COMBINING SAT AND COMPUTER ALGEBRA FOR CIRCUIT VERIFICATION

Daniela Kaufmann

Johannes Kepler University

Linz, Austria

Beyond Satisfiability Workshop

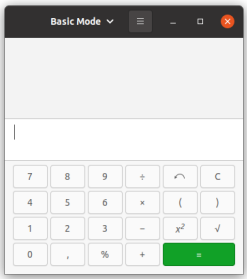
Simons Institute for the Theory of Computing (online)

February 16, 2021

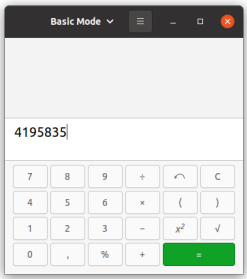
JOHANNES KEPLER
UNIVERSITY LINZ

$$4\,195\,835 \div 3\,145\,727 = ?$$

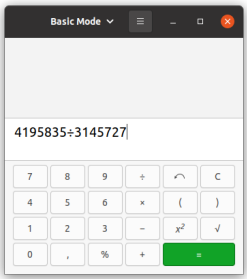
$$4\,195\,835 \div 3\,145\,727 = ?$$



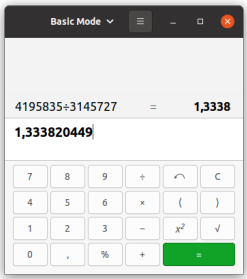
$$4\,195\,835 \div 3\,145\,727 = ?$$



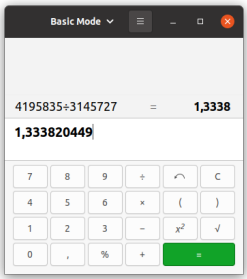
$$4\,195\,835 \div 3\,145\,727 = ?$$

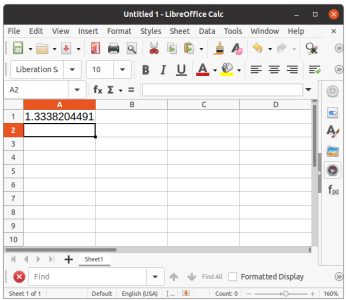


$$4\,195\,835 \div 3\,145\,727 = ?$$

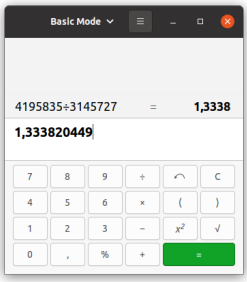


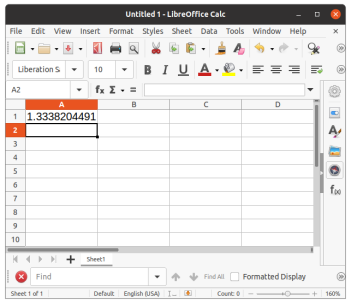
$$4\,195\,835 \div 3\,145\,727 = ?$$

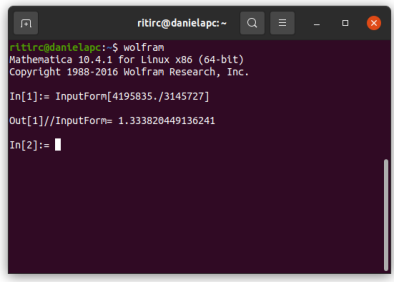


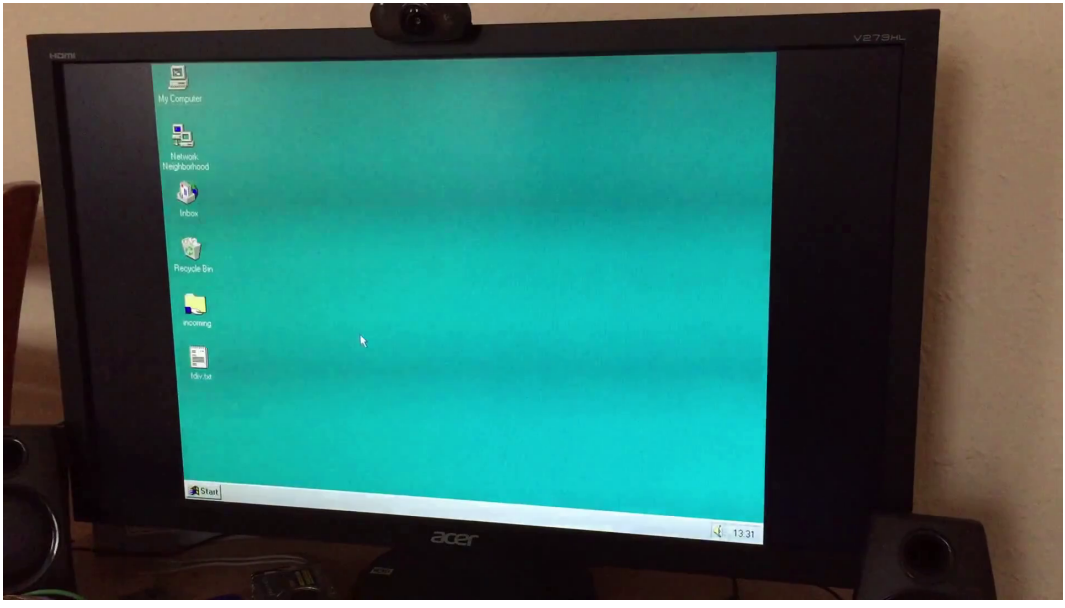


$$4\,195\,835 \div 3\,145\,727 = ?$$

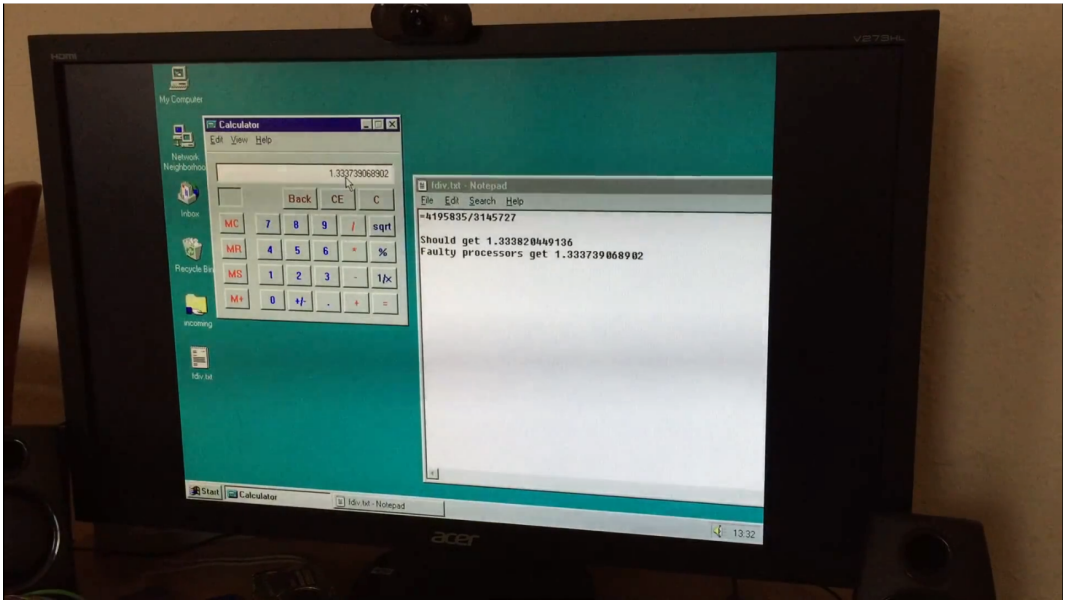








Pentium FDIV bug and AMI GUI BIOS demo `https://youtu.be/hE7qMJVll5U`



Pentium FDIV bug and AMI GUI BIOS demo `https://youtu.be/hE7qMJVl15U`

Intel Pentium Processor

http://neology.com.au/portfolios/a80502-90-sx923/

CPU-World

News • CPUs / Chips • Benchmarks • Information • Forum • Links • About

Identification • Pinouts • S-Spec numbers • Glossary

Unable to connect

SX923 (Intel Pentium 90 MHz)

All S-Specs » Pentium » SX923

SX923 specifications

General information

Type	CPU / Microprocessor
Family	Intel Pentium
Part number	A80502-90
Frequency (MHz)	**90**
Bus speed (MHz)	**60**
Package type	SPGA

Architecture / Microarchitecture / Other

CPUID	0522h
Core stepping	B3
Core voltage (V)	3.3 (3.135 - 3.465)

Notes on sSpec SX923

- This processor has FDIV bug.

https://www.cpu-world.com/sspec/SX/SX923.html

Intel Pentium FDIV bug 1994

http://neology.com.au/portfolios/a80502-90-sx923/

- Affected floating point unit (FPU) in early Intel processors.
- Processor might return incorrect result for division.
- Cost in 1994: 500 million dollars.

Even more than 25 years later, verification of arithmetic circuits is considered to be hard. Especially proving the correctness of gate-level integer multiplier circuits is a challenge.

Multiplication

$$\begin{array}{r} 1\ 1 \cdot 1\ 1 \\ \hline \end{array}$$

$3 \cdot 3 = 9$

Multiplication

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

$3 \cdot 3 = 9$

Multiplication

$$
\begin{array}{cccc}
1 & 1 & \cdot & 1 \quad 1 \\
\hline
 & & & 1 \quad 1 \\
 & & 1 & 1 \quad \\
\hline
\end{array}
$$

$3 \cdot 3 = 9$

Multiplication

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

$3 \cdot 3 = 9$

Multiplication

$$
\begin{array}{cccc}
1 & 1 & \cdot & 1 \quad 1 \\
\hline
 & & & 1 \quad 1 \\
 & & 1_{1} & 1_{0} \\
\hline
 & & & 0 \quad 1
\end{array}
$$

$3 \cdot 3 = 9$

Multiplication

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

$3 \cdot 3 = 9$

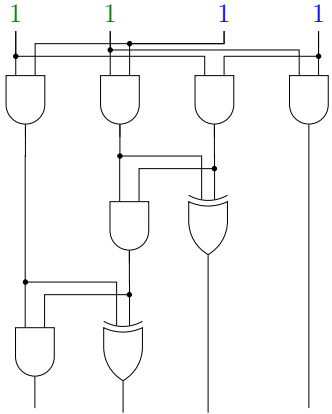
Multiplication

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

$3 \cdot 3 = 9$

Multiplication

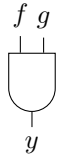
$$
\begin{array}{cccc}
1 & 1 & \cdot & 1 & 1 \\
\hline
 & & & 1 & 1 \\
 & & 1 & 1 & \\
 & 1 & 1 & 0 & \\
\hline
 & 1 & 0 & 0 & 1
\end{array}
$$



$3 \cdot 3 = 9$

Multiplication

AND-Gate



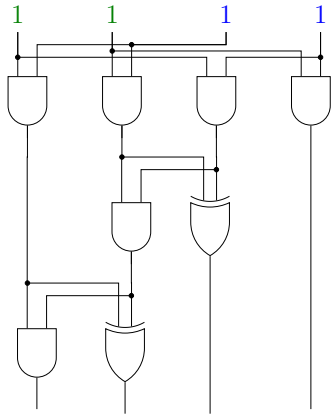
$f \wedge g = y$

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

XOR-Gate

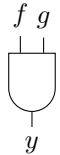
$f \oplus g = y$

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



Multiplication

AND-Gate



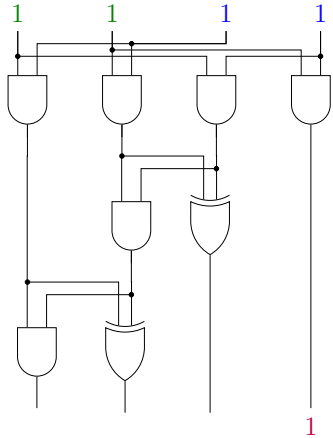
$f \wedge g = y$

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

XOR-Gate

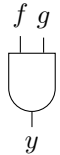
$f \oplus g = y$

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



Multiplication

AND-Gate



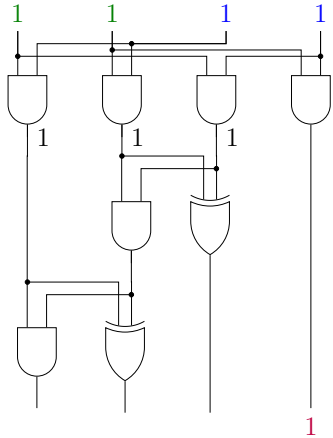
$f \wedge g = y$

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

XOR-Gate

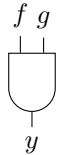
$f \oplus g = y$

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



Multiplication

AND-Gate



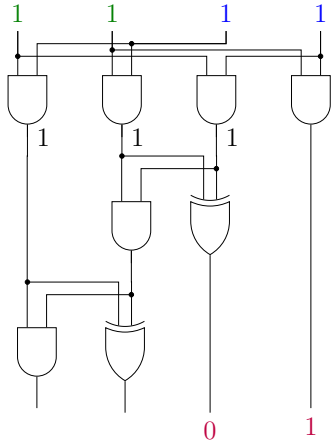
$f \wedge g = y$

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

XOR-Gate

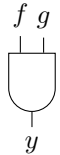
$f \oplus g = y$

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



Multiplication

AND-Gate



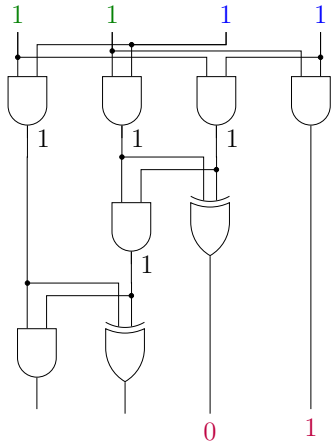
$f \wedge g = y$

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

XOR-Gate

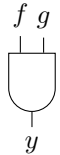
$f \oplus g = y$

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



Multiplication

AND-Gate



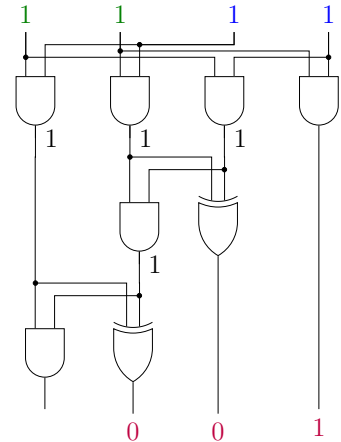
$f \wedge g = y$

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

XOR-Gate

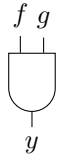
$f \oplus g = y$

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



Multiplication

AND-Gate



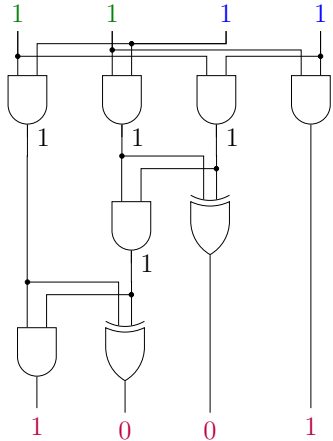
$f \wedge g = y$

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

XOR-Gate

$f \oplus g = y$

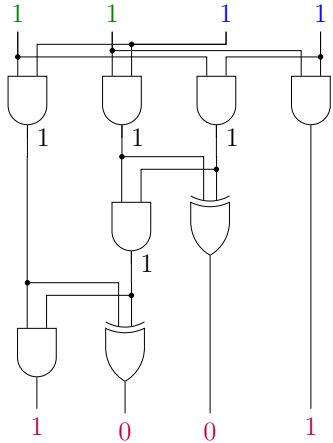
f	g	y
0	0	0
0	1	1
1	0	1
1	1	0



Is the circuit correct?

