SAT, COMPUTER ALGEBRA, MULTIPLIERS

Armin Biere joint work with Daniela Kaufmann and Manuel Kauers

6th Vampire Workshop

Lisbon, Portugal, July 7, 2019

FWF Der Wissenschaftsfonds.

RiSE Rigorous Systems Engineering

JOHANNES KEPLER UNIVERSITÄT LINZ

Multiplication

$$\begin{array}{ccccc} 1 & 3 & \cdot & 1 & 5 \\ \hline \\ \\ \hline \end{array}$$

Multiplication

$$\begin{array}{rrcrr} 1 & 3 & \cdot & 1 & 5 \\ \hline & & & 6 & 5 \\ & & & & \\ \hline \end{array}$$

Multiplication

$$
\begin{array}{cccccc}
1 & 3 & \cdot & 1 & 5 \\
\hline
 & & & 6 & 5 \\
 & & 1 & 3 & \\
\hline
\end{array}
$$

Multiplication

$$\begin{array}{cccccc}
1 & 3 & \cdot & 1 & 5 \\
\hline
 & & & 6 & 5 \\
 & & 1 & 3_{\color{blue}0} & \\
\hline
 & & & & 5
\end{array}$$

Multiplication

$$
\begin{array}{cccccc}
1 & 3 & \cdot & 1 & 5 \\
\hline
 & & & 6 & 5 \\
 & & 1_{\color{blue}0} & 3_{\color{blue}0} & \\
\hline
 & & & 9 & 5
\end{array}
$$

Multiplication

$$
\begin{array}{ccccc}
1 & 3 & \cdot & 1 & 5 \\
\hline
 & & & 6 & 5 \\
 & {}_{\color{blue}0} & 1 & {}_{\color{blue}0}\,3 & {}_{\color{blue}0} \\
\hline
 & & 1 & 9 & 5
\end{array}
$$

Binary multiplication

$$\begin{array}{cccccccccc} 1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\ \hline \\ \hline \end{array}$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{cccccccccc}
 & & & & 1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & & & & & 1 & 1 & 0 & 1 \\
\\
\hline
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & & & & & & \\
\hline
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & & & & & & & \\
\hline
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
\hline
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & & & & & {\scriptstyle 0} & \\
\hline
 & & & & & & & & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & & & & {\scriptstyle 0} & {\scriptstyle 0} & \\
\hline
 & & & & & & & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & & & & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} \\
\hline
 & & & & & & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & & {}_{2} & {}_{1} & {}_{0} & {}_{0} & \\
\hline
 & & & & & 0 & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} \\
\hline
 & & & & 0 & 0 & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & & & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} \\
\hline
 & & & 0 & 0 & 0 & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} & \\
\hline
 & & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

Binary multiplication

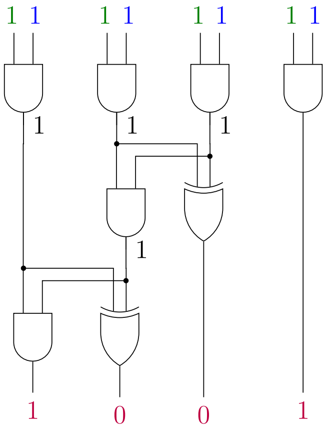
$$
\begin{array}{ccccccccc}
1 & 1 & 0 & 1 & \cdot & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 2} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} & \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}
$$

$$13 \cdot 15 = 195$$

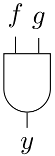
Example: 2 Bit - Binary Multiplication

$$
\begin{array}{cccccc}
 & 1 & 1 & \cdot & 1 & 1 \\
\hline
 & & & & 1 & 1 \\
 & & & 1 & 1 & \\
 & & {}_1 & {}_1 & {}_0 & \\
\hline
 & & 1 & 0 & 0 & 1
\end{array}
$$

$3 \cdot 3 = 9$

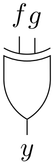


Example: 2 Bit - Binary Multiplication



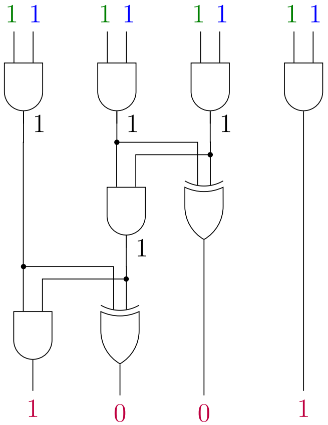
AND-Gate

f	g	y
0	0	0
0	1	0
1	0	0
1	1	1



XOR-Gate

f	g	y
0	0	0
0	1	1
1	0	1
1	1	0

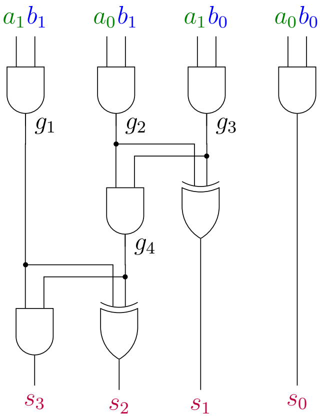


Motivation & Solving Techniques

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

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

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



Solving Techniques

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

Previous Work

- **SAT using CNF encoding**
  - A. Biere. **Weakness** of CDCL solvers. SAT Solving Workshop, 2016.
  - P. Beame and V. Liew. **Towards verifying** nonlinear integer arithmetic. In CAV, 2017.
- **Binary moment diagrams**
  - Y.-A. Chen and R.E. Bryant. Verification of arithmetic circuits with **binary moment diagrams**. In DAC, 1995.
- **Algebraic reasoning**
  - O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G.-M. Greuel. An algebraic approach for proving data correctness in **arithmetic data paths**. In CAV, 2008.
  - 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.
  - A.A.R. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler. Formal verification of integer multipliers by combining **Gröbner basis with logic reduction**. In DATE, 2016.

