathbb{Q} \).
- A **term** \( \tau_i \) is a product \( x_1^{e_1} \cdots x_n^{e_n} \) with \( e_j \geq 0 \).
- A **monomial** \( c_i \tau_i \) is a constant multiple of a term with \( c_i \in \mathbb{Q} \).
- A **polynomial** \( p \) is a finite sum of monomials.
Polynomials

\[ p = c_1 \tau_1 + \ldots + c_m \tau_m \in \mathbb{Q}[X] = \mathbb{Q}[x_1, \ldots, x_n] \]

- We fix a **term order** such that for all terms \( \tau, \sigma_1, \sigma_2 \) we have \( x_1^0 \cdots x_n^0 = 1 \leq \tau \) and \( \sigma_1 \leq \sigma_2 \Rightarrow \tau \sigma_1 \leq \tau \sigma_2 \).
- An order is a **lexicographic term order** if for all \( \sigma_1 = x_1^{u_1} \cdots x_n^{u_n}, \sigma_2 = x_1^{v_1} \cdots x_n^{v_n} \) we have \( \sigma_1 < \sigma_2 \) iff there exists an index \( i \) with \( u_j = v_j \) for all \( j < i \), and \( u_i < v_i \).

- \( \text{lm}(p) = c_1 \tau_1 \) is the **leading monomial** of \( p \).
- \( \text{lt}(p) = \tau_1 \) is the **leading term** of \( p \).
- \( p - \text{lm}(p) \) is the **tail** of \( p \).
Ideals

Ideal. A nonempty subset $I \subseteq \mathbb{Q}[X]$ is called an ideal if

$$\forall p, q \in I : p + q \in I \quad \text{and} \quad \forall p \in \mathbb{Q}[X] \forall q \in I : pq \in I$$

Basis. A set $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{Q}[X]$ is called a basis of an ideal $I$ if

$$I = \{q_1p_1 + \cdots + q_mp_m \mid q_1, \ldots, q_m \in \mathbb{Q}[X]\} = \langle P \rangle$$

$I$ is the set of polynomials which become zero, when the elements of $P$ become zero.
Circuit Polynomials

Gate Polynomials.

\[ s_3 = g_1 \land g_4 \quad -s_3 + g_1 g_4, \]
\[ s_2 = g_1 \oplus g_4 \quad -s_2 + g_1 + g_4 - 2g_1 g_4, \]
\[ g_4 = g_2 \land g_3 \quad -g_4 + g_2 g_3, \]
\[ s_1 = g_2 \oplus g_3 \quad -s_1 + g_2 + g_3 - 2g_2 g_3, \]
\[ g_1 = a_1 \land b_1 \quad -g_1 + a_1 b_1, \]
\[ g_2 = a_0 \land b_1 \quad -g_2 + a_0 b_1, \]
\[ g_3 = a_1 \land b_0 \quad -g_3 + a_1 b_0, \]
\[ s_0 = a_0 \land b_0 \quad -s_0 + a_0 b_0 \]

Boolean Value Polynomials.

\[ a_1, a_0 \in \mathbb{B} \quad a_1 (1 - a_1), \ a_0 (1 - a_0), \]
\[ b_1, b_0 \in \mathbb{B} \quad b_1 (1 - b_1), \ b_0 (1 - b_0) \]
Ideals associated to Circuits

Polynomial Circuit Constraints (PCCs).

A polynomial \( p \in \mathbb{Q}[X] \) such that for all

\[(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}) \in \{0, 1\}^{2n}\]

and resulting values \( g_1, \ldots, g_k, s_0, \ldots, s_{2n-1} \) implied by the gates of the circuit \( C \) the substitution of these values into \( p \) gives zero.

- The set of all PCCs is denoted by \( I(C) \).
- \( I(C) \) contains all relations of the circuit.
- \( I(C) \) is an ideal.
Ideals associated to Circuits

Examples for PCCs:

- $s_0 - a_0 b_0$ and gate
- $a_1^2 - a_1$ $a_1$ boolean
- $g_2^2 - g_2$ $g_2$ boolean
- $s_1 g_4$ xor-and constraint
- ... 

**Multiplier.** A circuit $C$ is called a multiplier if

\[
\sum_{i=0}^{2n-1} 2^i s_i - \left( \sum_{i=0}^{n-1} 2^i a_i \right) \left( \sum_{i=0}^{n-1} 2^i b_i \right) \in I(C).
\]
Ideal Membership Test

**Ideal membership problem.** Given a polynomial \( f \in \mathbb{Q}[X] \) and an ideal \( I = \langle g_1, \ldots, g_m \rangle = \langle G \rangle \subseteq \mathbb{Q}[X] \), determine if \( f \in I \).

Given arbitrary basis \( G \) of \( I \) it is not obvious how to solve ideal membership problem.

**Lemma (Ideal membership test)**

Let \( G = \{g_1, \ldots, g_m \} \subseteq \mathbb{Q}[X] \) be a Gröbner basis, and let \( f \in \mathbb{Q}[X] \). Then \( f \) is contained in the ideal \( I = \langle G \rangle \) iff the unique remainder of \( f \) with respect to \( G \) is zero.
Gröbner basis

- Every ideal $I \subseteq \mathbb{Q}[X]$ has a **Gröbner basis** w.r.t. a fixed term order.