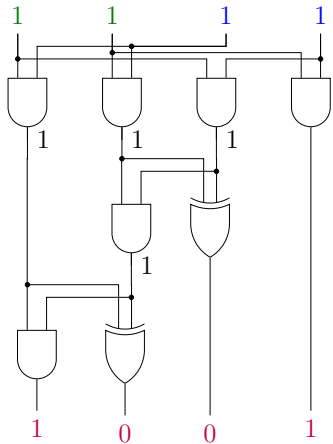
Multiplier circuit

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



Multiplier circuit

Circuit seems correct!



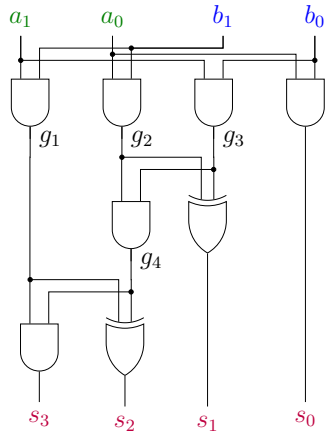
Is the circuit really correct?

Multiplier Circuits

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

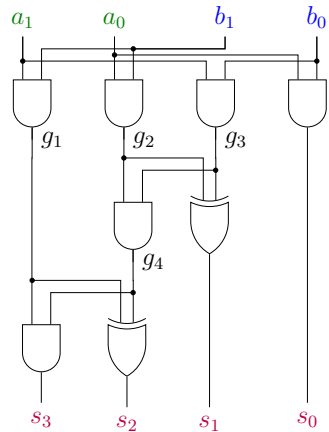
Question: For all possible $a_i, b_i \in \mathbb{B}$:

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



Testing

a_1a_0	·	b_1b_0
00	·	00
00	·	01
00	·	10
00	·	11
01	·	00
01	·	01
01	·	10
01	·	11
10	·	00
10	·	01
10	·	10
10	·	11
11	·	00
11	·	01
11	·	10
11	·	11

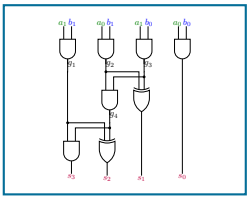


Testing

Bit-width	Cases	
2	16	
3	64	
4	256	
5	1 024	
6	4 096	
7	16 384	
8	65 536	
⋮	⋮	
32	18 446 744 073 709 551 616	quintillion
⋮	⋮	
64	340 282 366 920 938 463 463 374 607 431 768 211 456	undecillion

Formal Verification

System



Specification

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

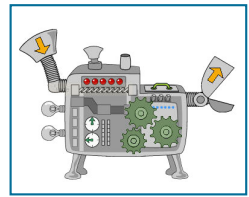
$(2a_1 + a_0) * (2b_1 + b_0) =$

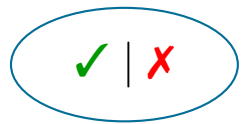
$8s_3 + 4s_2 + 2s_1 + s_0$?

Mathematical Model

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

Automated Decision Process





Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition

  [Biere SATComp'16]

- Exponential run-time of solvers

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
  [Biere SATComp'16]
- Exponential run-time of solvers

Decision Diagrams

- First technique to detect Pentium bug
  [ChenBryant DAC'95]
- Cannot be applied fully automatically

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
  [Biere SATComp'16]
- Exponential run-time of solvers

Decision Diagrams

- First technique to detect Pentium bug
  [ChenBryant DAC'95]
- Cannot be applied fully automatically

Theorem Proving

- Used in industry
- Past: Requires manual effort
- 2020: Progress in ACL2
  [TemelSlobodovaHunt CAV'20]

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
  [Biere SATComp'16]
- Exponential run-time of solvers

Decision Diagrams

- First technique to detect Pentium bug
  [ChenBryant DAC'95]
- Cannot be applied fully automatically

Theorem Proving

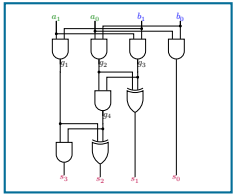
- Used in industry
- Past: Requires manual effort
- 2020: Progress in ACL2
  [TemelSlobodovaHunt CAV'20]

Algebraic Approach

- Seminal work:
  [CiesielskiYuBrownLiuRossi DAC'15]
  [SayedGroßeKühneSoekenDrechsler DATE'16]
- Polynomial encoding
- Works for non-trivial multiplier designs

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

Specification

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

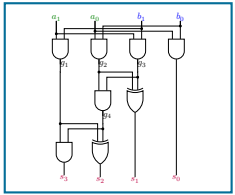
Implication

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

Specification

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

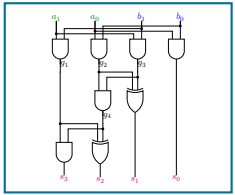
Roots

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

Specification

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

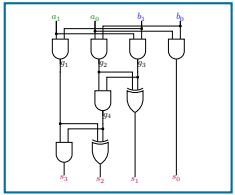
Ideal Membership

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

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

$= 0$ ✓

$\neq 0$ ✗

Multiplier Specification

Unsigned Integers:

$$0 = \sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \mathbb{Z}[X]$$

Multiplier Specification

Unsigned Integers:

$$0 = \sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \mathbb{Z}[X]$$

Signed Integers:

$$0 = -2^{2n-1} s_{2n-1} + \sum_{i=0}^{2n-2} 2^i s_i - \left(-2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i\right)\left(-2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i\right) \quad \in \mathbb{Z}[X]$$

Multiplier Specification

Unsigned Integers:

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \mathbb{Z}[X]$$

Signed Integers:

$$-2^{2n-1}s_{2n-1} + \sum_{i=0}^{2n-2} 2^i s_i - \left(-2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i\right)\left(-2^{n-1}b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i\right) \quad \in \mathbb{Z}[X]$$

Multiplier Specification

Unsigned Integers:

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \mathbb{Z}[X] \subseteq \mathbb{Q}[X]$$

Signed Integers:

$$-2^{2n-1} s_{2n-1} + \sum_{i=0}^{2n-2} 2^i s_i - \left(-2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i\right)\left(-2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i\right) \quad \in \mathbb{Z}[X] \subseteq \mathbb{Q}[X]$$

Multiplier Specification

Unsigned Integers:

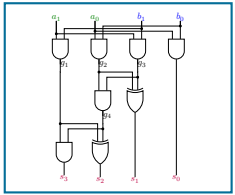
$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \in \mathbb{Q}[X]$$

Signed Integers:

$$-2^{2n-1}s_{2n-1} + \sum_{i=0}^{2n-2} 2^i s_i - \left(-2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i\right)\left(-2^{n-1}b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i\right) \quad \in \mathbb{Q}[X]$$

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

Specification

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

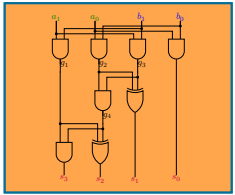
Ideal Membership

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

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

$= 0$ ✓

$\neq 0$ ✗

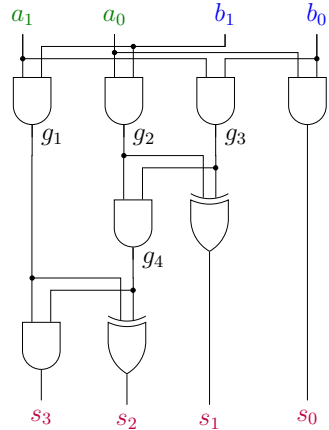
From Circuits to Polynomials

AND-Gate $f \wedge g = y$

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

XOR-Gate $f \oplus g = y$

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



From Circuits to Polynomials

AND-Gate $f \wedge g = y$

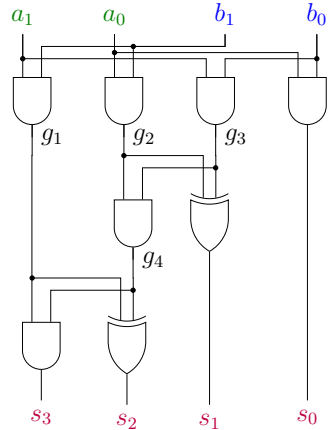
f	g	y
0	0	0
0	1	0
1	0	0
1	1	1

$$-y + fg$$

XOR-Gate $f \oplus g = y$

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

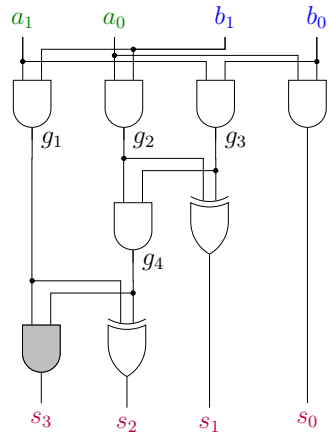
$$-y + f + g - 2fg$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

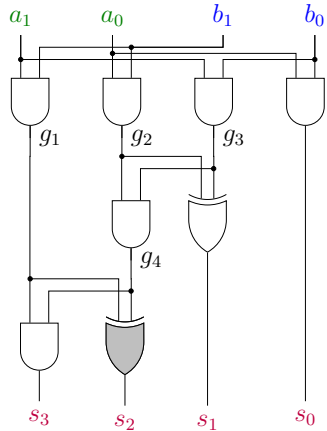
$$s_3 = g_1 \wedge g_4 \qquad -s_3 + g_4 g_1,$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

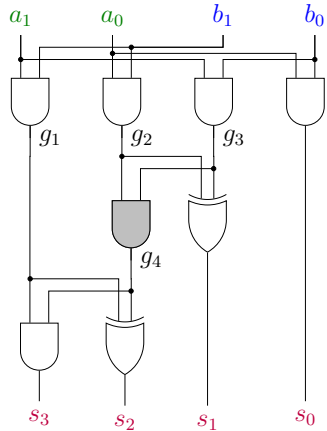
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \qquad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \qquad & -s_2 - 2g_4 g_1 + g_4 + g_1,
\end{aligned}
$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

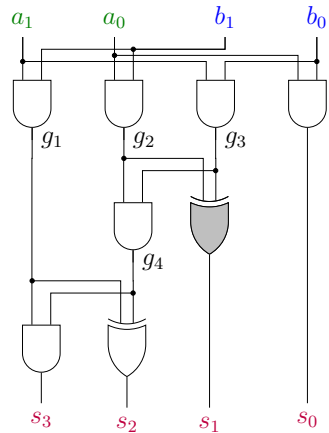
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \qquad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \qquad & -s_2 - 2g_4 g_1 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 \qquad & -g_4 + g_2 g_3,
\end{aligned}
$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

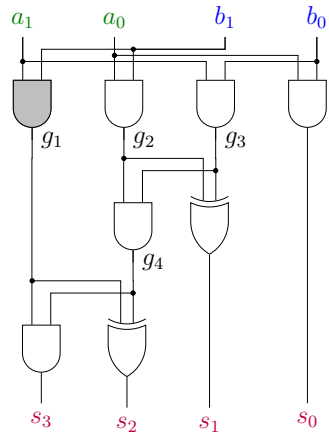
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \quad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \quad & -s_2 - 2g_4 g_1 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 \quad & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 \quad & -s_1 - 2g_2 g_3 + g_2 + g_3,
\end{aligned}
$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

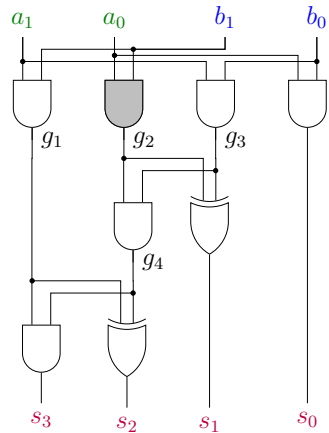
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \quad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \quad & -s_2 - 2g_4 g_1 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 \quad & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 \quad & -s_1 - 2g_2 g_3 + g_2 + g_3, \\
g_1 = a_1 \wedge b_1 \quad & -g_1 + a_1 b_1,
\end{aligned}
$$



From Circuits to Polynomials

Gate polynomials $G(C)$.

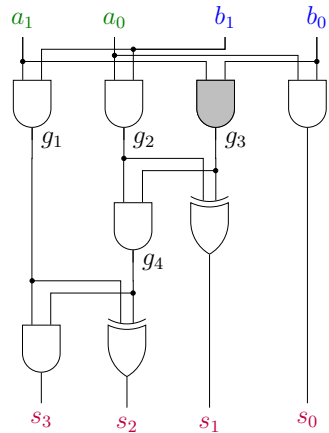
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \quad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \quad & -s_2 - 2g_4 g_1 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 \quad & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 \quad & -s_1 - 2g_2 g_3 + g_2 + g_3, \\
g_1 = a_1 \wedge b_1 \quad & -g_1 + a_1 b_1, \\
g_2 = a_0 \wedge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



From Circuits to Polynomials

Gate polynomials $G(C)$.