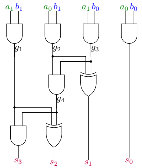
Recent Work

- D. Ritirc, A. Biere, and M. Kauers. Column-wise verification of multipliers using computer algebra. In FMCAD'17.
- D. Ritirc, A. Biere, and M. Kauers. A Practical Polynomial Calculus for Arithmetic Circuit Verification. In 3rd International Workshop on Satisfiability Checking and Symbolic Computation (SC2'18). CEUR-WS, 2018.
- D. Kaufmann, A. Biere, and M. Kauers. Incremental Column-wise verification of arithmetic circuits using computer algebra. FMSD, Feb 2019
- D. Kaufmann, M. Kauers, A. Biere, and D. Cok. Arithmetic Verification Problems Submitted to the SAT Race 2019. In Proc. of SAT Race 2019, 2019.
- D. Kaufmann, A. Biere, and M. Kauers. Verifying Large Multipliers by Combining SAT and Computer Algebra. In FMCAD'19.

- M. Ciesielski, T. Su, A. Yasin, and C. Yu. Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model. IEEE TCAD, 2019.
- A. Mahzoon, D. Große, and R. Drechsler. PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers. In ICCAD'18.
- A. Mahzoon, D. Große, and R. Drechsler. RevSCA: Using Reverse Engineering to Bring Light into Backward Rewriting for Big and Dirty Multipliers. In DAC'19.

Basic Idea of Algebraic Approach

Multiplier



Polynomials

$B = \{$
$x - a_0 * b_0,$
$y - a_1 * b_1,$
$s_0 - x * y,$
$\dots$
$\}$

Specification

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

Ideal Membership Test

$= 0$ ✓

$\neq 0$ ✗

Polynomials

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

- $\mathbb{Q}[X]$ is the **ring of polynomials** with variables $X = x_1, \ldots, x_n$ and coefficients in $\mathbb{Q}$.
- A **term** τ_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

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

- We fix a **term order** such that for all terms τ, σ_1, σ_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 $u_i < v_i$.

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

Ideals

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

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

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

$$I = \{q_1 p_1 + \cdots + q_m p_m \mid q_1, \ldots, q_m \in \mathbb{Q}[X]\} = \langle P \rangle$$

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

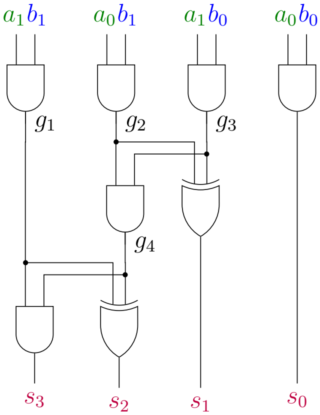
Circuit Polynomials

Gate Polynomials.

$$
\begin{array}{ll}
s_3 = g_1 \wedge g_4 & -s_3 + g_1 g_4, \\
s_2 = g_1 \oplus g_4 & -s_2 + g_1 + g_4 - 2g_1 g_4, \\
g_4 = g_2 \wedge g_3 & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 & -s_1 + g_2 + g_3 - 2g_2 g_3, \\
g_1 = a_1 \wedge b_1 & -g_1 + a_1 b_1, \\
g_2 = a_0 \wedge b_1 & -g_2 + a_0 b_1, \\
g_3 = a_1 \wedge b_0 & -g_3 + a_1 b_0, \\
s_0 = a_0 \wedge b_0 & -s_0 + a_0 b_0
\end{array}
$$

Boolean Value Polynomials.

$$
\begin{array}{ll}
a_1, a_0 \in \mathbb{B} & a_1(1 - a_1),\ a_0(1 - a_0), \\
b_1, b_0 \in \mathbb{B} & b_1(1 - b_1),\ b_0(1 - b_0)
\end{array}
$$



Ideals associated to Circuits

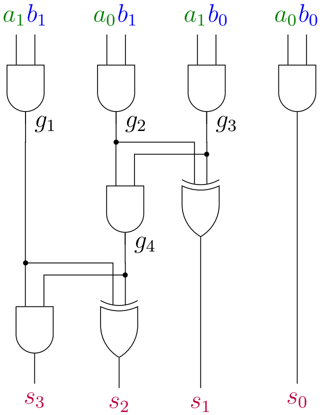
Polynomial Circuit Constraints (PCCs).

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.

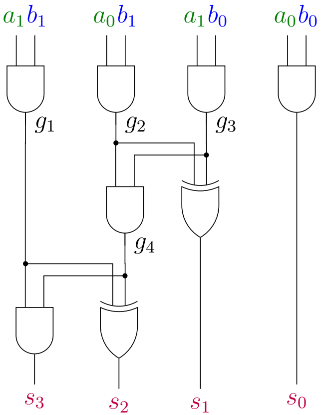
- The set of all PCCs is denoted by $I(C)$.
- $I(C)$ contains all relations of the circuit.
- $I(C)$ is an ideal.



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
- $\dots$



Multiplier. A circuit C is called a multiplier if

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \quad I(C).$$

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

Circuit Gröbner basis

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

- G = Gate Polynomials + 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.

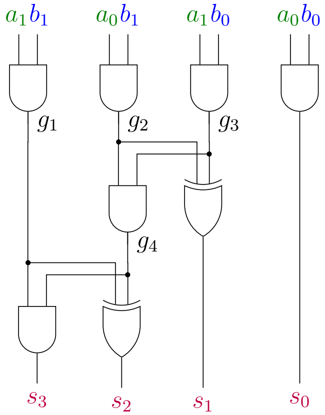
Circuit Polynomials

Gate Polynomials.

$$
\begin{array}{rl}
s_3 = g_1 \wedge g_4 & -s_3 + g_1 g_4, \\
s_2 = g_1 \oplus g_4 & -s_2 - 2g_1 g_4 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 & -s_1 - 2g_2 g_3 + g_2 + g_3, \\
g_1 = a_1 \wedge b_1 & -g_1 + a_1 b_1, \\
g_2 = a_0 \wedge b_1 & -g_2 + a_0 b_1, \\
g_3 = a_1 \wedge b_0 & -g_3 + a_1 b_0, \\
s_0 = a_0 \wedge b_0 & -s_0 + a_0 b_0
\end{array}
$$

Boolean Value Polynomials.

$$
\begin{array}{rl}
a_1, a_0 \in \mathbb{B} & -a_1^2 + a_1,\ -a_0^2 + a_0, \\
b_1, b_0 \in \mathbb{B} & -b_1^2 + b_1,\ -b_0^2 + b_0
\end{array}
$$



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

Non-Incremental Checking Algorithm.