- Construction algorithm by Buchberger which given an arbitrary basis of an ideal $I$ computes a Gröbner basis of it (double exponential)

- Algorithm is based on repeated reduction of so-called S-polynomials ($spol$).

- A basis $G$ is a Gröbner basis of $I = \langle G \rangle$ if for all $p, q \in G$ : $spol(p, q)$ reduces to zero.

- **Product criterion.** If $p, q \in \mathbb{Q}[X] \setminus \{0\}$ are such that the leading terms are coprime, i.e., $\text{lcm}(\text{lt}(p), \text{lt}(q)) = \text{lt}(p) \text{lt}(q)$, then $spol(p, q)$ reduces to zero.
Circuit Gröbner basis

We can deduce at least some elements of \( I(C) \):

- \( G = \text{Gate Polynomials} + \text{Boolean Value Polynomials} \)
- Let \( J(C) = \langle G \rangle \).
- Lexicographic term order: output variable of a gate is greater than input variables

**Theorem**

\( G \) is a Gröbner basis for \( J(C) \).

Proof idea: Application of Buchberger’s Product criterion.
Circuit Polynomials

Gate Polynomials.

\[ s_3 = g_1 \land g_4 \quad -s_3 + g_1 g_4, \]
\[ s_2 = g_1 \oplus g_4 \quad -s_2 - 2g_1 g_4 + g_4 + g_1, \]
\[ g_4 = g_2 \land g_3 \quad -g_4 + g_2 g_3, \]
\[ s_1 = g_2 \oplus g_3 \quad -s_1 - 2g_2 g_3 + g_2 + g_3, \]
\[ g_1 = a_1 \land b_1 \quad -g_1 + a_1 b_1, \]
\[ g_2 = a_0 \land b_1 \quad -g_2 + a_0 b_1, \]
\[ g_3 = a_1 \land b_0 \quad -g_3 + a_1 b_0, \]
\[ s_0 = a_0 \land b_0 \quad -s_0 + a_0 b_0 \]

Boolean Value Polynomials.

\[ a_1, a_0 \in \mathbb{B} \quad -a_1^2 + a_1, \quad -a_0^2 + a_0, \]
\[ b_1, b_0 \in \mathbb{B} \quad -b_1^2 + b_1, \quad -b_0^2 + b_0 \]
Soundness and completeness

Theorem (Soundness and completeness)

For all acyclic circuits $C$, we have $J(C) = I(C)$.

- $J(C) \subset I(C)$: soundness
- $I(C) \subset J(C)$: completeness
Non-Incremental Checking Algorithm

Divide polynomial

\[ \sum_{i=0}^{2n-1} 2^i s_i - \left( \sum_{i=0}^{n-1} 2^i a_i \right) \left( \sum_{i=0}^{n-1} 2^i b_i \right) \]

by elements of \( G \) until no further reduction is possible, then \( C \) is a multiplier iff remainder is zero.

Implications:

- Leading term is one variable, division is actually substitution by tail.
- Leading coefficient \( \pm 1 \) of all gate polynomials, computation stays in \( \mathbb{Z} \).
- Still can use rational coefficients \( \mathbb{Q} \) (important for Singular).
- Completeness proof allows to derive input assignment if \( C \) is incorrect.
Example: 2 Bit - Binary Multiplication

\[ G = \{ \]
\[-s_3 + g_1 g_4, \]
\[-s_2 + g_1 + g_4 - 2g_1 g_4, \]
\[-g_4 + g_2 g_3, \]
\[-s_1 + g_2 + g_3 - 2g_2 g_3, \]
\[-g_1 + a_1 b_1, \]
\[-g_2 + a_0 b_1, \]
\[-g_3 + a_1 b_0, \]
\[-s_0 + a_0 b_0, \]
\[-a_1^2 + a_1, \]
\[-a_0^2 + a_0, \]
\[-b_1^2 + b_1, \]
\[-b_0^2 + b_0 \}\]

\[ 8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \]
Example: 2 Bit - Binary Multiplication

\[ G = \{ \]
\[ -s_3 + g_1g_4, \]
\[-s_2 + g_1 + g_4 - 2g_1g_4, \]
\[-g_4 + g_2g_3, \]
\[-s_1 + g_2 + g_3 - 2g_2g_3, \]
\[-g_1 + a_1b_1, \]
\[-g_2 + a_0b_1, \]
\[-g_3 + a_1b_0, \]
\[-s_0 + a_0b_0, \]
\[-a_1^2 + a_1, \]
\[-a_0^2 + a_0, \]
\[-b_1^2 + b_1, \]
\[-b_0^2 + b_0 \} \]

\[ 8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1b_1 - 2a_1b_0 - 2a_0b_1 - a_0b_0 \]

\[ 8g_1g_4 + 4s_2 + 2s_1 + s_0 - 4a_1b_1 - 2a_1b_0 - 2a_0b_1 - a_0b_0 \]
Example: 2 Bit - Binary Multiplication