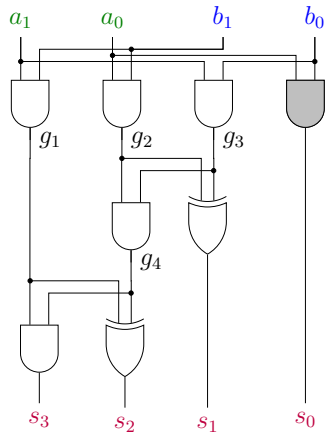
$$
\begin{aligned}
s_3 = g_1 \wedge g_4 \quad & -s_3 + g_4 g_1, \\
s_2 = g_1 \oplus g_4 \quad & -s_2 - 2g_4 g_1 + g_4 + g_1, \\
g_4 = g_2 \wedge g_3 \quad & -g_4 + g_2 g_3, \\
s_1 = g_2 \oplus g_3 \quad & -s_1 - 2g_2 g_3 + g_2 + g_3, \\
g_1 = a_1 \wedge b_1 \quad & -g_1 + a_1 b_1, \\
g_2 = a_0 \wedge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



From Circuits to Polynomials

Gate polynomials $G(C)$.

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



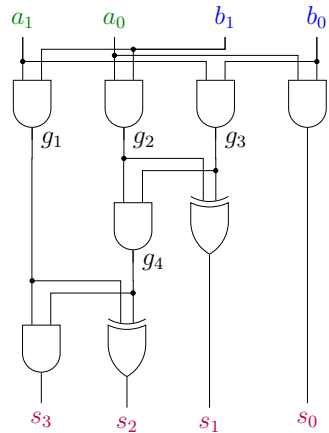
From Circuits to Polynomials

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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



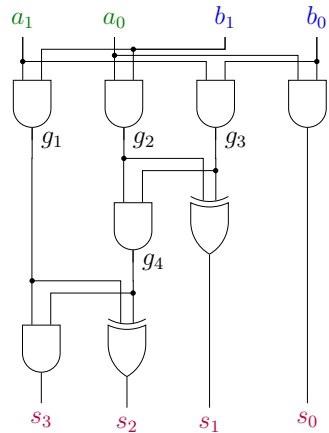
From Circuits to Polynomials

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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

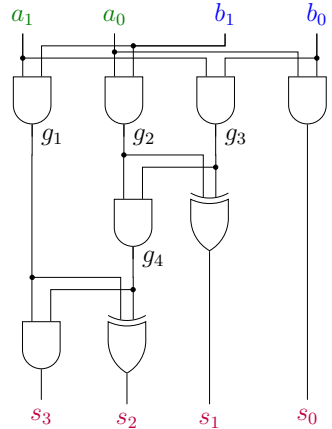


Ideals Associated to Circuits

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'20]

More circuit relations:

- $-s_0 + a_0 b_0$ 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
- ...



Polynomial Circuit Constraints.

A polynomial $p \in \mathbb{Q}[X]$ is a polynomial circuit constraint (PCC) if for all

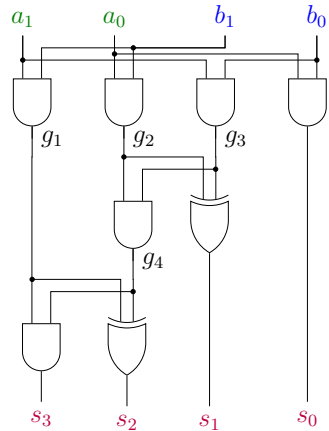
$$(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}) \in \{0,1\}^{2n}$$

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.

Ideals Associated to Circuits

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'20]

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



Ideal. A subset $I \subseteq R[X]$ is called an ideal if

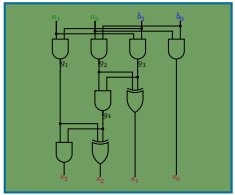
$$\forall\, p, q \in I : p+q \in I \quad \text{and} \quad \forall\, p \in R[X]\, \forall\, q \in I : pq \in I.$$

Multiplier. A circuit C is called a multiplier if

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

Basic Idea of Algebraic Approach

Multiplier



Polynomials

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

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

$= 0$ ✓

$\neq 0$ ✗

Ideal Membership Problem

Basis. A set $G = \{g_1, \ldots, g_m\} \subseteq R[X]$ is called a **basis** of an ideal I if

$$I = \{h_1 g_1 + \cdots + h_m g_m \mid h_1, \ldots, h_m \in R[X]\} = \langle G \rangle$$

Given an arbitrary basis G of I it is not obvious how to decide ideal membership.

Ideal Membership Problem

Basis. A set $G = \{g_1, \ldots, g_m\} \subseteq R[X]$ is called a **basis** of an ideal I if

$$I = \{h_1 g_1 + \cdots + h_m g_m \mid h_1, \ldots, h_m \in R[X]\} = \langle G \rangle$$

Given an arbitrary basis G of I it is not obvious how to decide ideal membership.

Solution: We need a basis with certain structural properties, called Gröbner basis.

Ideal Membership Problem

Basis. A set $G = \{g_1, \ldots, g_m\} \subseteq R[X]$ is called a **basis** of an ideal I if

$$I = \{h_1 g_1 + \cdots + h_m g_m \mid h_1, \ldots, h_m \in R[X]\} = \langle G \rangle$$

Given an arbitrary basis G of I it is not obvious how to decide ideal membership.

Solution: We need a basis with certain structural properties, called Gröbner basis.

Gröbner basis. [Buchberger'65]

- Offers decision procedure for ideal membership problem in $R[X]$, where R is a field.
- Every ideal of $R[X]$ has a finite Gröbner basis.
- Given an arbitrary basis of an ideal, we are able compute a Gröbner basis.

Ideal Membership Problem

Basis. A set $G = \{g_1, \ldots, g_m\} \subseteq R[X]$ is called a **basis** of an ideal I if

$$I = \{h_1 g_1 + \cdots + h_m g_m \mid h_1, \ldots, h_m \in R[X]\} = \langle G \rangle$$

Given an arbitrary basis G of I it is not obvious how to decide ideal membership.

Solution: We need a basis with certain structural properties, called Gröbner basis.

Gröbner basis. [Buchberger'65]

- Offers decision procedure for ideal membership problem in $R[X]$, where R is a field.
- Every ideal of $R[X]$ has a finite Gröbner basis.
- Given an arbitrary basis of an ideal, we are able compute a Gröbner basis.
- Computing a Gröbner basis over $\mathbb{Q}$ and deciding ideal membership is EXPSPACE-complete.[Mayr STACS'89]

Ideal Membership Problem

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'20]

- We can deduce some elements of $I(C)$:
  - Gate polynomials $G(C)$
  - Boolean value constraints $B(C)$
- Let $J(C) = \langle G(C) \cup B(C) \rangle$.
- Lexicographic term order: Reverse topological
  Output variable of a gate is greater than input variables [LvKallaEnescu TCAD'13].

Theorem

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

Proof idea: Application of Buchberger's Product criterion.

Soundness and Completeness

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'20]

Theorem

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

- $J(C) \subseteq I(C)$: soundness
- $I(C) \subseteq J(C)$: completeness

Non-Incremental Checking Algorithm

Verification Algorithm

Reduce specification $\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$ by elements of $G(C) \cup B(C)$

until no further reduction is possible, then C is a multiplier iff remainder is zero.

Verification

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

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

Verification

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

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

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

Verification

$$
\begin{aligned}
&G(C) \cup B(C) = \{ \\
&\quad -s_3 + g_1 g_4, \\
&\quad -s_2 - \boxed{2g_1 g_4} + g_4 + g_1, \\
&\quad -g_4 + g_2 g_3, \\
&\quad -s_1 - 2g_2 g_3 + g_3 + g_2, \\
&\quad -g_1 + a_1 b_1, \\
&\quad -g_2 + a_0 b_1, \\
&\quad -g_3 + a_1 b_0, \\
&\quad -s_0 + a_0 b_0, \\
&\quad -a_1^2 + a_1, \\
&\quad -a_0^2 + a_0, \\
&\quad -b_1^2 + b_1, \\
&\quad -b_0^2 + b_0 \}
\end{aligned}
$$

$$
\begin{aligned}
&8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
&\boxed{8g_1 g_4} + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
&4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
\boxed{4g_2 g_3} + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - \boxed{4a_1 b_1} - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - \boxed{2a_0 b_1} - a_0 b_0 \\
2g_3 + s_0 - 2a_1 b_0 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + s_0 - \boxed{2a_1 b_0} - a_0 b_0 \\
s_0 - a_0 b_0
\end{aligned}
$$

Verification

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + s_0 - 2a_1 b_0 - a_0 b_0 \\
s_0 - \boxed{a_0 b_0} \\
0
\end{aligned}
$$

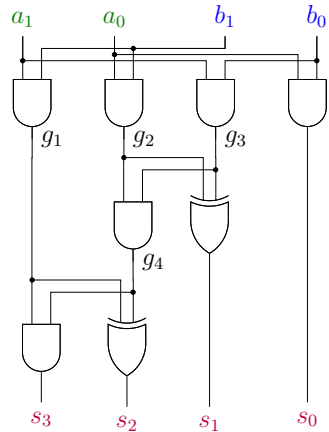
Faulty multiplier

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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



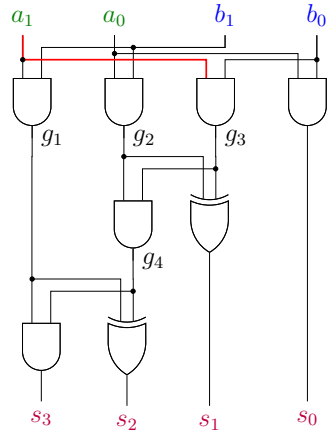
Faulty multiplier

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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



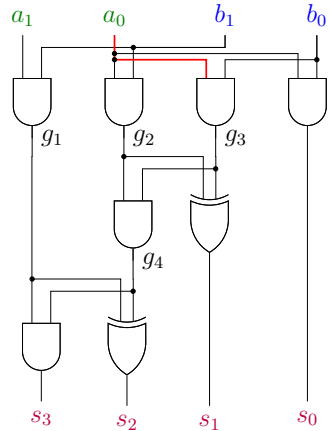
Faulty multiplier

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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



Verification of a faulty multiplier

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + s_0 - 2a_1 b_0 - a_0 b_0
\end{aligned}
$$

Verification of a faulty multiplier

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + s_0 - 2a_1 b_0 - a_0 b_0 \\
s_0 - 2a_1 b_0 + a_0 b_0
\end{aligned}
$$

Verification of a faulty multiplier

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

$$
\begin{aligned}
8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
2g_3 + s_0 - 2a_1 b_0 - a_0 b_0 \\
s_0 - 2a_1 b_0 + a_0 b_0
\end{aligned}
$$

Verification of a faulty multiplier

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

$$
\begin{aligned}
& 8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 8g_1 g_4 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 4g_4 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 4g_2 g_3 + 4g_1 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 4g_1 + 2g_3 + 2g_2 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 2g_3 + 2g_2 + s_0 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 \\
& 2g_3 + s_0 - 2a_1 b_0 - a_0 b_0 \\
& s_0 - 2a_1 b_0 + a_0 b_0 \\
& -2a_1 b_0 + 2a_0 b_0
\end{aligned}
$$

Counter Example

Remainder: $-2a_1b_0 + 2a_0b_0 \neq 0$

Counter Example

Remainder: $-2a_1b_0 + 2a_0b_0 \neq 0$

$a_1 = 0, \quad a_0 = 1$
$b_1 = 0, \quad b_0 = 1$

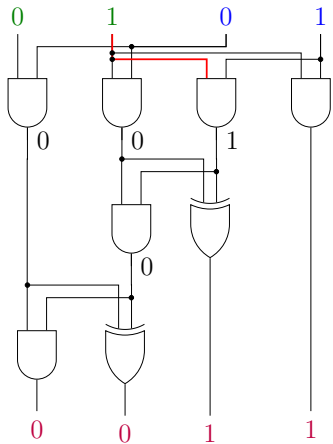
$01 \cdot 01 = 0001$

Counter Example

Remainder: $-2a_1b_0 + 2a_0b_0 \neq 0$

$a_1 = 0, \quad a_0 = 1$
$b_1 = 0, \quad b_0 = 1$

$01 \cdot 01 = 0001$



Non-Incremental Checking Algorithm

Verification Algorithm

Reduce specification $\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$ by elements of $G(C) \cup B(C)$

until no further reduction is possible, then C is a multiplier iff remainder is zero.

Non-Incremental Checking Algorithm

Verification Algorithm

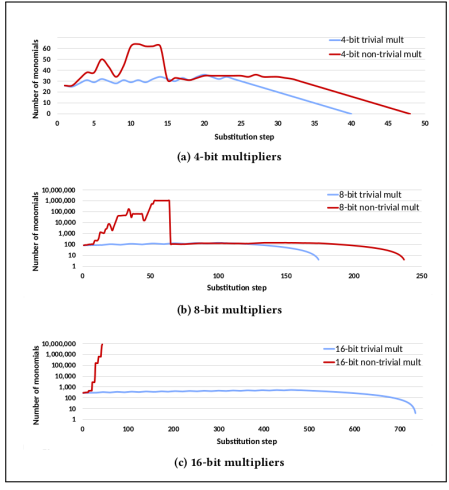
Reduce specification $\sum_{i=0}^{2n-1} 2^i s_i - (\sum_{i=0}^{n-1} 2^i a_i)(\sum_{i=0}^{n-1} 2^i b_i)$ by elements of $G(C) \cup B(C)$ until no further reduction is possible, then C is a multiplier iff remainder is zero.

Computational Problems

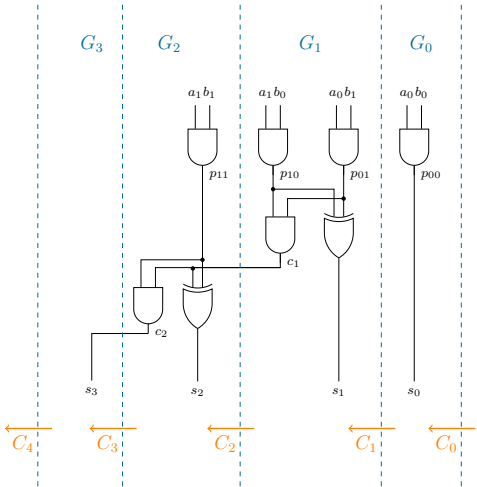
- The number of monomials in the intermediate results increases drastically.
- 8-bit multiplier cannot be verified within 20 minutes.