Divide polynomial $\sum_{i=0}^{2n-1} 2^i s_i - (\sum_{i=0}^{n-1} 2^i a_i)(\sum_{i=0}^{n-1} 2^i b_i)$ 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 ± 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

$$
\begin{aligned}
G = \{ & \\
& -s_3 + g_1 g_4, \\
& -s_2 + g_1 + g_4 - 2 g_1 g_4, \\
& -g_4 + g_2 g_3, \\
& -s_1 + g_2 + g_3 - 2 g_2 g_3, \\
& -g_1 + a_1 b_1, \\
& -g_2 + a_0 b_1, \\
& -g_3 + a_1 b_0, \\
& -s_0 + a_0 b_0, \\
& -a_1^2 + a_1, \\
& -a_0^2 + a_0, \\
& -b_1^2 + b_1, \\
& -b_0^2 + b_0 \}
\end{aligned}
$$

$$
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
$$

Example: 2 Bit - Binary Multiplication

$$
\begin{aligned}
G = \{ & \\
& \color{red}{-s_3 + g_1 g_4,} \\
& -s_2 + g_1 + g_4 - 2g_1 g_4, \\
& -g_4 + g_2 g_3, \\
& -s_1 + g_2 + g_3 - 2g_2 g_3, \\
& -g_1 + a_1 b_1, \\
& -g_2 + a_0 b_1, \\
& -g_3 + a_1 b_0, \\
& -s_0 + a_0 b_0, \\
& -a_1^2 + a_1, \\
& -a_0^2 + a_0, \\
& -b_1^2 + b_1, \\
& -b_0^2 + b_0 \}
\end{aligned}
$$

$$8\color{blue}{s_3}\color{black}{} + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0$$

$$8\color{red}{g_1 g_4}\color{black}{} + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0$$

Example: 2 Bit - Binary Multiplication

$$
\begin{aligned}
G = \{ & \\
& -s_3 + g_1 g_4, \\
& -s_2 + g_1 + g_4 - 2g_1 g_4, \\
& -g_4 + g_2 g_3, \\
& -s_1 + g_2 + g_3 - 2g_2 g_3, \\
& -g_1 + a_1 b_1, \\
& -g_2 + a_0 b_1, \\
& -g_3 + a_1 b_0, \\
& -s_0 + a_0 b_0, \\
& -a_1^2 + a_1, \\
& -a_0^2 + a_0, \\
& -b_1^2 + b_1, \\
& -b_0^2 + b_0 \}
\end{aligned}
$$

$$8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0$$

$$8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0$$

$$
\begin{aligned}
8g_1 g_4 + 4(g_1 + g_4 - 2g_1 g_4) + 2s_1 + s_0 \\
- 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
\end{aligned}
$$

Example: 2 Bit - Binary Multiplication

$$
\begin{aligned}
G = \{ & \\
& -s_3 + g_1 g_4, \\
& -s_2 + g_1 + g_4 - 2g_1 g_4, \\
& -g_4 + g_2 g_3, \\
& -s_1 + g_2 + g_3 - 2g_2 g_3, \\
& -g_1 + a_1 b_1, \\
& -g_2 + a_0 b_1, \\
& -g_3 + a_1 b_0, \\
& -s_0 + a_0 b_0, \\
& -a_1^2 + a_1, \\
& -a_0^2 + a_0, \\
& -b_1^2 + b_1, \\
& -b_0^2 + b_0 \}
\end{aligned}
$$

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4(g_1 + g_4 - 2g_1 g_4) + 2s_1 + s_0 \\
- 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
\vdots \\
{\color{red}0}
\end{aligned}
$$

Computational Issues

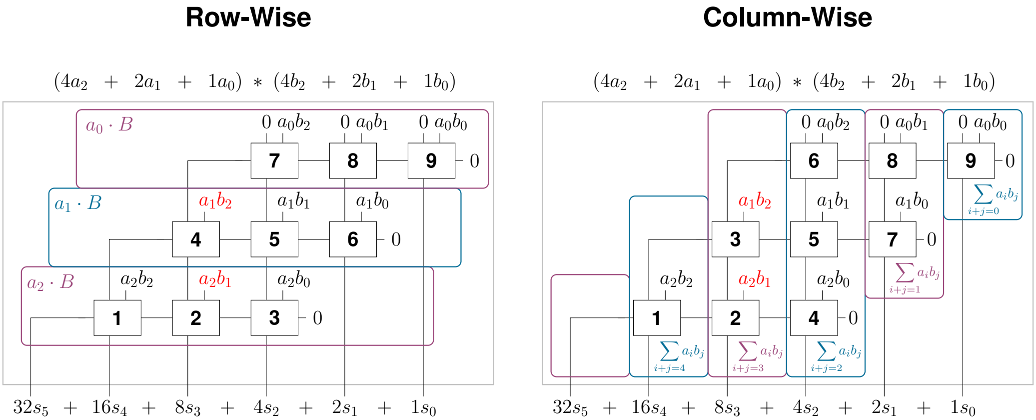
Generally the number of monomials in the intermediate results increases drastically:

- 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
Column-Wise
$(4a_2 + 2a_1 + 1a_0) * (4b_2 + 2b_1 + 1b_0)$
$a_0 \cdot B$
$a_1 \cdot B$
$a_2 \cdot B$
$\sum_{i+j=0} a_i b_j$
$\sum_{i+j=1} a_i b_j$
$\sum_{i+j=2} a_i b_j$
$\sum_{i+j=3} a_i b_j$
$\sum_{i+j=4} a_i b_j$
$32s_5 + 16s_4 + 8s_3 + 4s_2 + 2s_1 + 1s_0$

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:

$$S_0 = I_0 \qquad S_{i+1} = I_{i+1} \setminus \bigcup_{j=0}^{i} S_j$$

Sliced Gröbner Bases. Let G_i be the set of gate and boolean value polynomials in S_i.