\[ G = \{ \]
\[-s_3 + g_1 g_4, \]
\[-s_2 + g_1 + g_4 - 2 g_1 g_4, \]
\[-g_4 + g_2 g_3, \]
\[-s_1 + g_2 + g_3 - 2 g_2 g_3, \]
\[-g_1 + a_1 b_1, \]
\[-g_2 + a_0 b_1, \]
\[-g_3 + a_1 b_0, \]
\[-s_0 + a_0 b_0, \]
\[-a_1^2 + a_1, \]
\[-a_0^2 + a_0, \]
\[-b_1^2 + b_1, \]
\[-b_0^2 + b_0 \} \]

\[ 8 s_3 + 4 s_2 + 2 s_1 + s_0 - 4 a_1 b_1 - 2 a_1 b_0 - 2 a_0 b_1 - a_0 b_0 \]
\[ 8 g_1 g_4 + 4 s_2 + 2 s_1 + s_0 - 4 a_1 b_1 - 2 a_1 b_0 - 2 a_0 b_1 - a_0 b_0 \]
\[ 8 g_1 g_4 + 4 (g_1 + g_4 - 2 g_1 g_4) + 2 s_1 + s_0 \]
\[ - 4 a_1 b_1 - 2 a_1 b_0 - 2 a_0 b_1 - a_0 b_0 \]
Example: 2 Bit - Binary Multiplication

\[ G = \{ \]
\[ -s_3 + g_1 g_4, \]
\[ -s_2 + g_1 + g_4 - 2g_1 g_4, \]
\[ -g_4 + g_2 g_3, \]
\[ -s_1 + g_2 + g_3 - 2g_2 g_3, \]
\[ -g_1 + a_1 b_1, \]
\[ -g_2 + a_0 b_1, \]
\[ -g_3 + a_1 b_0, \]
\[ -s_0 + a_0 b_0, \]
\[ -a_1^2 + a_1, \]
\[ -a_0^2 + a_0, \]
\[ -b_1^2 + b_1, \]
\[ -b_0^2 + b_0 \} \]

\[ 8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \]
\[ 8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \]
\[ 8g_1 g_4 + 4(g_1 + g_4 - 2g_1 g_4) + 2s_1 + s_0 \]
\[ - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \]
\[ \vdots \]
\[ 0 \]
Computational Issues

Generally the number of monomials in the intermediate results increases drastically:

- 8-bit multiplier can not be verified within 20 minutes.

Tailored heuristics become very important:

- Choose appropriate term order.
- Divide verification problem into smaller sub-problems.
- Rewrite and thus simplify Gröbner basis $G$. 
Order

Row-Wise

$$\left(4a_2 + 2a_1 + a_0\right) \times \left(4b_2 + 2b_1 + b_0\right)$$

Column-Wise

$$\left(4a_2 + 2a_1 + a_0\right) \times \left(4b_2 + 2b_1 + b_0\right)$$
Slicing

Partial Products. Let \( P_k = \sum_{k=i+j} a_i b_j \).

Input Cone. For each output bit \( s_i \) we determine its input cone

\[
I_i = \{ \text{gate } g \mid g \text{ is in input cone of output } s_i \}
\]

Slice. Slices \( S_i \) are defined as the difference of consecutive cones \( I_i \):

\[
S_0 = I_0 \quad S_{i+1} = I_{i+1} \setminus \bigcup_{j=0}^{i} S_j
\]

Sliced Gröbner Bases. Let \( G_i \) be the set of gate and boolean value polynomials in \( S_i \).
Carry Recurrence Relation

A sequence of $2n + 1$ polynomials $C_0, \ldots, C_{2n}$ is called a carry sequence if

$$-C_i + 2C_{i+1} + s_i - P_i \in I(C) \quad \text{for all } 0 \leq i < 2n + 1.$$ 

Then $R_i = -C_i + 2C_{i+1} + s_i - P_i$ are the carry recurrence relations for $C_0, \ldots, C_{2n}$.

**Theorem**

Let $C$ be a circuit where all carry recurrence relations are contained in $I(C)$. Then $C$ is a multiplier, iff $C_0 - 2^{2n}C_{2n} \in I(C)$. 

Incremental Checking Algorithm.

input: Circuit $C$ with sliced Gröbner bases $G_i$
output: Determine whether $C$ is a multiplier

$C_{2n} \leftarrow 0$

for $i \leftarrow 2n - 1$ to 0

$C_i \leftarrow \text{Remainder} \left( 2C_{i+1} + s_i - P_i, G_i \right)$

return $C_0 = 0$
Multiplier
Multipliers as And-Inverter-Graph

Fulladder
Halfadder
XOR-Gate
Single Gates
Variable Elimination

Identify sub-circuits $C_S$ in the AIG and eliminate internal variables:

- Full-adder rewriting
- Half-adder rewriting
- XOR- Rewriting
- Common-Rewriting

Variable elimination is based on elimination theory of Gröbner bases.
Elimination theory of Gröbner bases

Elimination order. Let $X = Y \cup Z$ and we want to eliminate $Z$. Order the terms such that for all terms $\sigma, \tau$ where a variable from $Z$ is contained in $\sigma$ but not in $\tau$, we obtain $\tau < \sigma$.