Non-Incremental Checking Algorithm

[MahzoonGroßeDrechsler ICCAD'18]



Incremental Verification



[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'20]

Let $P_k = \sum_{k=i+j} a_i b_j$.

Column-Wise Checking Algorithm

Input: Circuit C with sliced Gröbner bases G_i
Output: Determine whether C is a multiplier

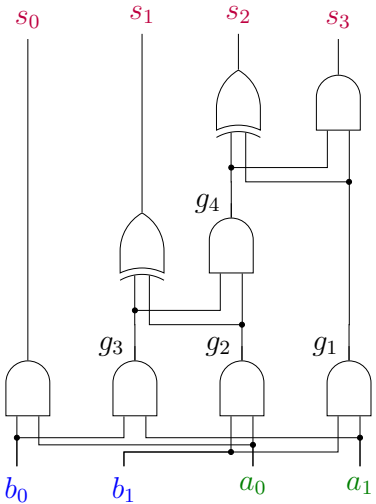
$C_{2n} \leftarrow 0$

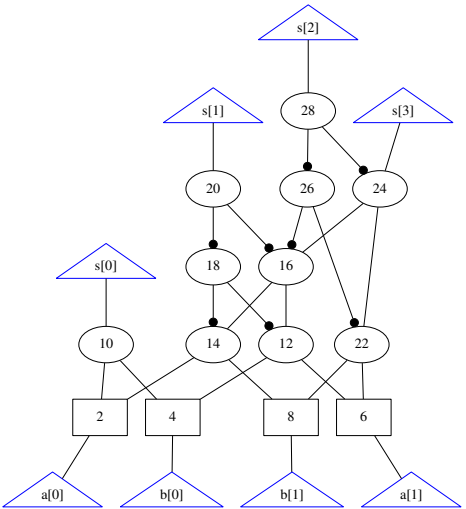
for $i \leftarrow 2n-1$ **to** 0

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

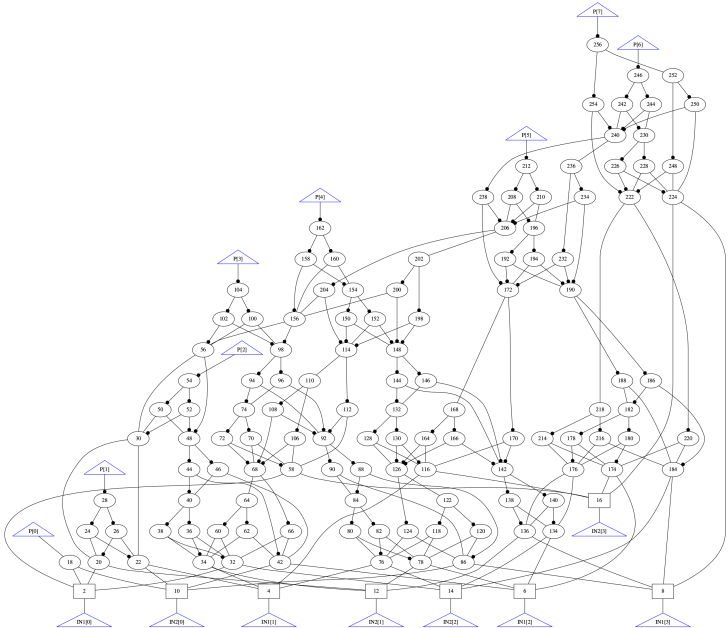
return $C_0 = 0$

AND-Inverter Graphs (AIGs)

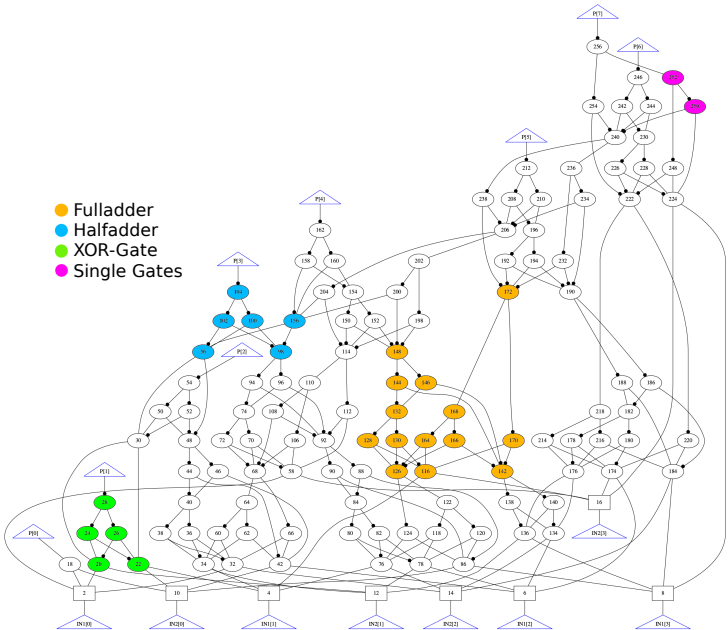




Rewriting



Rewriting



Variable Elimination

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

Identify sub-circuits C_S in the AIGs and eliminate internal variables [RitircBiereKauers DATE'18, KaufmannBiereKauers FMSD'20]:

- Full-adder rewriting
- Half-adder rewriting
- XOR rewriting
- Eliminating nodes with single dependency

Repeated elimination of variables with single dependency [KaufmannBiereKauers FMCAD'19]:

- Does not rely on specific patterns.

Elimination theory of Gröbner bases

Elimination order. Let $X = Y \dot{\cup} Z$ 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 $\langle H \rangle = J$.

Variable Elimination

Example: $I \subseteq \mathbb{Q}[o, x, y, a, b]$. Eliminate y.

Ideal

$$I = \langle$$
$$-o + xy - x - y + 1,$$
$$-x + ab - a - b + 1,$$
$$-y + ab \rangle$$

Extended ideal

$$I = \langle$$
$$-o + xy - x - y + 1,$$
$$-o + xab - x - ab + 1,$$
$$-x + ab - a - b + 1,$$
$$-y + ab \rangle$$

Rewritten ideal

$$I = \langle$$
$$-o + xab - x - ab + 1,$$
$$-x + ab - a - b + 1,$$
$$-y + ab \rangle$$

Elimination ideal

$$I \cap \mathbb{Q}[o, x, a, b] = \langle$$
$$-o + xab - x - ab + 1,$$
$$-x + ab - a - b + 1 \rangle$$

Local variable elimination

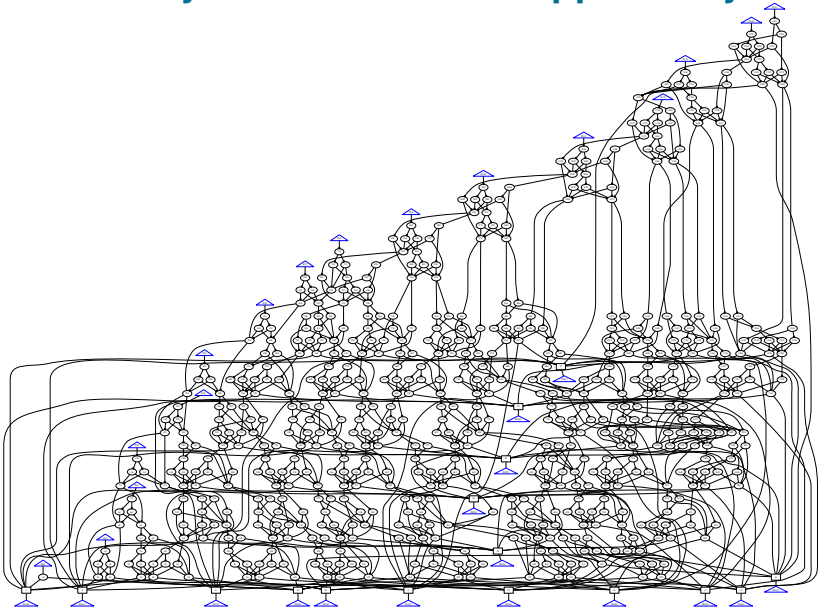
Theorem

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 $\langle G \rangle \cap \mathbb{Q}[Y] = (\langle G_A \rangle + \langle G_B \rangle) \cap \mathbb{Q}[Y] = \langle G_A \rangle + (\langle G_B \rangle \cap \mathbb{Q}[Y])$.

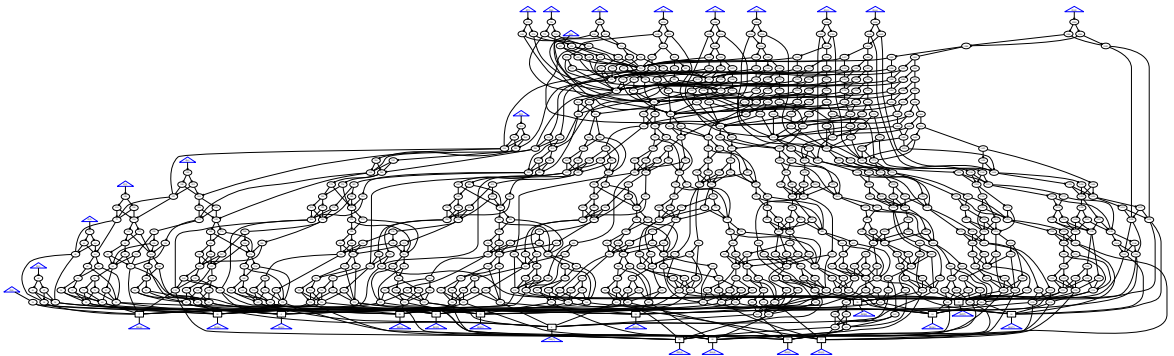
Theorem

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 $<_H$.

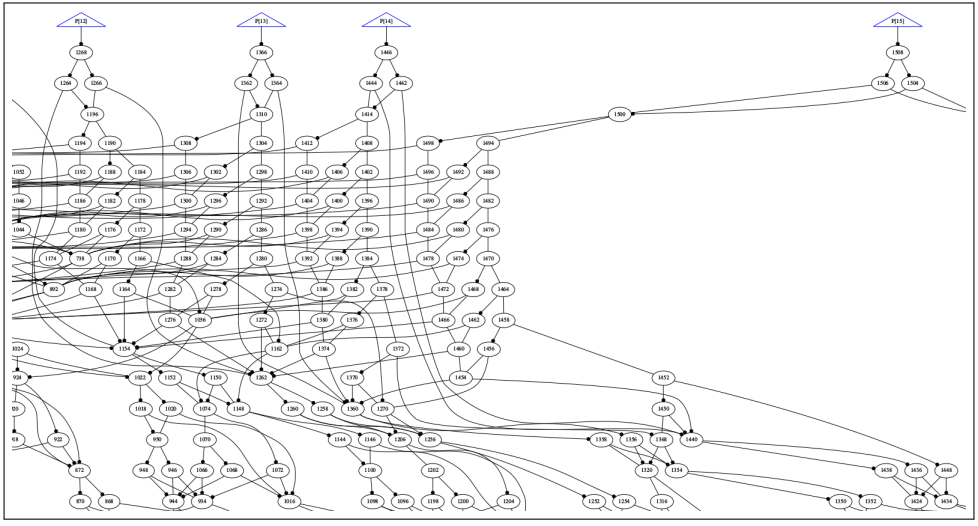
Multiplier – Array Accumulation & Ripple-Carry Adder



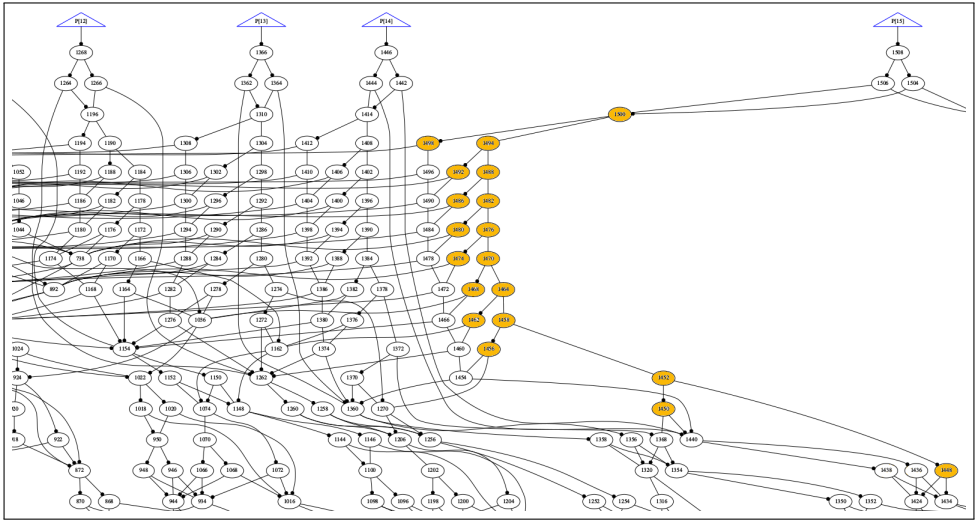
Multiplier – Wallace Tree Acc. & Carry Look-Ahead Adder



Multiplier – Wallace Tree Acc. & Carry Look-Ahead Adder

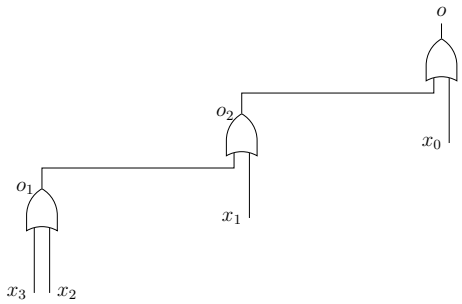


Multiplier – Wallace Tree Acc. & Carry Look-Ahead Adder



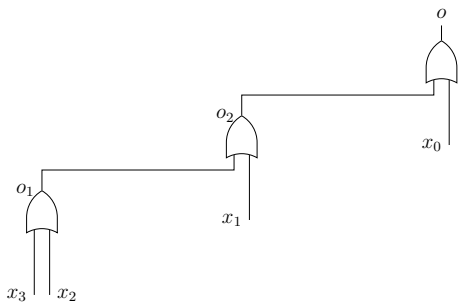
OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + x_0 - o_2 x_0, \\
o_2 = o_1 \vee x_1 \quad & -o_2 + o_1 + x_1 - o_1 x_1, \\
o_1 = x_3 \vee x_2 \quad & -o_1 + x_3 + x_2 - x_3 x_2
\end{aligned}
$$



OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + x_0 - o_2 x_0, \\
o_2 = o_1 \vee x_1 \quad & -o_2 + o_1 + x_1 - o_1 x_1, \\
o_1 = x_3 \vee x_2 \quad & -o_1 + x_3 + x_2 - x_3 x_2
\end{aligned}
$$

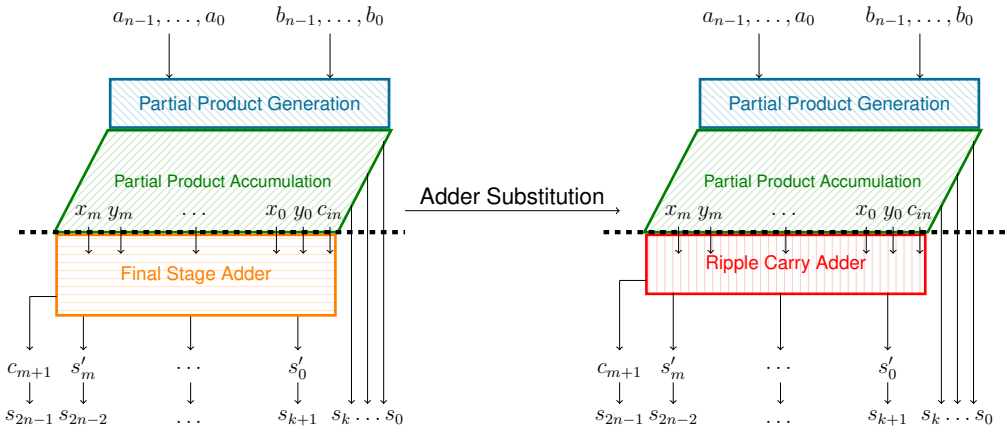


$$o = x_0 + x_1 - x_0x_1 + x_2 - x_0x_2 - x_1x_2 + x_0x_1x_2 + x_3 - x_0x_3 - x_1x_3 + x_0x_1x_3 - x_2x_3 + x_0x_2x_3 + x_1x_2x_3 - x_0x_1x_2x_3$$

$15 = 2^4 - 1$ monomials

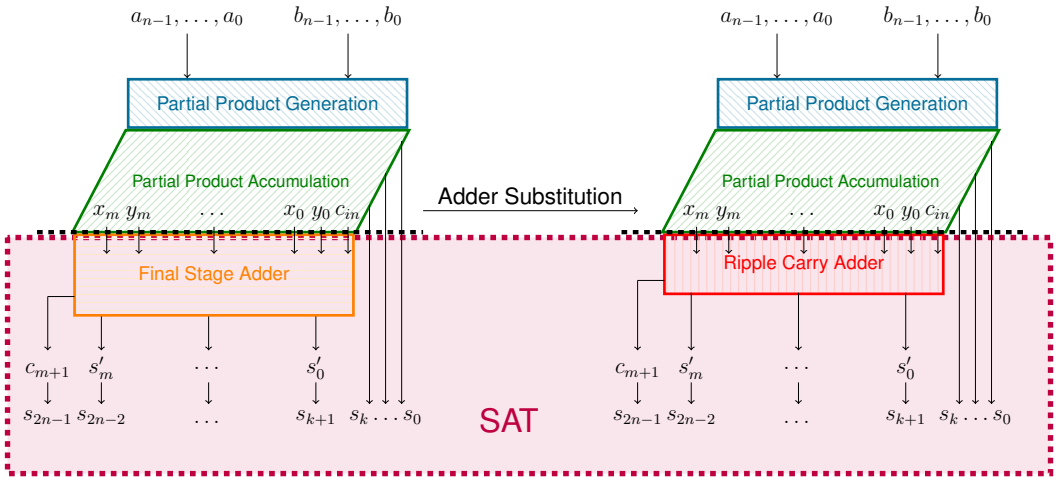
SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]



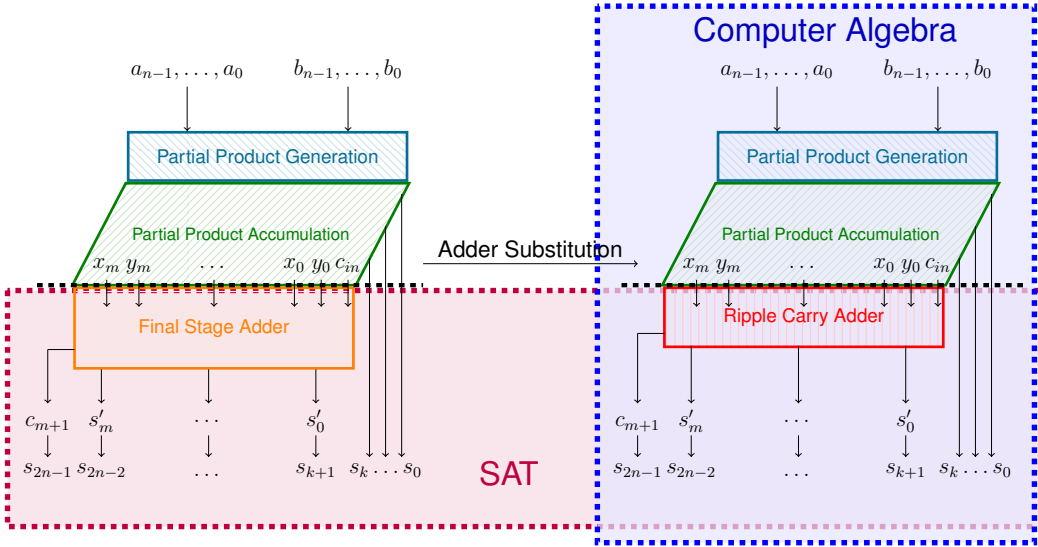
SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]

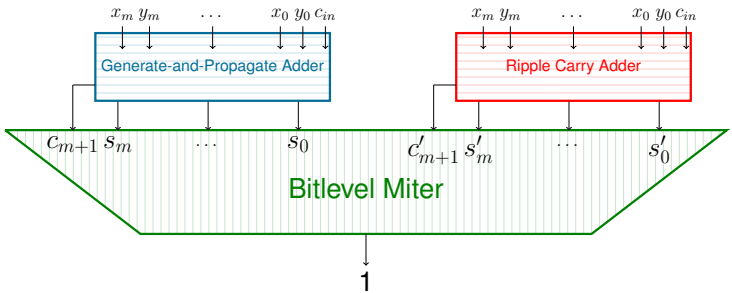


SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]



SAT



- AIG is translated to CNF.
- CNF is given to SAT solver.
- To show correctness SAT solver needs to return UNSAT.

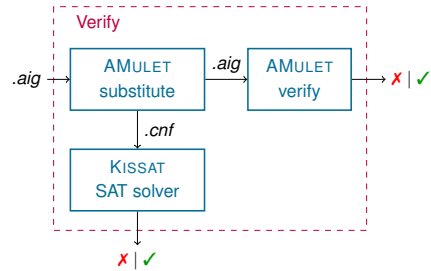
AMulet 2.0

Computer Algebra Systems [KaufmannBiereKauers FMSD'20]

- Mathematica, Singular
- Algorithmic power
  (e.g., Mathematica has > 5000 built-in functions).
- Easy to use.
- Designed for general purposes.

AMULET 2.0 [KaufmannBiere TACAS'21]

- C++ implementation ($\sim$ 6 000 lines)
- Automatically applies adder substitution and circuit verification.
- Designed to make use of properties of polynomials.
- On-the-fly reduction of Boolean value constraints.



```
daniela@daninovo:~/algprop/src/amulet2$ ./amulet ../benchmarks/btor/btor512.aig -verify -v2
```

```
daniela@dantnovo:~/algprop/src/amulet2$ ./amulet ../benchmarks/btor/btor512.aig -verify -v2
[amulet2] AMulet Version 2.0
[amulet2] Aiger multiplier examination tool
[amulet2] Copyright(C) 2020, Daniela Kaufmann, Johannes Kepler University Linz
[amulet2] selected mode: verification
[amulet2] reading '../benchmarks/btor/btor512.aig'
[amulet2] MILOA 2092544 1024 0 1024 2091520
[amulet2] assuming ordering as in BTOR generated benchmarks
[amulet2] (a[0], a[1], ..., a[511]) =(input[0], input[2], ..., input[1022])
[amulet2] (b[0], b[1], ..., b[511]) =(input[1], input[3], ..., input[1023])
[amulet2] (s[0], ..., s[1023]) =(output[0], ..., output[1023])
[amulet2] allocating 2093568 gates
[amulet2] found 262144 partial products
[amulet2] assuming simple pp generator
[amulet2] found 522752 xor-gates
[amulet2] marking xor chain gates
[amulet2] marked 0 xor gates in last slice
[amulet2] remove internal xor gates
```