Carry Recurrence Relation

Carry Recurrence Relation.
A sequence of $2n+1$ polynomials $C_0, \dots, C_{2n}$ 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 $C_0, \dots, C_{2n}$.

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 Algorithm

Incremental Checking Algorithm.

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

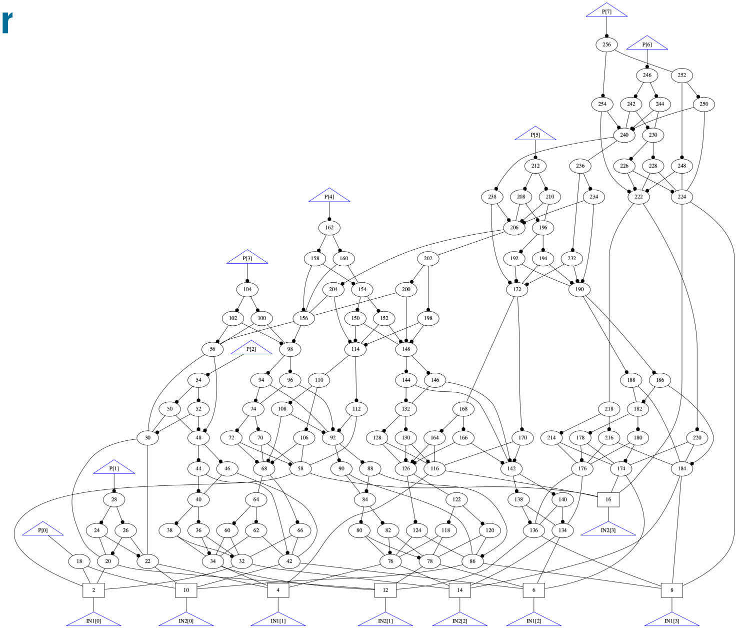
$C_{2n} \leftarrow 0$

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

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

return $C_0 = 0$

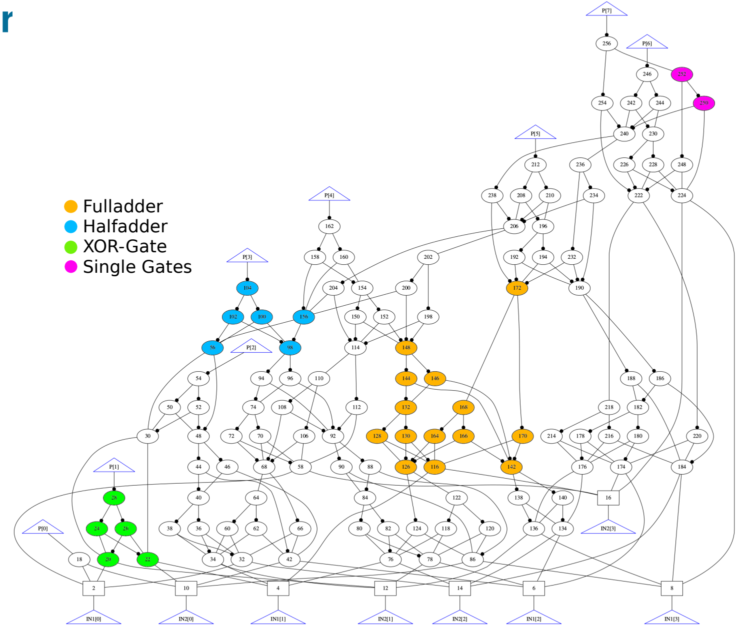
Multiplier



P[7]
256
P[6]
246
252
254
242
244
250
P[5]
240
230
212
236
226
228
248
P[4]
238
208
210
234
222
224
162
206
196
P[3]
158
160
202
192
194
232
104
204
154
200
172
190
102
100
156
150
152
198
56
P[2]
98
114
148
54
94
96
110
144
146
188
186
50
52
74
108
112
132
168
218
182
30
48
72
70
106
92
128
130
164
166
170
214
178
216
180
220
P[1]
44
46
68
58
90
88
126
116
142
176
174
184
28
40
64
84
122
138
140
16
P[0]
24
26
38
36
60
62
66
80
82
124
118
120
136
134
IN2[3]
18
20
22
34
32
42
76
78
86
2
10
4
12
14
6
8
IN1[0]
IN2[0]
IN1[1]
IN2[1]
IN2[2]
IN1[2]
IN1[3]

Multiplier

- 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 $X = Y \mathbin{\dot\cup} Z$ and we want to eliminate Z. Order the terms such that for all terms σ, τ where a variable from Z is contained in σ but not in τ, 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.

$$
\begin{tikzcd}
 & G \arrow[dl] \arrow[drr] & & & \\
G_A \arrow[ddddr] & & & G_B \arrow[d] & \\
 & & & H_B \arrow[dl] \arrow[dr] & \\
 & & H_Y \arrow[dl] & & H_Z \\
 & H & & &
\end{tikzcd}
$$

Step 1: original Gröbner basis G

Step 2: split G into two subbases

Step 3: change order of $<_G$ to $<_H$

Step 4: eliminate the variables of Z

Step 5: rejoin bases $H = G_A \cup H_Y$

Elimination procedure

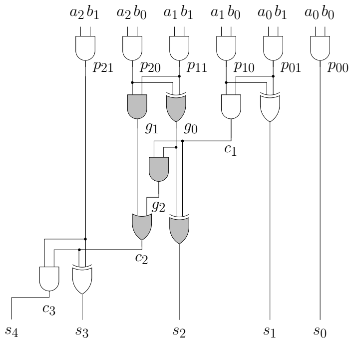
Theorem

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



$$
\begin{aligned}
G_A &= G \backslash G_B \\
G_B &= \{ -g_0 + p_{20} + p_{11} - 2p_{20}p_{11}, \quad -g_1 + p_{20}p_{11}, \quad -g_2 + c_1 g_0, \\
&\qquad -s_2 + c_1 + g_0 - 2c_1 g_0, \quad -c_2 + g_1 + g_2 - g_1 g_2 \}
\end{aligned}
$$