Elimination ideal. The elimination ideal $J$ where the $Z$-variables are eliminated of $I \subseteq \mathbb{Q}[X] = \mathbb{Q}[Y, Z]$ is defined by

$$J = I \cap \mathbb{Q}[Y].$$

Elimination theorem. Given an ideal $I \subseteq \mathbb{Q}[X] = \mathbb{Q}[Y, Z]$. Further let $G$ be a Gröbner basis of $I$ with respect to an elimination order $Y < Z$. Then the set

$$H = G \cap \mathbb{Q}[Y]$$

is a Gröbner basis of the elimination ideal $J = I \cap \mathbb{Q}[Y]$, in particular $\langle H \rangle = J$. 
Elimination procedure

**Problem:** Computing a Gröbner basis $H$ for $I(C)$ w.r.t an elimination order is costly.

**Solution:** Split $G$ into two parts.

\[
\begin{align*}
\text{Step 1:} & \quad \text{original Gröbner basis } G \\
\text{Step 2:} & \quad \text{split } G \text{ into two subbases} \\
\text{Step 3:} & \quad \text{change order of } <_G \text{ to } <_H \\
\text{Step 4:} & \quad \text{eliminate the variables of } Z \\
\text{Step 5:} & \quad \text{rejoin bases } H = G_A \cup H_Y
\end{align*}
\]
Elimination procedure

**Theorem**

Let $G \subseteq \mathbb{Q}[X] = \mathbb{Q}[Y, Z]$ be a Gröbner basis with respect to some term order $<_G$. Let $G_A = G \cap \mathbb{Q}[Y]$ and $G_B = G \setminus G_A$. Let $<_H$ be an elimination order for $Z$ which agrees with $<_G$ for all terms that are free of $Z$, i.e., terms free of $Z$ are equally ordered in $<_G$ and $<_H$. Suppose that $\langle G_B \rangle$ has a Gröbner basis $H_B$ with respect to $<_H$ which is such that every leading term in $H_B$ is free of $Z$ or free of $Y$. Then $\langle G \rangle \cap \mathbb{Q}[Y] = (\langle G_A \rangle + \langle G_B \rangle) \cap \mathbb{Q}[Y] = \langle G_A \rangle + (\langle G_B \rangle \cap \mathbb{Q}[Y])$.

**Theorem**

Let $G, G_A, G_B, H_B, H_Y, H_Z, <_H, <_G$ be as before. Then $H = G_A \cup H_Y$ is a Gröbner basis w.r.t. the ordering $<_H$. 

Example: Full-Adder Rewriting

\[ G_A = G \setminus G_B \]
\[ G_B = \{ -g_0 + p_{20} + p_{11} - 2p_{20}p_{11}, -g_1 + p_{20}p_{11}, -g_2 + c_1 g_0, -s_2 + c_1 + g_0 - 2c_1 g_0, -c_2 + g_1 + g_2 - g_1 g_2 \} \]

Original lexicographic term ordering \(<_G>:\)
\[
\begin{align*}
    b_0 &< b_1 < a_0 < a_1 < a_2 < p_{00} < s_0 < p_{01} < p_{10} < s_1 < c_1 < \\
    p_{11} &< p_{20} < g_0 < g_1 < g_2 < s_2 < c_2 < p_{21} < s_3 < c_3 < s_4
\end{align*}
\]

Elimination order \(<_H>:\)
\[
\begin{align*}
    b_0 &< b_1 < a_0 < a_1 < a_2 < p_{00} < s_0 < p_{01} < p_{10} < s_1 < c_1 < \\
    p_{11} &< p_{20} < s_2 < c_2 < p_{21} < s_3 < c_3 < s_4 < g_0 < g_1 < g_2
\end{align*}
\]

Gröbner basis \(H_B\) w.r.t. elimination order \(<_H>:\)
\[
H_B = \{ g_0 + 2p_{20}p_{11} - p_{20} - p_{11}, g_1 - p_{20}p_{11}, \\
    g_2 + 2p_{20}p_{11}c_1 - p_{20}c_1 - p_{11}c_1, \\
    s_2 - 4p_{20}p_{11}c_1 + 2p_{20}p_{11} + 2p_{20}c_1 - p_{20} + 2p_{11}c_1 - p_{11} - c_1, \\
    2c_2 + s_2 - p_{20} - p_{11} - c_1 \} \]
Experiments

Multiplier

`AIG` to `AigMulToPoly`

Polynomials

\[
B = \{ \\
x - a_0 \ast b_0, \\
y - a_1 \ast b_1, \\
s_0 - x \ast y, \\
\ldots \\
\}
\]