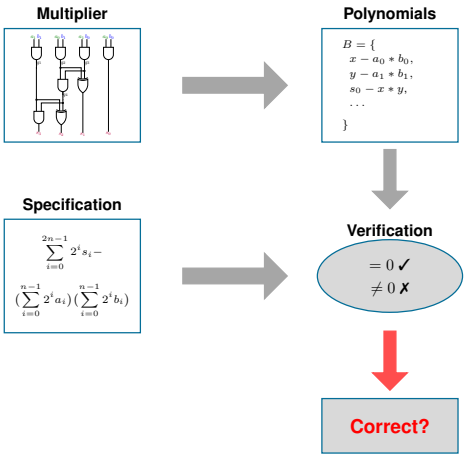
```
daniela@daninovo:~/algprop/src/amulet2$ ./amulet ../benchmarks/btor/btor512.aig -verify -v2
[amulet2] AMulet Version 2.0
[amulet2] Aiger multiplier examination tool
[amulet2] Copyright(C) 2020, Daniela Kaufmann, Johannes Kepler University Linz
[amulet2] selected mode: verification
[amulet2] reading '../benchmarks/btor/btor512.aig'
[amulet2] MILOA 2092544 1024 0 1024 2091520
[amulet2] assuming ordering as in BTOR generated benchmarks
[amulet2] (a[0], a[1], ..., a[511]) =(input[0], input[2], ..., input[1022])
[amulet2] (b[0], b[1], ..., b[511]) =(input[1], input[3], ..., input[1023])
[amulet2] (s[0], ..., s[1023]) =(output[0], ..., output[1023])
[amulet2] allocating 2093568 gates
[amulet2] found 262144 partial products
[amulet2] assuming simple pp generator
[amulet2] found 522752 xor-gates
[amulet2] marking xor chain gates
[amulet2] marked 0 xor gates in last slice
[amulet2] remove internal xor gates
[amulet2] removed 522752 internal xor gates
[amulet2] slicing based on xor
[amulet2] remove single occurence gates
[amulet2] removed 0 single occurence gates
[amulet2] moved 0 gates to smaller slices
[amulet2] remove gates that are not assigned to slices
[amulet2] removed 0 gates that are not assigned to slices
[amulet2]
[amulet2]
[amulet2] started reducing
[amulet2] reducing by slice 1023
[amulet2] after reducing by slice 1023
[amulet2] remainder is -8988465674311579538646525953945123668089884894711532863671504057886633790275048156635423866120376801056005693993569667882939488440720831124642371531973706218888394671243274263815110980062304705972654147604250
2884419075341171231440736956555270413618581675255342293149119973622969239858152417678164812112068608*l4185088+89884656743115795386465259539451236680898848947115328636715040578866337902750481566354238661203768010560056939935696678829
3948844072083112464237153197370621888839467124327426381511098006230470597265414760425028844190753411712314407369565552704136185816752553422931491199736229692398581524176781648121120686 08;
[amulet2]
[amulet2] reducing by slice 1022
[amulet2] after reducing by slice 1022
[amulet2] remainder is -4494232837155789769323262976972561834044942447355766431835752028943316895137524078317711933060188400528002846996784833941469744220360415562321185765986853109444197335621637131907555490031152352986327073802125
1442209537670585615720368478277635206809290837627671146574559986811484619929076208839082406056034304*l4185072-44942328371557897693232629769725618340449424473557664318357520289433168951375240783177119330601884005280028469967848339414
6974422036041556232118576598685310944419733562163713190755549003115235298632707380212514422095376705856157203684782776352068092908376276711465745599868114846199290762088390824060560343 04*l4185070+898846567431157953864652595394512366
8089884894711532863671504057886633790275048156635423866120376801056005693993569667882939488440720831124642371531973706218888394671243274263815110980062304705972654147604250288441907534117123144073695655527041361858167525534229314911
99736229692398581524176781648121120686 08;
[amulet2]
[amulet2] reducing by slice 1021
[amulet2] after reducing by slice 1021
[amulet2] remainder is -2247116418577894884661631488486280917022471223677883215917876014471658447568762039158855966530094200264001423498392416970734872110180207781160592882993426554722098667810818565953777745015576176493163536901062
5721104768835292807860184239138817603404645418813835573287279993405742309964538104419541203028017152*l4185040-22471164185778948846616314884862809170224712236778832159178760144716584475687620391588559665300942002640014234983924169707
3487211018020778116059288299342655472209866781081856595377774501557617649316353690106257211047688352928078601842391388176034046454188138355732872799934057423099645381044195412030280171 52*l4185038-22471164185778948846616314884862809 1
7022471223677883215917876014471658447568762039158855966530094200264001423498392416970734872110180207781160592882993426554722098667810818565953777745015576176493163536901062572110476883529280786018423913881760340464541881383557328727
9993405742309964538104419541203028017152*l4185036+6741349255733684653984894465458842751067413671033649647753628043414975342706286117476567899590282600792004270495177250912204616330540623343481778648980279664166296003432455697861333 2
350467285294794906107031877163314306505878423580552717416452810213936256441506719861839980217226929893614313258623609084051456;
[amulet2]
[amulet2] reducing by slice 1020
[amulet2] after reducing by slice 1020
[amulet2] remainder is -1123558209288947442330815744243140458511235611838941607958938007235829223784381019579427983265047100132000711749196208485367436055090103890580296441496713277361049333905409282976888872507788088246581768450531
2860552384417646403930092119569408801702322709406917786643639996702871154982269052209770601514008576*l4184992-11235582092889474423308157442431404585112356118389416079589380072358292237843810195794279832650471001320007117491962084853
6743605509010389058029644149671327736104933390540928297688887250778808824658176845053128605523844176464039300921195694088017023227094069177866436399967028711549822690522097706015140085 76*l4184990-112355820928894744233081574424314045
8511235611838941607958938007235829223784381019579427983265047100132000711749196208485367436055090103890580296441496713277361049333905409282976888872507788088246581768450531286055238441764640393009211956940880170232270940691778664363
9996702871154982269052209770601514008576*l4184988-1123558209288947442330815744243140458511235611838941607958938007235829223784381019579427983265047100132000711749196208485367436055090103890580296441496713277361049333905409282976888 8
725077880882465817684505312860552384417646403930092119569408801702322709406917786643639996702871154982269052209770601514008576*l4184986+449423283715578976932326297697256183404494244735576643183575202894331689513752407831771193306018
84005280028469967848339414697442203604155623211857659868531094441973356216371319075554900311523529863270738021251442209537670585615720368478277635206809290837627671146574559986811484619929076208839082406056034304;
```

```
5162325512880128*l4182880+2331602729237317690383772906754563939781616870271241129797600845093883887247861002545199672302954187285646008268595647491607228141666475358870244080254409877336161947801752662427625834061571960472642830036 5
029129564274304197859718208744223626663688902106923535445915756829095313676110665757759533718962176;
[amulet2]
[amulet2] reducing by slice 1006
[amulet2] after reducing by slice 1006
[amulet2] remainder is -6857655085992110854069920313984011587592990794915415087640002485570246727199591183956469624420453492016605906672340139681197729828430809879030129647807087874518123375887507830669487747239917530801890676577949
74398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182640-6857655085992110854069920313984011587592990794915415087640002485570246727199591183956469624420453492016605906672340139681197729
82843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182638-6857655085992110854069920313984011587592990794
915415087640002485570246727199591183956469624420453492016605906672340139681197729828430809879030129647807087874518123375887507830669487747239917530801890676577949743989492442411135211237865948125489320265325565745719386987302675092 2
5767960757581162756440064*l4182636-68576550859921108540699203139840115875929907949154150876400024855702467271995911839564696244204534920166059066723401396811977298284308098790301296478070878745181233758875078306694877472399175308018
9067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182634-68576550859921108540699203139840115875929907949154150876400024855702467271995911839564696244204534920166059066723401
3968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182632-68576550859921108540699203139840115
8759299079491541508764000248557024672719959118395646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698
730267509225767960757581162756440064*l4182630-685765508599211085406992031398401158759299079491541508764000248557024672719959118395646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723
9917530801890676577949743989492442411135211237865948125489320265325565745719386987302675092257679607575811627564400 64*l4182628-685765508599211085406992031398401158759299079491541508764000248557024672719959118395646962442045349201660
590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182626-685765508599211085406992
0313984011587592990794915415087640002485570246727199591183956469624420453492016605906672340139681197729828430809879030129647807087874518123375887507830669487747239917530801890676577949743989492442411135211237865948125489320265325565
7457193869873026750922576796075758116275644006 4*l4182624-6857655085992110854069920313984011587592990794915415087640002485570246727199591183956469624420453492016605906672340139681197729828430809879030129647807087874518123375887507830
669487747239917530801890676577949743989492442411135211237865948125489320265325565745719386987302675092257679607575811627564400 64*l4182622-6857655085992110854069920313984011587592990794915415087640002485570246727199591183956469624420
45349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182620-6857655085992
1108540699203139840115875929907949154150876400024855702467271995911839564696244204534920166059066723401396811977298284308098790301296478070878745181233758875078306694877472399175308018906765779497439894924424111352112378659481254893
2026532556574571938698730267509225767960757581162756440064*l4182618-68576550859921108540699203139840115875929907949154150876400024855702467271995911839564696244204534920166059066723401396811977298284308098790301296478070878745181233
7588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182616-68576550859921108540699203139840115875929907949154150876400024855702467271995911839
5646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182614-68
5765508599211085406992031398401158759299079491541508764000248557024672719959118395646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786
594812548932026532556574571938698730267509225767960757581162756440064*l4182612-685765508599211085406992031398401158759299079491541508764000248557024672719959118395646962442045349201660590667234013968119772982843080987903012964780708
7874518123375887507830669487747239917530801890676577949743989492442411135211237865948125489320265325565745719386987302675092257679607575811627564400 64*l4182610-6857655085992110854069920313984011587592990794915415087640002485570246 72
719959118395646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241113521123786594812548932026532556574571938698730267509225767960757581162756440064*
l4182608-685765508599211085406992031398401158759299079491541508764000248557024672719959118395646962442045349201660590667234013968119772982843080987903012964780708787451812337588750783066948774723991753080189067657794974398949244241 1
13521123786594812548932026532556574571938698730267509225767960757581162756440064*l4182606+12343779154785799537325856565171220857667383430847747157775200447402644410895926413112164532395681628562989063201021225142615591369117545778225
4233366052758174132622076597514095205077945031851555443403217840309539181086396340043380228158706625880776477586018342294896577144815166063823293636460929615921152;
[amulet2]
[amulet2] reducing by slice 1005
[amulet2] after reducing by slice 1005
[amulet2] remainder is -3428827542996055427034960156992005793796495397457707543820001242785123363599795591978234812210226746008302953336170069840598864914215404939515064823903543937259061687943753915334743873619958765400945338288974
87199474622120556760561893297406274466013266278287285969349365133754612883980378790581378220032*l4182352-3428827542996055427034960156992005793796495397457707543820001242785123363599795591978234812210226746008302953336170069840598864
91421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120556760561893297406274466013266278287285969349365133754612883980378790581378220032*l4182350-3428827542996055427034960156992005793796495397
45770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120556760561893297406274466013266278287285969349365133754 61
2883980378790581378220032*l4182348-34288275429960554270349601569920057937964953974577075438200012427851233635997955919782348122102267460083029533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009
4533828897487199474622120556760561893297406274466013266278287285969349365133754612883980378790581378220032*l4182346-34288275429960554270349601569920057937964953974577075438200012427851233635997955919782348122102267460083029533361700
6984059886491421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120556760561893297406274466013266278287285969349365133754612883980378790581378220032*l4182344-34288275429960554270349601569920057
9379649539745770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120556760561893297406274466013266278287285969349
365133754612883980378790581378220032*l4182342-342882754299605542703496015699200579379649539745770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354393725906168794375391533474387361
99587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182340-342882754299605542703496015699200579379649539745770754382000124278512336359979559197823481221022674600830
29533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182338-342882754299605542703496
01569920057937964953974577075438200012427851233635997955919782348122102267460083029533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446601326627 82
8728596934936513375461288398037879058137822003 2*l4182336-3428827542996055427034960156992005793796495397457707543820001242785123363599795591978234812210226746008302953336170069840598864914215404939515064823903543937259061687943753915
3347438736199587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182334-3428827542996055427034960156992005793796495397457707543820001242785123363599795591978234812210
2267460083029533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182332-3428827542996
0554270349601569920057937964953974577075438200012427851233635997955919782348122102267460083029533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446
6013266278287285969349365133754612883980378790581378220032*l4182330-34288275429960554270349601569920057937964953974577075438200012427851233635997955919782348122102267460083029533361700698405988649142154049395150648239035439372590616
87943753915334743873619958765400945338288974871994746221205567605618932974062744660132662782872859693493651337546128839803787905813782200 32*l4182328-34288275429960554270349601569920057937964953974577075438200012427851233635997955919
782348122102267460083029533361700698405988649142154049395150648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182326-34
2882754299605542703496015699200579379649539745770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120556760561893
297406274466013266278287285969349365133754612883980378790581378220032*l4182324-342882754299605542703496015699200579379649539745770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354
3937259061687943753915334743873619958765400945338288974871994746221205567605618932974062744660132662782872859693493651337546128839803787905813782200 32*l4182322-3428827542996055427034960156992005793796495397457707543820001242785123 36
3599795591978234812210226746008302953336170069840598864914215404939515064823903543937259061687943753915334743873619958765400945338288974871994746221205567605618932974062744660132662782872859693493651337546128839803787905813782200 32*
l4182320-342882754299605542703496015699200579379649539745770754382000124278512336359979559197823481221022674600830295333617006984059886491421540493951506482390354393725906168794375391533474387361995876540094533828897487199474622120 5
5676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182318-3428827542996055427034960156992005793796495397457707543820001242785123363599795591978234812210226746008302953336170069840598864914215404939515
0648239035439372590616879437539153347438736199587654009453382889748719947462212055676056189329740627446601326627828728596934936513375461288398037879058137822003 2*l4182316+6514772331692505311366424298284811008213341255169644333258002
3612917343908396116247586461431994308174157756113387231326971378433370092693850786231654167334807922172070931324391360133598779216542617961427490522567900178202905784506759726507192148542520592874584334176379375413376447956271970210
```

```
[amulet2] remainder is -4096*l3040-4096*l3038-4096*l3036-4096*l3034-4096*l3032-4096*l3030-4096*l3028-4096*l3026-4096*l3024-4096*l3022+4096*l3016+40960;
[amulet2]
[amulet2] reducing by slice 11
[amulet2] after reducing by slice 11
[amulet2] remainder is -2048*l2870-2048*l2868-2048*l2866-2048*l2864-2048*l2862-2048*l2860-2048*l2858-2048*l2856-2048*l2854+2048*l2848+18432;
[amulet2]
[amulet2] reducing by slice 10
[amulet2] after reducing by slice 10
[amulet2] remainder is -1024*l2716-1024*l2714-1024*l2712-1024*l2710-1024*l2708-1024*l2706-1024*l2704-1024*l2702+1024*l2696+8192;
[amulet2]
[amulet2] reducing by slice 9
[amulet2] after reducing by slice 9
[amulet2] remainder is -512*l2578-512*l2576-512*l2574-512*l2572-512*l2570-512*l2568-512*l2566+512*l2560+3584;
[amulet2]
[amulet2] reducing by slice 8
[amulet2] after reducing by slice 8
[amulet2] remainder is -256*l2456-256*l2454-256*l2452-256*l2450-256*l2448-256*l2446+256*l2440+1536;
[amulet2]
[amulet2] reducing by slice 7
[amulet2] after reducing by slice 7
[amulet2] remainder is -128*l2350-128*l2348-128*l2346-128*l2344-128*l2342+128*l2336+640;
[amulet2]
[amulet2] reducing by slice 6
[amulet2] after reducing by slice 6
[amulet2] remainder is -64*l2260-64*l2258-64*l2256-64*l2254+64*l2248+256;
[amulet2]
[amulet2] reducing by slice 5
[amulet2] after reducing by slice 5
[amulet2] remainder is -32*l2186-32*l2184-32*l2182+32*l2176+96;
[amulet2]
[amulet2] reducing by slice 4
[amulet2] after reducing by slice 4
[amulet2] remainder is -16*l2128-16*l2126+16*l2120+32;
[amulet2]
[amulet2] reducing by slice 3
[amulet2] after reducing by slice 3
[amulet2] remainder is -8*l2086+8*l2080+8;
[amulet2]
[amulet2] reducing by slice 2
[amulet2] after reducing by slice 2
[amulet2] remainder is 4*l2056;
[amulet2]
[amulet2] reducing by slice 1
[amulet2] after reducing by slice 1
[amulet2] remainder is 0;
[amulet2]
[amulet2] reducing by slice 0
[amulet2] after reducing by slice 0
[amulet2] remainder is 0;
[amulet2]
[amulet2]
[amulet2] CORRECT MULTIPLIER
[amulet2]
[amulet2] maximun resident set size:                1966.98 MB
[amulet2] used time for initializing:                 1.33 seconds
[amulet2] used time for slicing/elimination:          7.62 seconds
[amulet2] used time for reduction:                   30.72 seconds
[amulet2] used time for freeing memory:               3.40 seconds
[amulet2] total process time:                        43.07 seconds
daniela@danilnovo:~/algprop/src/amulet2$
```

Is the circuit really really correct?

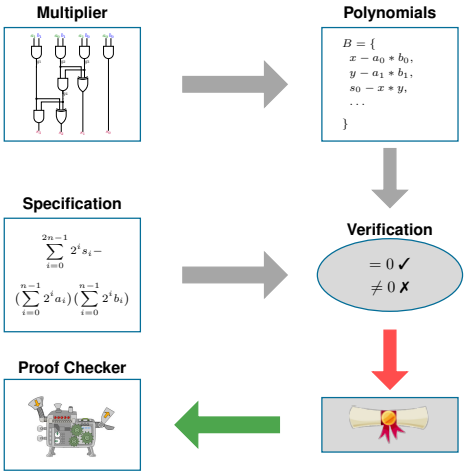
Proofs



Problem:

- Can we trust our own implementation?
- Is the verification process correct?

Proofs



Problem:

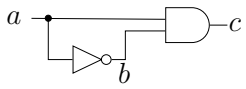
- Can we trust our own implementation?
- Is the verification process correct?

Solution:

Validate result of verification process.

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

Practical Algebraic Calculus



$$
\begin{aligned}
G(C) &= \{-b+1-a,\ -c+ab\} \\
B(C) &= \{-a^2+a\} \\
\text{Spec} &= c
\end{aligned}
$$

[RitircBiereKauers SC2'18]

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

[KaufmannBiereKauers FMCAD'19]

```
* : -b+1-a,    a,          -a*b;
+ : -a*b,      -c+a*b,     -c;
* : -c,        -1,         c;
```

[KaufmannFleuryBiere FMCAD'20]

```
3   * 1,   a,    -a*b;
1   d;
4   + 3,   2,    -c;
5   * 5,   -1,   c;
```

Proof Checker

PACTRIM [RitircBiereKauers SC2'18]

- Written in C (~ 2000 lines of code).
- Supports older PAC formats, where antecedents are given explicitly.

PACHECK [KaufmannFleuryBiere FMCAD'20]

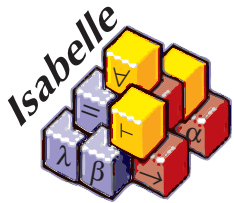
- Written in C (~ 1700 lines of code).
- Supports new calculus.
- Backward compatible to older PAC formats.