Original lexicographic term ordering $<_G$:

$$
\begin{aligned}
&b_0 < b_1 < a_0 < a_1 < a_2 < p_{00} < s_0 < p_{01} < p_{10} < s_1 < c_1 < \\
&\qquad p_{11} < p_{20} < g_0 < g_1 < g_2 < s_2 < c_2 < p_{21} < s_3 < c_3 < s_4
\end{aligned}
$$

Elimination order $<_H$:

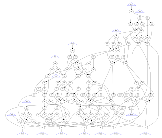
$$
\begin{aligned}
&b_0 < b_1 < a_0 < a_1 < a_2 < p_{00} < s_0 < p_{01} < p_{10} < s_1 < c_1 < \\
&\qquad p_{11} < p_{20} < s_2 < c_2 < p_{21} < s_3 < c_3 < s_4 < g_0 < g_1 < g_2
\end{aligned}
$$

Gröbner basis H_B w.r.t. elimination order $<_H$:

$$
\begin{aligned}
H_B = \{ & g_0 + 2p_{20}p_{11} - p_{20} - p_{11}, \quad g_1 - p_{20}p_{11}, \\
& g_2 + 2p_{20}p_{11}c_1 - p_{20}c_1 - p_{11}c_1, \\
& s_2 - 4p_{20}p_{11}c_1 + 2p_{20}p_{11} + 2p_{20}c_1 - p_{20} + 2p_{11}c_1 - p_{11} - c_1, \\
& 2c_2 + s_2 - p_{20} - p_{11} - c_1 \}
\end{aligned}
$$

Experiments

Multiplier



AIG

AigMulToPoly

Polynomials

$$
\begin{aligned}
B = \{ & \\
& x - a_0 * b_0, \\
& y - a_1 * b_1, \\
& s_0 - x * y, \\
& \ldots \\
\} &
\end{aligned}
$$

CAS-File

Mathematica

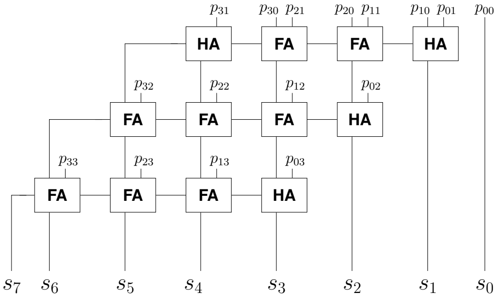
Singular

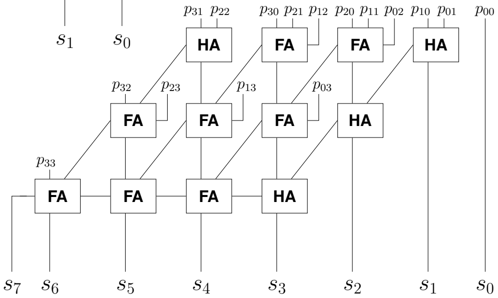
Ideal Membership

$C_0 = 0$ ✓

$C_0 \neq 0$ ✗

Experiments



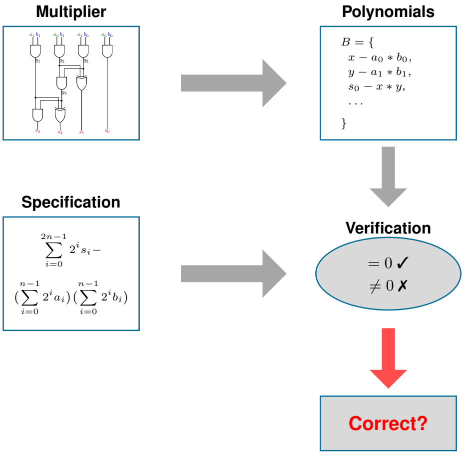


Experiments

mult	n	Mathematica				Singular			
		non-inc	incremental			non-inc	incremental		
				+xor	+xor +add			+xor	+xor +add
btor	16	3	5	2	1	1	1	1	1
btor	32	56	31	14	2	42	28	10	1
btor	64	MO	292	131	11	MO	MO	MO	14
btor	128	TO	TO	TO	101	EE	EE	EE	EE
sp-ar-rc	16	9	7	4	1	TO	6	1	0
sp-ar-rc	32	326	171	30	2	TO	242	28	2
sp-ar-rc	64	MO	TO	300	11	MO	EE	MO	16
sp-ar-rc	128	TO	TO	TO	102	EE	EE	EE	EE

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

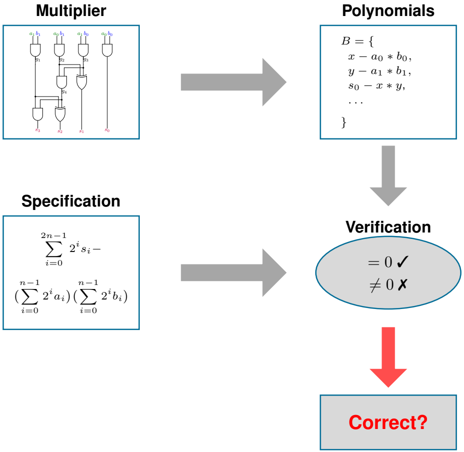
Proofs



Problem:

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

Proofs



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

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

Addition	$\dfrac{p_i \qquad p_j}{p_i + p_j}$	$\forall\, p_i, p_j \in \langle G \rangle : p_i + p_j \in \langle G \rangle$
Multiplication	$\dfrac{p_i}{qp_i}$	$\forall\, q \in \mathbb{Q}[X]\ \forall\, p_i \in \langle G \rangle : qp_i \in \langle G \rangle$

Proofs

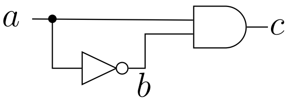
Polynomial calculus: Sequence $P = (p_1, \ldots p_n)$, where each p_k is obtained by:

Addition	$\dfrac{p_i \qquad p_j}{p_i + p_j}$	$\forall\, p_i, p_j \in \langle G \rangle : p_i + p_j \in \langle G \rangle$
Multiplication	$\dfrac{p_i}{qp_i}$	$\forall\, q \in \mathbb{Q}[X]\ \forall\, p_i \in \langle G \rangle : qp_i \in \langle G \rangle$