Ideal Membership

`Mathematica`

`SINGULAR`

\[
C_0 = 0 \checkmark \\
C_0 \neq 0 \times
\]
Experiments

\[
\begin{array}{cccc}
S_7 & S_6 & S_5 & S_4 \\
\text{FA} & \text{FA} & \text{HA} & \text{FA} \\
\text{FA} & \text{FA} & \text{FA} & \text{HA} \\
\text{FA} & \text{FA} & \text{FA} & \text{FA} \\
\text{HA} & \text{FA} & \text{FA} & \text{FA} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>mult</th>
<th>( n )</th>
<th>Mathematica ( \text{non-inc} )</th>
<th>Mathematica ( \text{incremental} )</th>
<th>Mathematica ( +\text{xor} )</th>
<th>Mathematica ( +\text{xor} +\text{add} )</th>
<th>Singular ( \text{non-inc} )</th>
<th>Singular ( \text{incremental} )</th>
<th>Singular ( +\text{xor} )</th>
<th>Singular ( +\text{xor} +\text{add} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>non-inc</td>
<td>incremental</td>
<td></td>
<td>+xor</td>
<td>+xor +add</td>
<td>non-inc</td>
<td>incremental</td>
<td>+xor</td>
</tr>
<tr>
<td>btor</td>
<td>16</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>56</td>
<td>31</td>
<td>14</td>
<td>2</td>
<td></td>
<td>42</td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>MO</td>
<td>292</td>
<td>131</td>
<td>11</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>101</td>
<td>EE</td>
<td>EE</td>
<td>EE</td>
<td>EE</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>16</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>TO</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>326</td>
<td>171</td>
<td>30</td>
<td>2</td>
<td>TO</td>
<td>242</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>MO</td>
<td>TO</td>
<td>300</td>
<td>11</td>
<td>MO</td>
<td>EE</td>
<td>MO</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>102</td>
<td>EE</td>
<td>EE</td>
<td>EE</td>
<td>EE</td>
</tr>
</tbody>
</table>

Table: time in sec; TO = 1200 sec, MO = 14GB, EE=more than 32767 variables
Proofs

Multiplier

Polynomials

\[ B = \{ \]
\[ x - a_0 \cdot b_0, \]
\[ y - a_1 \cdot b_1, \]
\[ s_0 - x \cdot y, \]
\[ \ldots \} \]

Verification

\[ = 0 \checkmark \]
\[ \neq 0 \times \]

Correct?

Problem:

- Can we trust the CAS?
- Can we trust our own implementation of the optimizations?

Solution:

- Validate result of verification process [SC2'18]
- Generate machine-checkable proofs
- Check by independent proof checkers
Proofs

Problem:

- Can we trust the CAS?
- Can we trust our own implementation of the optimizations?

Solution:

- Generate machine-checkable proofs
- Check by independent proof checkers

Specification

\[
\sum_{i=0}^{2n-1} 2^i s_i - \left( \sum_{i=0}^{n-1} 2^i a_i \right) \left( \sum_{i=0}^{n-1} 2^i b_i \right) = 0 \checkmark \\
\neq 0 \times
\]

Verifications

Polynomials

\[
B = \{ \\
x - a_0 \ast b_0, \\
y - a_1 \ast b_1, \\
s_0 - x \ast y, \\
\ldots \\
\}
\]

Multiple

- Can we trust the CAS?
- Can we trust our own implementation of the optimizations?

Solution:

- Generate machine-checkable proofs
- Check by independent proof checkers

Correct?
Proofs

Polynomial calculus: Sequence \( P = (p_1, \ldots p_n) \), where each \( p_k \) is obtained by:

Addition \[
\frac{p_i}{p_i + p_j} \quad \forall \ p_i, p_j \in \langle G \rangle : p_i + p_j \in \langle G \rangle
\]

Multiplication \[
\frac{p_i}{qp_i} \quad \forall \ q \in \mathbb{Q}[X] \ \forall \ p_i \in \langle G \rangle : qp_i \in \langle G \rangle
\]
Proofs

Polynomial calculus: Sequence $P = (p_1, \ldots, p_n)$, where each $p_k$ is obtained by:

Addition

\[
\frac{p_i}{p_i + p_j} \quad \forall p_i, p_j \in \langle G \rangle : p_i + p_j \in \langle G \rangle
\]

Multiplication

\[
\frac{p_i}{qp_i} \quad \forall q \in \mathbb{Q}[X] \forall p_i \in \langle G \rangle : qp_i \in \langle G \rangle
\]

If $p_n = f$ we write $G \vdash f$ or in algebraic terms $f \in \langle G \rangle$.

Refutation: $G \vdash 1$ or $1 \in \langle G \rangle$. 
Example

\[
G = \{ -b + 1 - a, \quad b = -a \\
-c + ab, \quad c = a \land b = a \land \neg a \\
a^2 - a \} \\
a \in \mathbb{B}
\]

\[
f = c
\]

\[
\text{red}_G(f) = 0
\]

\[
* \quad \frac{-b + 1 - a}{-ab + a - a^2} \quad \frac{a^2 - a}{a^2 - a} \\
+ \quad \frac{-ab}{-ab} \quad \frac{-c + ab}{-c + ab} \\
* \quad \frac{-c}{c}
\]

\[
P = (-ab + a - a^2, -ab, -c, c)
\]
We translate the polynomial calculus into a concrete \textit{proof} format:

\begin{itemize}
  \item \textit{given} gate and boolean value polynomials serve as axioms
  \item instances of addition or multiplication \textit{rule} derive polynomials
\end{itemize}

\textbf{Practical Algebraic Calculus (PAC)}
allows automated proof checking
PAC Syntax

letter ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'
number ::= '0' | '1' | ... | '9'
count const ::= (number)^
variable ::= letter (letter | number)*
power ::= variable ['^' constant]
term ::= power ('*' power)*
monomial ::= constant | [ constant '*' ] term
operator ::= '+' | '-'
polynomial ::= [ '-' ] monomial (operator monomial)*
given ::= (polynomial ';' )* rule ::= ('+' | '*') ':' polynomial ',', polynomial ',', polynomial ';'
proof ::= (rule ';' )*
Example - PAC

\[ G = \{ -b + 1 - a, \quad -c + ab, \quad a^2 - a \} \]

\[ f = c \]

\[ * : \ -b+1-a, \quad a, \quad -a*b+a-a^2; \]

\[ + : \ -a*b+a-a^2, \quad a^2-a, \quad -a*b; \]

\[ + : \ -a*b, \quad -c+a*b, \quad -c; \]

\[ * : \ -c, \quad -1, \quad c; \]
Proof Checking

A proof rule contains four components:

\[ o : v, w, p; \]