Is the circuit really really really correct?

PASTÈQUE

[KaufmannFleuryBiere FMCAD'20]

Theorem Prover Isabelle/HOL



Refinement Approach, relying on Isabelle's Refinement Framework

- abstract specification on ideals: specification in ideal
- final step: executable checker

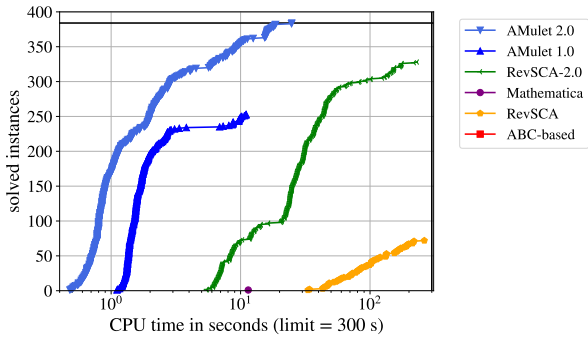
Isabelle's Archive of Formal Proofs 8000 lines of code

Recent Related Work

- M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi. Verification of Gate-level Arithmetic Circuits by Function Extraction. In DAC, 2015.
- C. Yu, M. Ciesielski, and A. Mishchenko. Fast Algebraic Rewriting Based on And-Inverter Graphs. IEEE TCAD, 2018.
- M. Ciesielski, T. Su, A. Yasin, and C. Yu. Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model. IEEE TCAD, 2019.

- A. 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'16.
- 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.
- A. Mahzoon, D. Große, C. Scholl, and R. Drechsler. Towards Formal Verification of Optimized and Industrial Multipliers. In DATE, 2020.

Evaluation

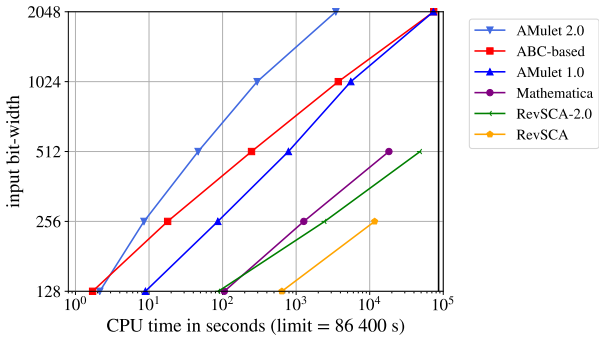


Comparing our approaches to most recent related work.

- Our work
  Mathematica [KaufmannBiereKauers FMSD'20]
  AMULET [KaufmannBiereKauers FMCAD'19]
  AMULET 2.0 [KaufmannBiere TACAS'21]
- Yu et al. [ABC-based]
- Mahzoon et al.
  [RevSCA] [RevSCA-2.0]

- 384 different multipliers
- Input bit-width $n = 64$
- Time-limit: 300 sec

Evaluation



Comparing our approaches to most recent related work.

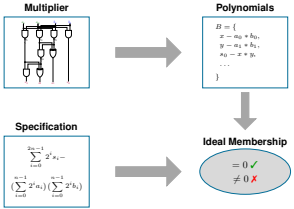
- Our work
  Mathematica [KaufmannBiereKauers FMSD'20]
  AMULET [KaufmannBiereKauers FMCAD'19]
  AMULET 2.0 [KaufmannBiere TACAS'21]
- Yu et al. [ABC-based]
- Mahzoon et al.
  [RevSCA] [RevSCA-2.0]

- Array ripple-carry multiplier
- Input bit-width n in $[128, 2048]$
- Time-limit: 86 400 sec (24h)

Conclusion

- **Is the circuit correct?**
  Yes, because we have tested some cases.
- **Is the circuit really correct?**
  Yes, because we applied formal verification.
- **Is the circuit really really correct?**
  Yes, because we generated and checked a proof certificate.
- **Is the circuit really really really correct?**
  Yes, because we used a verified proof checker.

Basic Idea of Algebraic Approach



Incremental Verification

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'19]

Let $P_k = \sum_{k=i+j} a_i b_j$.

Column-Wise Checking Algorithm

Input: Circuit C with sliced Gröbner bases G_i
Output: Determine whether C is a multiplier

$C_{2n} \leftarrow 0$

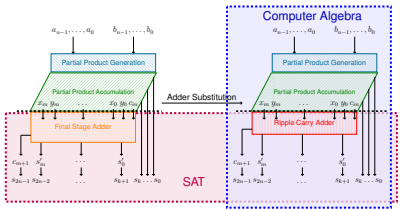
for $i \leftarrow 2n-1$ **to** 0

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

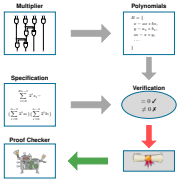
return $C_0 = 0$

SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]



Proofs



Problem:

- Can we trust our own implementation?
- Is the verification process correct?

Solution:

Validate result of verification process.

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

COMBINING SAT AND COMPUTER ALGEBRA FOR CIRCUIT VERIFICATION

Daniela Kaufmann

Johannes Kepler University

Linz, Austria

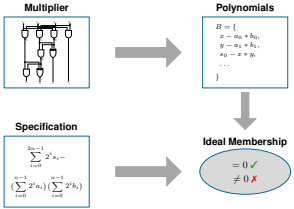
Beyond Satisfiability Workshop

Simons Institute for the Theory of Computing (online)

February 16, 2021

JOHANNES KEPLER
UNIVERSITY LINZ

Basic Idea of Algebraic Approach



Incremental Verification

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'19]

Let $P_k = \sum_{k=i+j} a_i b_j$.

Column-Wise Checking Algorithm

Input: Circuit C with sliced Gröbner bases G_i
Output: Determine whether C is a multiplier

$C_{2n} \leftarrow 0$

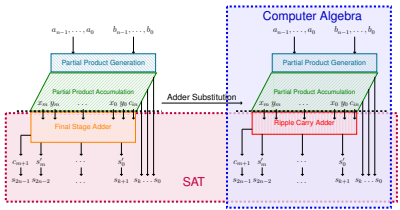
for $i \leftarrow 2n-1$ **to** 0

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

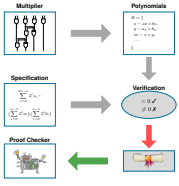
return $C_0 = 0$

SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]



Proofs



Problem:

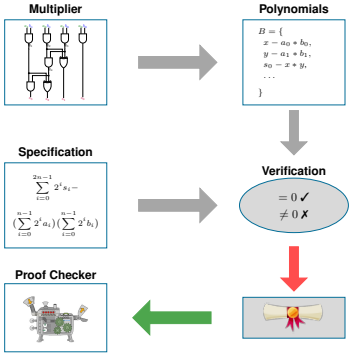
- Can we trust our own implementation?
- Is the verification process correct?

Solution:

Validate result of verification process.

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

Proofs



Problem:

- Can we trust our own implementation?
- Is the verification process correct?

Solution:

Validate result of verification process.

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

Ideal

Ideal membership problem. Given a polynomial $f \in R[X]$ and a (finite) set of polynomials $P \subseteq R[X]$, decide whether $f \in \langle P \rangle$, where $\langle P \rangle$ is the smallest ideal containing all elements of P, also known as the ideal *generated by* P.

Ideal. A nonempty subset $I \subseteq R[X]$ is called an ideal if

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

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

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

Ideal Membership Problem

Given an ideal $I = \langle p_1, \ldots, p_s \rangle \subset R[X]$ we can decide whether a given polynomial $f \in R[X]$ lies in I as follows:

- Compute a Gröbner basis G for I using Buchberger's algorithm.
- Given a Gröbner basis G, there is a computable function $\mathrm{red}_G \colon R[X] \to R[X]$ such that $\forall\, f \in R[X] : \mathrm{red}_G(f) = 0 \iff f \in \langle G \rangle$.

Polynomial Calculus

[CleggEdmondsImpagliazzo STOC'96]

Let $G \subseteq R[X]$ and $f \in R[X]$.

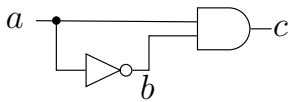
Proof: Sequence $P = (p_1, \dots p_n)$, where each p_k is obtained by one of the rules:

Boolean Axiom	$\dfrac{}{x_j^2 - x_j}$	$x_j \in X$
Addition	$\dfrac{p_i \qquad p_j}{\alpha p_i + \beta p_j}$	$\alpha, \beta \in R$ p_i, p_j appearing earlier in the proof or are contained in G
Multiplication	$\dfrac{p_i}{q p_i}$	p_i appearing earlier in the proof or is contained in G and $q \in [X]$ being arbitrary

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



$$
\begin{aligned}
R &= \mathbb{Q} \\
G &= \{ \quad -b+1-a, && b = \neg a \\
&\qquad\; -c+ab\} && c = a \wedge b = a \wedge \neg a \\
f &= \quad c
\end{aligned}
$$

$$\mathrm{red}_G(f) = 0$$

$$
\begin{array}{lcccc}
 & -b+1-a & & & \\
* & \overline{\qquad\qquad} & & & \\
 & -ab+a-a^2 & & a^2-a & \\
+ & \overline{\qquad\qquad\qquad\qquad} & & \overline{\qquad\qquad} & \\
 & & -ab & & -c+ab \\
 & + & \overline{\qquad\qquad} & & \overline{\qquad\qquad} \\
 & & & c &
\end{array}
$$

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

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

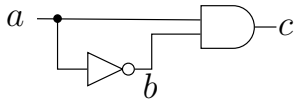
Practical Algebraic Calculus (PAC)

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;
-c+ab;
-a^2+a;
```

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

```
a,
-1,
a^2-a,
-c+a*b,
-1,
```

```
-a*b+a-a^2;
a^2-a;
-a*b;
-c;
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 $o = " + "$

- Target check: at least one p_i is equal to target f

Proof Checking Algorithm

input G sequence of given polynomials
$r_1 \cdots r_k$ sequence of PAC proof rules
f target polynomial

output "incorrect", "target-checked", or "correct-proof"

$P_0 \leftarrow G$
for $i \leftarrow 1 \ldots k$
  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** $P_i \leftarrow \text{append}(P_{i-1}, p_i)$
    else return "incorrect"
  case $o_i = *$
    if $v_i \in P_{i-1} \ \wedge \ p_i = v_i * w_i$ **then** $P_i \leftarrow \text{append}(P_{i-1}, p_i)$
    else return "incorrect"
for $p_i \in P_k$
  if $p_i = f$ **then return** "target-checked"
return "correct-proof"

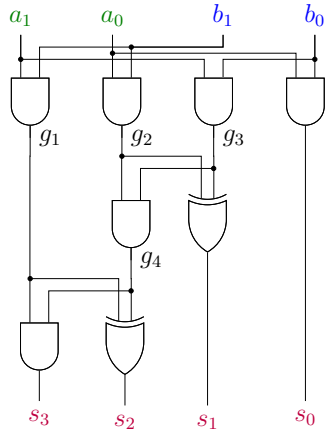
Proof Generation in AMULET

Gate polynomials $G(C)$.

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

Boolean value constraints $B(C)$.

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



Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

1 $H \leftarrow G$;
2 **while** $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
3 $\quad r \leftarrow$ Reduce(p, q);
4 $\quad H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
5 **return** $G' \leftarrow H$

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

$H \leftarrow G$;
while $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
    $r \leftarrow$ Reduce(p, q);
    $H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
return $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

$r \leftarrow f$;
for $i = 1$ **to** m **do**
    $r \leftarrow$ Reduce(r, g_i);
return $r = 0$

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

$H \leftarrow G$;
while $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
  $r \leftarrow$ Reduce(p, q);
  $H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
return $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

$r \leftarrow f$;
for $i = 1$ **to** m **do**
  $r \leftarrow$ Reduce(r, g_i);
return $r = 0$

Algorithm: Reduce(p, q)

Input : Polynomials p and q
Output: Remainder r of reducing p modulo q

$p_d \leftarrow$ Divide-by-lt(p, q);
$p_m \leftarrow$ Multiply(p_d, q);
$r \leftarrow$ Add(p, p_m);
return r

Preprocessing Example

Example: $I \subseteq \mathbb{Q}[o, x, y, a, b]$. Eliminate y.

Ideal

$$I = \langle$$
$$-o + xy - x - y + 1,$$
$$-x + ab - a - b + 1,$$
$$-y + ab\rangle$$

Elimination ideal

$$I \cap \mathbb{Q}[o, x, a, b] = \langle$$
$$-o + xab - x - ab + 1,$$
$$-x + ab - a - b + 1\rangle$$

Algorithm: Reduce(p, q)

Input : Polynomials p and q
Output: Remainder r of reducing p modulo q

1 $p_d \leftarrow$ Divide-by-lt(p, q);
2 $p_m \leftarrow$ Multiply(p_d, q);
3 $r \leftarrow$ Add(p, p_m);
4 **return** r

Reduce $(-o + xy - x - y + 1, -y + ab)$

$$\begin{aligned} p_d &= x - 1 \\ p_m &= (x - 1)(-y + ab) = -xy + y + xab - ab \\ r &= (-o + xy - x - y + 1) + (-xy + y + xab - ab) \\ &= -o + xab - x - ab + 1 \end{aligned}$$

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

$H \leftarrow G$;
while $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
  $r \leftarrow$ Reduce(p, q);
  $H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
return $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

$r \leftarrow f$;
for $i = 1$ **to** m **do**
  $r \leftarrow$ Reduce(r, g_i);
return $r = 0$

Algorithm: Reduce(p, q)

Input : Polynomials p and q
Output: Remainder r of reducing p modulo q

$p_d \leftarrow$ Divide-by-lm(p, q);
$p_m \leftarrow$ Multiply(p_d, q);
$r \leftarrow$ Add(p, p_m);
return r

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

1 $H \leftarrow G$;
2 **while** $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
3 $\quad r \leftarrow$ Reduce(p, q);
4 $\quad H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
5 **return** $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

1 $r \leftarrow f$;
2 **for** $i = 1$ **to** m **do**
3 $\quad r \leftarrow$ Reduce(r, g_i);
4 **return** $r = 0$

Algorithm: Reduce(p, q)