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



$$
\begin{aligned}
G &= \{ & -b + 1 - a, \quad & b = \neg a \\
& & -c + ab, \quad & c = a \wedge b = a \wedge \neg a \\
& & a^2 - a\} \quad & a \in \mathbb{B} \\
f &= & c & \\
\\
\mathrm{red}_G(f) &= & 0 &
\end{aligned}
$$

$$
\begin{array}{lll}
* & -b+1-a & \\
\hline
+ & -ab + a - a^2 & a^2 - a \\
\hline
& + \quad -ab & -c + ab \\
\hline
& * \quad -c & \\
\hline
& \quad\;\; c &
\end{array}
$$

$$P = (-ab + a - a^2,\ -ab,\ -c,\ c)$$

We translate the polynomial calculus into a concrete **proof** format:

- **given** gate and boolean value polynomials serve as axioms
- instances of addition or multiplication **rule** derive polynomials

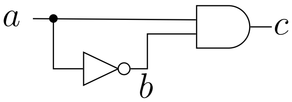
Practical Algebraic Calculus (PAC)

allows automated proof checking

PAC Syntax

letter	::=	‘a’ | ‘b’ | ... | ‘z’ | ‘A’ | ‘B’ | ... | ‘Z’
number	::=	‘0’ | ‘1’ | ... | ‘9’
constant	::=	(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



$$G = \{ \; -b + 1 - a, \; -c + ab, \; a^2 - a\}$$

$$f = c$$

```
* : -b+1-a,        a,             -a*b+a-a^2;
+ : -a*b+a-a^2,    a^2-a,         -a*b;
+ : -a*b,          -c+a*b,        -c;
* : -c,            -1,            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"

$P_0 \leftarrow G$

for $i \leftarrow 1 \dots k$

  let $r_i = (o_i, v_i, w_i, p_i)$

  case $o_i = +$

    if $v_i \in P_{i-1} \ \wedge \ w_i \in P_{i-1} \ \wedge \ p_i = v_i + w_i$ **then** $P_i \leftarrow$ append(P_{i-1}, p_i)

    else return "incorrect"

  case $o_i = *$

    if $v_i \in P_{i-1} \ \wedge \ p_i = v_i * w_i$ **then** $P_i \leftarrow$ append(P_{i-1}, p_i)

    else return "incorrect"

for $i \leftarrow 1 \dots k$

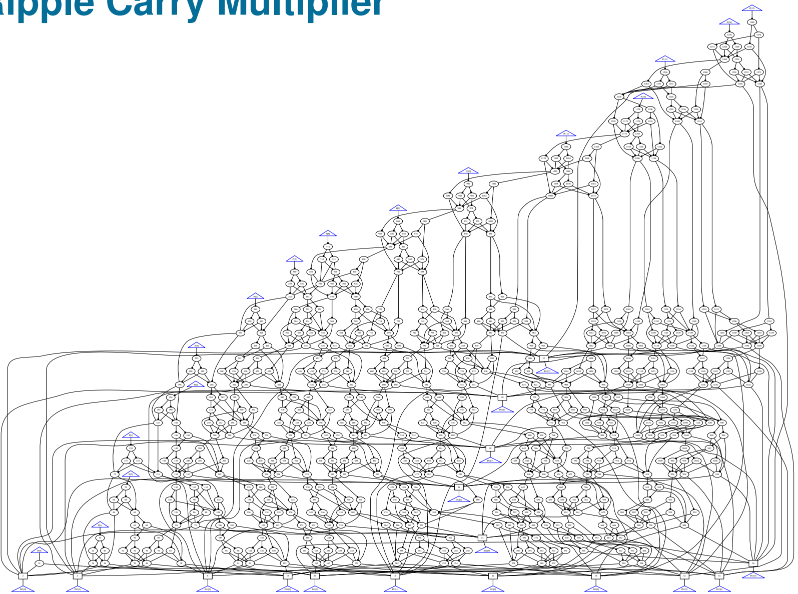
  if p_i is a non zero constant polynomial **then return** "correct-refutation"

return "correct-proof"

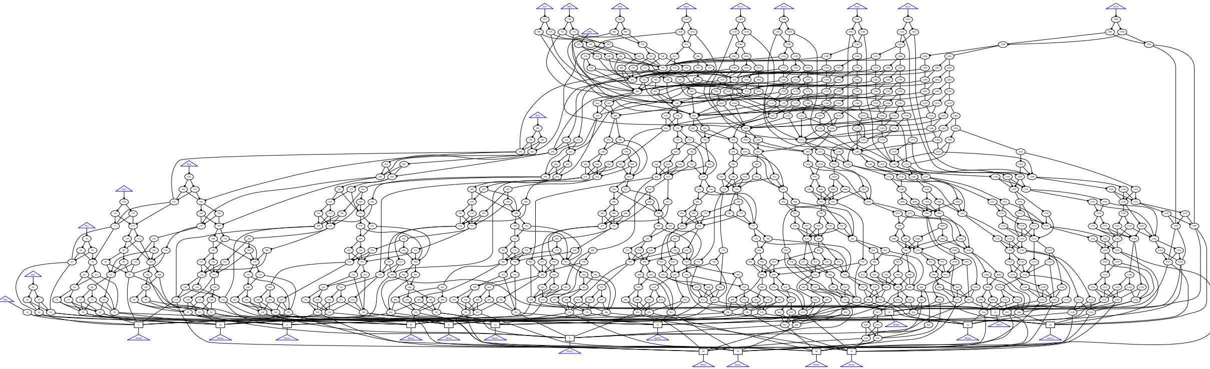
Verifying Correctness of FMCAD'17 and DATE'18 Results

n	mult	option	verification	proof generation	PacTrim	length
16	btor	incrm+elim	1	37	0	12738
32	btor	incrm+elim	2	801	1	53122
64	btor	incrm+elim	11	22378	4	216834
16	sparrc	incrm+elim	1	112	20	17804
32	**sparrc**	**incrm+elim**	**2**	**2611**	**2**	**75244**
64	sparrc	incrm+elim	11	80906	12	309164

Array Ripple Carry Multiplier



Wallace-Tree Carry-Lookahead Multiplier



Checking Commutativity of Multiplication with SAT

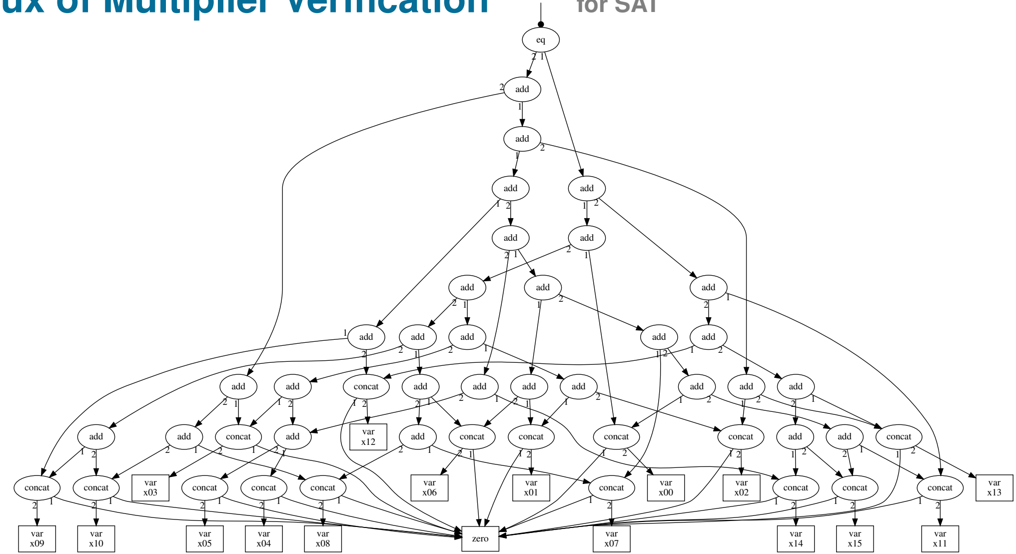
bits	1 core Glucose	1 core Lingeling	12 core cube-and-conquer March|iLingeling	12 core Treengeling
01	0.00	0.00	0.00	0.01
02	0.00	0.00	0.00	0.01
03	0.00	0.00	0.00	0.01
04	0.00	0.00	0.02	0.03
05	0.00	0.01	0.05	0.13
06	0.02	0.03	0.36	0.31
07	0.14	0.27	0.63	0.72
08	1.18	1.98	1.38	2.47
09	7.85	10.98	2.63	4.65
10	37.16	41.49	5.02	10.86
11	147.62	214.98	15.72	21.96
12	833.62	649.49	56.57	61.48
13	------	------	238.10	263.44

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 12))
(declare-fun y () (_ BitVec 12))
(assert (distinct (bvmul x y) (bvmul y x)))
(check-sat)
```