Proof checking:

- **Connection property:** \( v, w \) are given polynomials or conclusions \( p_i \) of previous rules
- **Inference property:** verify correctness of each rule, e.g. \( p = v + w \) for \( o = "+" \)
- **Refutation check:** at least one \( p_i \) is a non-zero constant
Proof Checking Algorithm

**input**  
\( G \)  sequence of given polynomials  
\( r_1 \cdots r_k \) sequence of PAC proof rules

**output**  
“incorrect”, “correct-proof”, or “correct-refutation”

\[ P_0 \leftarrow G \]

\[ \text{for } i \leftarrow 1 \ldots k \]

\[ \text{let } r_i = (o_i, v_i, w_i, p_i) \]

**case** \( o_i = + \)

\[ \text{if } v_i \in P_{i-1} \land w_i \in P_{i-1} \land p_i = v_i + w_i \text{ then } P_i \leftarrow \text{append}(P_{i-1}, p_i) \]

**else return** “incorrect”

**case** \( o_i = * \)

\[ \text{if } v_i \in P_{i-1} \land p_i = v_i \ast w_i \text{ then } P_i \leftarrow \text{append}(P_{i-1}, p_i) \]

**else return** “incorrect”

\[ \text{for } i \leftarrow 1 \ldots k \]

\[ \text{if } p_i \text{ is a non zero constant polynomial then return } \text{“correct-refutation”} \]

**return** “correct-proof”
## Verifying Correctness of FMCAD’17 and DATE’18 Results

<table>
<thead>
<tr>
<th>$n$</th>
<th>mult</th>
<th>option</th>
<th>verification</th>
<th>proof generation</th>
<th>PACTRIM</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>btor</td>
<td>incr+elim</td>
<td>1</td>
<td>37</td>
<td>0</td>
<td>12738</td>
</tr>
<tr>
<td>32</td>
<td>btor</td>
<td>incr+elim</td>
<td>2</td>
<td>801</td>
<td>1</td>
<td>53122</td>
</tr>
<tr>
<td>64</td>
<td>btor</td>
<td>incr+elim</td>
<td>11</td>
<td>22378</td>
<td>4</td>
<td>216834</td>
</tr>
<tr>
<td>16</td>
<td>sparrc</td>
<td>incr+elim</td>
<td>1</td>
<td>112</td>
<td>20</td>
<td>17804</td>
</tr>
<tr>
<td><strong>32</strong></td>
<td><strong>sparrc</strong></td>
<td><strong>incr+elim</strong></td>
<td><strong>2</strong></td>
<td><strong>2611</strong></td>
<td><strong>2</strong></td>
<td><strong>75244</strong></td>
</tr>
<tr>
<td>64</td>
<td>sparrc</td>
<td>incr+elim</td>
<td>11</td>
<td>80906</td>
<td>12</td>
<td>309164</td>
</tr>
</tbody>
</table>
Array Ripple Carry Multiplier
Wallace-Tree Carry-Lookahead Multiplier
Checking Commutativity of Multiplication with SAT

<table>
<thead>
<tr>
<th>bits</th>
<th>Glucose</th>
<th>Lingeling</th>
<th>cube-and-conquer</th>
<th>Treengeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>05</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.13</td>
</tr>
<tr>
<td>06</td>
<td>0.02</td>
<td>0.03</td>
<td>0.36</td>
<td>0.31</td>
</tr>
<tr>
<td>07</td>
<td>0.14</td>
<td>0.27</td>
<td>0.63</td>
<td>0.72</td>
</tr>
<tr>
<td>08</td>
<td>1.18</td>
<td>1.98</td>
<td>2.38</td>
<td>2.47</td>
</tr>
<tr>
<td>09</td>
<td>7.85</td>
<td>10.98</td>
<td>2.63</td>
<td>4.65</td>
</tr>
<tr>
<td>10</td>
<td>37.16</td>
<td>41.49</td>
<td>5.02</td>
<td>10.86</td>
</tr>
<tr>
<td>11</td>
<td>147.62</td>
<td>214.98</td>
<td>15.72</td>
<td>21.96</td>
</tr>
<tr>
<td>12</td>
<td>833.62</td>
<td>649.49</td>
<td>56.57</td>
<td>61.48</td>
</tr>
<tr>
<td>13</td>
<td>------</td>
<td>-------</td>
<td>238.10</td>
<td>263.44</td>
</tr>
</tbody>
</table>

(limit of 900 seconds wall clock time)
Crux of Multiplier Verification for SAT
Adder Substitution

Partial Product Generation

Partial Product Accumulation

Final Stage Adder

Input

Output

Adder substitution

Partial Product Generation

Partial Product Accumulation

Final Stage Adder

Input

Output
Verification and Certification Flow

**Verify**
- AMulet substitution
- AMulet verify
- CaDiCaL
- X | ✓

**Certify**
- AMulet substitution
- AMulet certify
- CaDiCaL
- X | ✓
- drat-trim
- PacTrim
- .spec
- X | ✓
## Verification and Certification Time