limit of 900 seconds wall clock time

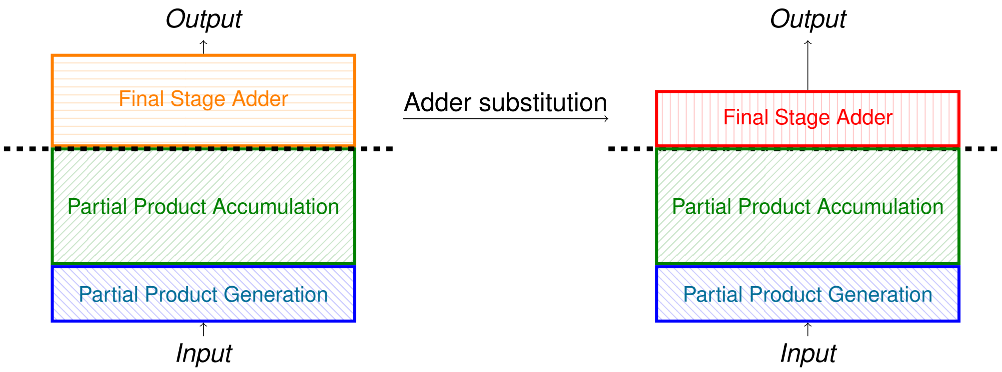
Crux of Multiplier Verification

for SAT

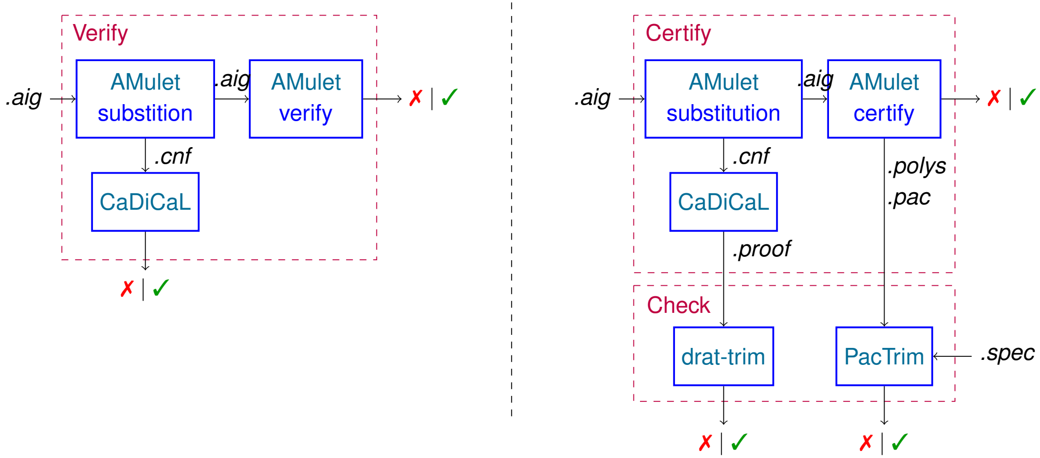


Adder Substitution

[KaufmannBiereKauers-FMCAD'19]



Verification and Certification Flow



Verification and Certification Time

architecture	n	SPEC	nosat	nomod	noelim	Verify				MGD	YCM	CSYY	RBK	Certify				Check			total	proof size	
						sub	sat	aig	tot	DAC19	TCAD17	TCAD19	FMCAD17	sub	sat	aig	tot	sat	aig	tot		sat (drat)	aig (pac)
sp-ar-rc	32	u	0	0	0	0	0	0	0	NA$_2$	NA$_4$	0	8	0	0	1	1	0	1	1	2	0	46 018
sp-dt-lf	32	u	TO	0	1	0	0	0	0	3	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	2 614	44 986
sp-wt-cl	32	u	TO	TO	1	0	1	0	1	6	TO	NA$_4$	TO	0	1	1	2	1	1	2	4	52 390	47 017
sp-bd-ks	32	u	TO	TO	TO	0	0	0	1	12	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	21 188	46 971
sp-ar-ck	32	u	TO	0	0	0	0	0	0	5	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	663	45 514
bp-ar-rc	32	u	0	TO	264	0	0	0	0	4	0	NA$_4$	TO	0	0	1	1	0	1	1	2	0	41 935
bp-ct-bk	32	u	TO	TO	210	0	0	0	0	9	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	1 832	35 803
bp-os-cu	32	u	0	TO	272	0	0	0	0	7	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	0	44 528
bp-wt-cs	32	u	0	TO	257	0	0	0	0	5	TO	NA$_4$	TO	0	0	1	1	0	1	1	2	0	42 963
sp-ar-rc	64	u	2	1	5	0	0	2	2	NA$_2$	NA$_4$	0	133	0	0	13	13	0	6	6	19	0	188 290
sp-dt-lf	64	u	TO	2	13	0	0	3	3	31	TO	NA$_4$	TO	0	0	17	17	0	8	8	25	34 423	186 170
sp-wt-cl	64	u	TO	TO	9	0	9	2	11	96	TO	NA$_4$	TO	0	9	14	23	7	6	13	37	264 471	191 621
sp-bd-ks	64	u	TO	TO	7	0	1	2	3	162	TO	NA$_4$	TO	0	2	14	16	1	6	7	23	78 567	190 911
sp-ar-ck	64	u	TO	1	5	0	0	2	2	143	TO	NA$_4$	TO	0	0	13	13	0	6	6	19	1 432	187 251
bp-ar-rc	64	u	2	TO	TO	0	0	2	2	53	0	NA$_4$	TO	0	0	15	15	0	5	5	20	0	161 935
bp-ct-bk	64	u	TO	TO	TO	0	0	2	3	119	TO	NA$_4$	TO	0	0	16	17	0	5	5	22	27 552	138 179
bp-os-cu	64	u	3	TO	TO	0	0	3	4	95	TO	NA$_4$	TO	0	0	18	18	0	6	7	25	0	166 963
bp-wt-cs	64	u	3	TO	TO	0	0	3	3	75	TO	NA$_4$	TO	0	0	15	15	0	5	5	20	0	161 745
sp-ar-rc	32	s	0	0	0	0	0	0	0	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	0	1	1	0	1	1	2	0	46 090
bp-wt-cl	32	s	TO	0	245	0	1	0	1	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	1	1	2	1	1	2	4	50 620	38 097
btor	32	t	0	NA$_3$	0	0	0	0	0	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	0	0	0	0	0	1	1	0	16 633
sp-ar-rc	64	s	2	1	5	0	0	2	2	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	0	13	13	0	6	6	19	0	188 426
bp-wt-cl	64	s	TO	47	TO	0	10	3	12	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	10	17	26	7	5	12	38	261 650	151 355
btor	64	t	1	NA$_3$	1	0	0	1	1	NA$_1$	NA$_1$	NA$_1$	NA$_1$	0	0	4	4	0	3	4	8	0	69 068

NA$_1$: tool not applicable to type SPEC   NA$_2$: tool not yet available   NA$_3$: would lead to incompleteness   NA$_4$: not reproducible

Large Multipliers

architecture	n	SPEC	Verify				MGD	CSYY	AIG size
			sub	sat	aig	tot	DAC19	TCAD19	
btor	128	u	0	0	34	34	NA_2	0	129 920
kjvnkv	128	u	0	0	19	19	NA_2	NA_4	194 560
sp-ar-rc	128	u	0	0	19	19	349	2	194 560
sp-dt-lf	128	u	0	2	48	50	490	NA_4	193 550
sp-wt-bk	128	u	0	1	54	56	746	NA_4	197 518
btor	256	u	1	0	536	537	NA_2	NA_4	521 984
kjvnkv	256	u	1	0	252	253	NA_2	NA_4	782 336
sp-ar-rc	256	u	2	0	254	255	8 720	NA_4	782 336
sp-dt-lf	256	u	4	6	901	911	12 874	NA_4	780 302
sp-wt-bk	256	u	3	3	985	992	21 454	NA_4	790 098
btor	512	u	7	0	10 458	10 465	NA_2	NA_4	2 092 544
kjvnkv	512	u	9	0	4 592	4 601	NA_2	NA_4	3 137 536
sp-ar-rc	512	u	10	0	4 541	4 551	192 640	NA_4	3 137 536
sp-dt-lf	512	u	26	21	21 563	21 609	240 051	NA_4	3 133 454
sp-wt-bk	512	u	25	9	48 197	48 231	492 320	NA_4	3 156 866
btor	1024	u	96	0	306 157	306 253	NA_2	NA_4	8 379 392
kjvnkv	1024	u	107	0	97 670	97 777	NA_2	NA_4	12 566 528

NA_1: tool not applicable to type SPEC NA_2: tool not yet available
NA_3: would lead to incompleteness 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 *"really really correct"*
- boolean proofs

SAT, COMPUTER ALGEBRA, MULTIPLIERS

Armin Biere joint work with Daniela Kaufmann and Manuel Kauers

6^{th} Vampire Workshop

Lisbon, Portugal, July 7, 2019

Der Wissenschaftsfonds.

Rigorous Systems Engineering

JOHANNES KEPLER
UNIVERSITÄT LINZ