<table>
<thead>
<tr>
<th>architecture</th>
<th>n</th>
<th>SPEC</th>
<th>nosat</th>
<th>nomod</th>
<th>noelim</th>
<th>Verify</th>
<th>MGD</th>
<th>YCM</th>
<th>CSYY</th>
<th>RBK</th>
<th>Certify</th>
<th>Check</th>
<th>total</th>
<th>proof size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sub sat aig tot</td>
<td>sub sat aig tot</td>
<td>sat aig tot</td>
<td>total</td>
<td>sub sat aig tot</td>
<td>sat aig tot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td>s</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u</td>
<td>u</td>
<td>u</td>
<td></td>
<td>u</td>
<td></td>
<td></td>
<td>u</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td>s</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>t</td>
<td>t</td>
<td>t</td>
<td></td>
<td>t</td>
<td></td>
<td></td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>32</td>
<td>u</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
<td>NA₂</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td>2</td>
</tr>
<tr>
<td>sp-dt-if</td>
<td>32</td>
<td>u</td>
<td>TO</td>
<td>0</td>
<td>1</td>
<td>0 0 0 0</td>
<td>3</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td>2</td>
</tr>
<tr>
<td>sp-wt-cl</td>
<td>32</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>1</td>
<td>0 1 0 1</td>
<td>6</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 2</td>
<td>1 1 2 4</td>
<td>4</td>
</tr>
<tr>
<td>sp-bd-ks</td>
<td>32</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>0 0 0 1</td>
<td>12</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>sp-ar-ck</td>
<td>32</td>
<td>u</td>
<td>TO</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
<td>5</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>bp-ar-rc</td>
<td>32</td>
<td>u</td>
<td>0</td>
<td>TO</td>
<td>264</td>
<td>0 0 0 0</td>
<td>4</td>
<td>0</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>bp-ct-bk</td>
<td>32</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>0 0 0 0</td>
<td>9</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>bp-os-cu</td>
<td>32</td>
<td>u</td>
<td>0</td>
<td>TO</td>
<td>272</td>
<td>0 0 0 0</td>
<td>7</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>bp-wt-cs</td>
<td>32</td>
<td>u</td>
<td>0</td>
<td>TO</td>
<td>257</td>
<td>0 0 0 0</td>
<td>5</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 2</td>
<td>2</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>64</td>
<td>u</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0 0 2 2</td>
<td>NA₂</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 13 13</td>
<td>0 6 6 19</td>
<td>19</td>
</tr>
<tr>
<td>sp-dt-if</td>
<td>64</td>
<td>u</td>
<td>TO</td>
<td>2</td>
<td>13</td>
<td>0 0 3 3</td>
<td>31</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 17 17</td>
<td>0 8 8 25</td>
<td>25</td>
</tr>
<tr>
<td>sp-wt-cl</td>
<td>64</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>9</td>
<td>0 9 2 11</td>
<td>96</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 14 23</td>
<td>7 6 13 37</td>
<td>37</td>
</tr>
<tr>
<td>sp-bd-ks</td>
<td>64</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>7</td>
<td>0 1 2 3</td>
<td>162</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 14 16</td>
<td>1 6 7 23</td>
<td>23</td>
</tr>
<tr>
<td>sp-ar-ck</td>
<td>64</td>
<td>u</td>
<td>TO</td>
<td>1</td>
<td>5</td>
<td>0 0 2 2</td>
<td>143</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 13 13</td>
<td>0 6 6 19</td>
<td>19</td>
</tr>
<tr>
<td>bp-ar-rc</td>
<td>64</td>
<td>u</td>
<td>2</td>
<td>TO</td>
<td>269</td>
<td>0 0 2 2</td>
<td>53</td>
<td>0</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 15 15</td>
<td>0 5 5 20</td>
<td>20</td>
</tr>
<tr>
<td>bp-ct-bk</td>
<td>64</td>
<td>u</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>0 0 2 3</td>
<td>119</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 16 17</td>
<td>0 5 5 22</td>
<td>22</td>
</tr>
<tr>
<td>bp-os-cu</td>
<td>64</td>
<td>u</td>
<td>3</td>
<td>TO</td>
<td>27</td>
<td>0 0 3 4</td>
<td>95</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 18 18</td>
<td>0 6 7 25</td>
<td>25</td>
</tr>
<tr>
<td>bp-wt-cs</td>
<td>64</td>
<td>u</td>
<td>3</td>
<td>TO</td>
<td>TO</td>
<td>0 0 3 3</td>
<td>75</td>
<td>TO</td>
<td>NA₄</td>
<td>TO</td>
<td>0</td>
<td>0 15 15</td>
<td>0 5 5 20</td>
<td>20</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>32</td>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td>2</td>
</tr>
<tr>
<td>sp-wt-cl</td>
<td>32</td>
<td>s</td>
<td>TO</td>
<td>0</td>
<td>245</td>
<td>0 1 0 1</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 1 1 2</td>
<td>1 1 2 4</td>
<td>4</td>
</tr>
<tr>
<td>btor</td>
<td>32</td>
<td>t</td>
<td>0</td>
<td>NA₃</td>
<td>0</td>
<td>0 0 0 0</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>64</td>
<td>s</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0 0 2 2</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 13 13</td>
<td>0 6 6 19</td>
<td>19</td>
</tr>
<tr>
<td>bp-wt-cl</td>
<td>64</td>
<td>s</td>
<td>TO</td>
<td>47</td>
<td>TO</td>
<td>0 10 3 12</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 10 26</td>
<td>7 5 12 38</td>
<td>38</td>
</tr>
<tr>
<td>btor</td>
<td>64</td>
<td>t</td>
<td>1</td>
<td>NA₃</td>
<td>1</td>
<td>0 0 1 1</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>NA₁</td>
<td>0</td>
<td>0 4 4 4</td>
<td>0 3 4 8</td>
<td>8</td>
</tr>
</tbody>
</table>

NA₁: tool not applicable to type SPEC  NA₂: tool not yet available  NA₃: would lead to incompleteness  NA₄: not reproducible
## Large Multipliers

<table>
<thead>
<tr>
<th>Architecture</th>
<th>( n )</th>
<th>SPEC</th>
<th>Verify</th>
<th>MGD</th>
<th>CSYY</th>
<th>AIG size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAC19</td>
<td>TCAD19</td>
<td></td>
</tr>
<tr>
<td>btor</td>
<td>128</td>
<td>u</td>
<td>0 0 34</td>
<td>NA(_2)</td>
<td>129 920</td>
<td></td>
</tr>
<tr>
<td>kjvnkv</td>
<td>128</td>
<td>u</td>
<td>0 0 19</td>
<td>NA(_2)</td>
<td>194 560</td>
<td></td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>128</td>
<td>u</td>
<td>0 0 19</td>
<td>NA(_2)</td>
<td>194 560</td>
<td></td>
</tr>
<tr>
<td>sp-dt-lf</td>
<td>128</td>
<td>u</td>
<td>0 2 48</td>
<td>NA(_2)</td>
<td>193 550</td>
<td></td>
</tr>
<tr>
<td>sp-wt-bk</td>
<td>128</td>
<td>u</td>
<td>0 1 54</td>
<td>NA(_2)</td>
<td>197 518</td>
<td></td>
</tr>
<tr>
<td>btor</td>
<td>256</td>
<td>u</td>
<td>1 0 536</td>
<td>NA(_2)</td>
<td>521 984</td>
<td></td>
</tr>
<tr>
<td>kjvnkv</td>
<td>256</td>
<td>u</td>
<td>1 0 252</td>
<td>NA(_2)</td>
<td>782 336</td>
<td></td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>256</td>
<td>u</td>
<td>2 0 254</td>
<td>NA(_2)</td>
<td>782 336</td>
<td></td>
</tr>
<tr>
<td>sp-dt-lf</td>
<td>256</td>
<td>u</td>
<td>4 6 901</td>
<td>NA(_2)</td>
<td>780 302</td>
<td></td>
</tr>
<tr>
<td>sp-wt-bk</td>
<td>256</td>
<td>u</td>
<td>3 3 985</td>
<td>NA(_2)</td>
<td>790 098</td>
<td></td>
</tr>
<tr>
<td>btor</td>
<td>512</td>
<td>u</td>
<td>7 0 10 458</td>
<td>NA(_2)</td>
<td>2 092 544</td>
<td></td>
</tr>
<tr>
<td>kjvnkv</td>
<td>512</td>
<td>u</td>
<td>9 0 4 592</td>
<td>NA(_2)</td>
<td>3 137 536</td>
<td></td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>512</td>
<td>u</td>
<td>10 0 4 541</td>
<td>NA(_2)</td>
<td>3 137 536</td>
<td></td>
</tr>
<tr>
<td>sp-dt-lf</td>
<td>512</td>
<td>u</td>
<td>26 21 21 563</td>
<td>NA(_2)</td>
<td>3 133 454</td>
<td></td>
</tr>
<tr>
<td>sp-wt-bk</td>
<td>512</td>
<td>u</td>
<td>25 9 48 197</td>
<td>NA(_2)</td>
<td>3 156 866</td>
<td></td>
</tr>
<tr>
<td>btor</td>
<td>1024</td>
<td>u</td>
<td>96 0 306 157</td>
<td>NA(_2)</td>
<td>8 379 392</td>
<td></td>
</tr>
<tr>
<td>kjvnkv</td>
<td>1024</td>
<td>u</td>
<td>107 0 97 670</td>
<td>NA(_2)</td>
<td>12 566 528</td>
<td></td>
</tr>
</tbody>
</table>

\( \text{NA}_1 \): tool not applicable to type SPEC  \( \text{NA}_2 \): tool not yet available  
\( \text{NA}_3 \): would lead to incompleteness  \( \text{NA}_4 \): not reproducible
Future Work

Circuit Verification

- other word-level operators (shift, division, ...)
- floating point operators (addition, ...)
- synthesized multipliers

Proof Generation

- connection to clausal proof systems
- certified proof checker
- boolean proofs

“really really correct”
SAT, COMPUTER ALGEBRA, MULTIPLIERS

Armin Biere joint work with Daniela Kaufmann and Manuel Kauers

6th Vampire Workshop
Lisbon, Portugal, July 7, 2019