Input : Polynomials p and q
Output: Remainder r of reducing p modulo q

1 $p_d \leftarrow$ Divide-by-lm(p, q);
2 $p_m \leftarrow$ Multiply(p_d, q);
3 Print-Mult-Rule(q, p_d, p_m) ; `// *: q, p_d, p_m;`
4 $r \leftarrow$ Add(p, p_m);
5 Print-Add-Rule(p, p_m, r) ; `// +: p, p_m, r;`
6 **return** r

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

1 $H \leftarrow G$;
2 **while** $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
3 $\quad r \leftarrow$ Reduce(p, q);
4 $\quad H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
5 **return** $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

1 $r \leftarrow f$;
2 **for** $i = 1$ **to** m **do**
3 $\quad r \leftarrow$ Reduce(r, g_i);
4 **return** $r = 0$

Algorithm: Reduce(p, q)

Input : Polynomials p and q
Output: Remainder r of reducing p modulo q

1 $p_d \leftarrow$ Divide-by-lm(p, q);
2 $p_m \leftarrow$ Multiply(p_d, q);
3 Print-Mult-Rule(q, p_d, p_m) ; `// *: q, p_d, p_m;`
4 $r \leftarrow$ Add(p, p_m);
5 Print-Add-Rule(p, p_m, r) ; `// +: p, p_m, r;`
6 **return** r

Proof Generation in AMULET

Algorithm: Preprocessing

Input : Gröbner basis $G = \{g_1, \ldots, g_m\}$
Output: Simplified Gröbner basis $G' = \{g'_1, \ldots, g'_n\}$

$H \leftarrow G$;
while $\exists q \in H : \mathrm{lt}(q)$ occurs in only one $p \in H$ with $p \neq q$ **do**
  $r \leftarrow$ Reduce(p, q, **1**);
  $H \leftarrow (H \setminus \{p, q\}) \cup \{r\}$;
return $G' \leftarrow H$

Algorithm: Spec-Reduction

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f
Output: Determine whether $f \in \langle G \rangle$

$r \leftarrow f$;
for $i = 1$ **to** m **do**
  $r \leftarrow$ Reduce(r, g_i, **0**);
Print-Add-Rules-for-Factor-Stack();
return $r = 0$

Algorithm: Reduce(p, q, print)

Input : Polynomials p and q, printing flag
Output: Remainder r of reducing p modulo q

$p_d \leftarrow$ Divide-by-lt-and-negate(p, q);
$p_m \leftarrow$ Multiply(p_d, q);
Print-Mult-Rule(q, p_d, p_m) ; `// *: q, p_d, p_m;`
$r \leftarrow$ Add(p, p_m);
if print **then** Print-Add-Rule(p, p_m, r);
else Push-to-Factor-Stack(p_m);
return r

Reduction Example

Algorithm: Spec-Reduction in AMULET

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f

Output: Determine whether $f \in \langle G \rangle$

```
1 r ← f;
2 for i = 1 to m do
3 |   r ← Reduce(r, g_i, 0);
4 Print-Add-Rules-Factor-Stack();
5 return r = 0
```

Reduction	Stack
$r_1 = f$	
$r_2 = r_1 + p_{d_1} g_1$	$p_{d_1} g_1$
$r_3 = r_2 + p_{d_2} g_2$	$p_{d_2} g_2$
$\vdots$	$\vdots$
$r_{m+1} = r_m + p_{d_m} g_m = 0$	$p_{d_m} g_m$

Reduction Example

Algorithm: Spec-Reduction in AMULET

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f

Output: Determine whether $f \in \langle G \rangle$

```
1 r ← f;
2 for i = 1 to m do
3 |   r ← Reduce(r, g_i, 0);
4 Print-Add-Rules-Factor-Stack();
5 return r = 0
```

Reduction	Stack
$r_1 = f$	
$r_2 = r_1 + p_{d_1} g_1$	$p_{d_1} g_1$
$r_3 = r_2 + p_{d_2} g_2$	$p_{d_2} g_2$
$\vdots$	$\vdots$
$r_{m+1} = r_m + p_{d_m} g_m = 0$	$p_{d_m} g_m$

$$0 \quad = r_{m+1} = f + p_{d_1} g_1 + p_{d_2} g_2 + \ldots + p_{d_m} g_m$$

Reduction Example

Algorithm: Spec-Reduction in AMULET

Input : Ordered Gröbner basis $G = \{g_1, \ldots, g_m\}$, Specification f

Output: Determine whether $f \in \langle G \rangle$

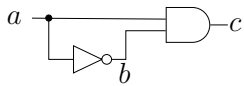
```
1 r ← f;
2 for i = 1 to m do
3 |   r ← Reduce(r, g_i, 0);
4 Print-Add-Rules-Factor-Stack();
5 return r = 0
```

Reduction	Stack
$r_1 = f$	
$r_2 = r_1 + p_{d_1} g_1$	$p_{d_1} g_1$
$r_3 = r_2 + p_{d_2} g_2$	$p_{d_2} g_2$
$\vdots$	$\vdots$
$r_{m+1} = r_m + p_{d_m} g_m = 0$	$p_{d_m} g_m$

$$f = -1 \cdot (p_{d_1} g_1 + p_{d_2} g_2 + \ldots + p_{d_m} g_m)$$

Practical Algebraic Calculus

[RitircBiereKauers SC2'18]



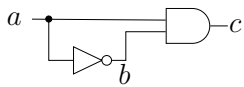

```
P =   -b+1-a,      b = ¬a
      -c+a*b,      c = a ∧ b = a ∧ ¬a
      -a^2+a       a = ⊥ ∨ a = ⊤
Spec = c           c = ⊥
```

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

$\forall\, p, q \in I : p + q \in I$ and $\forall\, p \in R[X]\ \forall\, q \in I : pq \in I$

Practical Algebraic Calculus

[RitircBiereKauers SC2'18]




```
P =   -b+1-a,        b = ¬a
      -c+a*b,        c = a ∧ b = a ∧ ¬a
      -a^2+a         a = ⊥ ∨ a = ⊤
Spec = c             c = ⊥
```

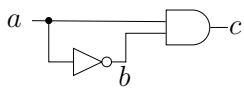
```
* : -b+1-a,          a,          -a*b+a-a^2;
* : -a^2+a,          -1,         a^2-a;
+ : -a*b+a-a^2,      a^2-a,      -a*b;
+ : -a*b,            -c+a*b,     -c;
* : -c,              -1,         c;
```

Can be checked by our older proof checker PACTRIM.

1. Boolean Variables

[KaufmannFleuryBiere FMCAD'20]

Handle Boolean-value constraints implicitly to reduce number of proof steps.



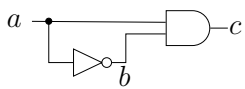

```
P =   -b+1-a,
      -c+a*b,
      -a^2+a
Spec =  c
```

```
* : -b+1-a,      a,        -a*b;
+ : -a*b,        -c+a*b,   -c;
* : -c,          -1,       c;
```

2. Indices

[KaufmannFleuryBiere FMCAD'20]

Introduce indices to reduce proof size.



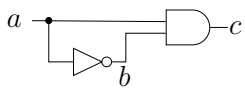

```
P =   1 -b+1-a;
      2 -c+a*b;
Spec = c
```

```
3   * 1,   a,   -a*b;
4   + 3,   2,   -c;
5   * 4,  -1,   c;
```

3. Deletion Rule

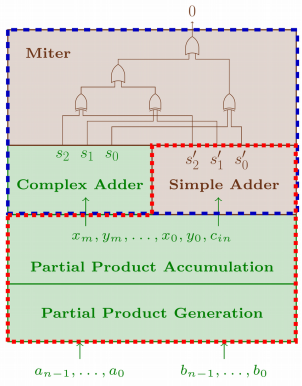
[KaufmannFleuryBiere FMCAD'20]

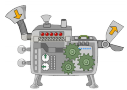
Introduce a deletion rule to reduce the memory usage of the proof checker.




```
P =   1 -b+1-a;
      2 -c+a*b;
Spec = c
```

```
3   * 1,   a,   -a*b;
1   d;
4   + 3,   2,   -c;
2   d;
3   d;
5   * 4,  -1,   c;
```





SAT

DRUP

Computer Algebra

Practical Algebraic Calculus

Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{\overline{x} \vee \overline{y} \qquad y \vee z}{\overline{x} \vee z}$$

Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

```
P =   1 x*y;
      2 y*z-y-z+1;
Spec =  -x*z+x
```

```
3  =  f, -z+1;
4  *  3,  y-1, -f*y+f-y*z+y+z-1;
5  +  2,    4, -f*y+f;
```

Ext(i, v, p) $(X, P) \Rightarrow (X \cup \{v\}, P(i \mapsto -v + p))$
provided that $P(i) = \bot$ and $v \notin X$ and $p \in R[X]/\langle B(X)\rangle$,
and $p^2 - p \equiv 0 \mod \langle B(X)\rangle$.

Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

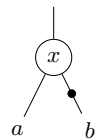
$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

```
P =   1 x*y;
      2 y*z-y-z+1;
Spec =  -x*z+x
```

```
 3  =  f, -z+1;
 4  *  3,  y-1, -f*y+f-y*z+y+z-1;
 5  +  2,    4, -f*y+f;
 6  *  1,    f, f*x*y;
 7  *  5,    x, -f*x*y+f*x;
 8  +  6,    7, f*x;
 9  *  3,    x, -f*x-x*z+x;
10  +  8,    9, -x*z+x;
```

From DRUP to PAC

1. Polynomial encoding of CNF



	Propostional Formula	Polynomial Relation
	$(x \leftrightarrow a \wedge \overline{b}) = \top$	$-x + a(1-b) = 0$
CNF	$(x \vee \overline{a} \vee b) = \top$	$(1-x)a(1-b) = 0$
	$(\overline{x} \vee a) = \top$	$x(1-a) = 0$
	$(\overline{x} \vee \overline{b}) = \top$	$xb = 0$

- Add bit-flipping polynomials (similar to “Polynomial Calculus with Resolution”):
  Clause $x \vee y \vee z$ can be translated to $(1-x)(1-y)(1-z) = 0$, which generates 2^3 monomials.
  If we introduce $-f_x + 1 - x = 0$, $-f_y + 1 - y = 0$, $-f_z + 1 - z = 0$, the same equation can be depicted as $f_x f_y f_z = 0$.

From DRUP to PAC

2. Encoding of resolution steps

- Encode resolution using the traces provided in TraceCheck:

```
1  1 -2 -3 0 0
2  1  2  0 0
3 -1 -2  0 0
4 -1  2  0 0
5  3  0  0
6 -2  0 3 1 5 0
7  0  4 2 6 0
```

Let a =1, b =2 and c =3.

We encode the first resolution step of rule 6 (resolving clause 3 and 1).

Thus from $a \vee \overline{b} \vee \overline{c}$ and $\overline{a} \vee \overline{b}$ we resolve the clause $\overline{b} \vee \overline{c}$.

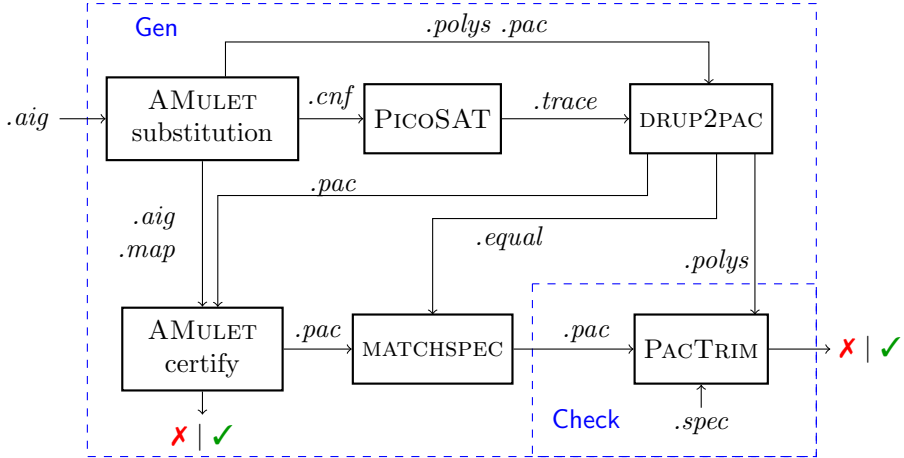
The corresponding PAC encoding is:

```
* :         b*a,     c,  c*b*a;
+ : -c*b*a+c*b, c*b*a,    c*b;
```

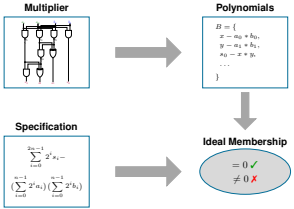
3. Merge PAC proofs

PAC proofs can be merged by combining constraint sets and proof rules.

From DRUP to PAC



Basic Idea of Algebraic Approach



Incremental Verification

[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'19]

Let $P_k = \sum_{k=i+j} a_i b_j$.

Column-Wise Checking Algorithm

Input: Circuit C with sliced Gröbner bases G_i
Output: Determine whether C is a multiplier

$C_{2n} \leftarrow 0$

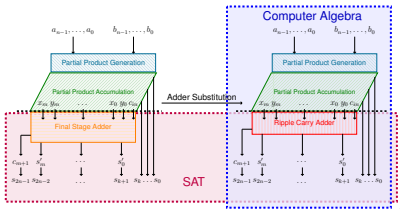
for $i \leftarrow 2n-1$ **to** 0

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

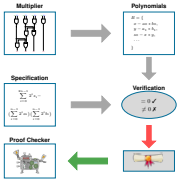
return $C_0 = 0$

SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]



Proofs



Problem:

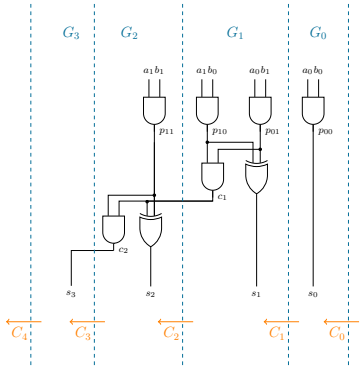
- Can we trust our own implementation?
- Is the verification process correct?

Solution:

Validate result of verification process.

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

Incremental Verification



[RitircBiereKauers FMCAD'17, KaufmannBiereKauers FMSD'19]

Let $P_k = \sum_{k=i+j} a_i b_j$.

Column-Wise Checking Algorithm

Input: Circuit C with sliced Gröbner bases G_i
Output: Determine whether C is a multiplier

$C_{2n} \leftarrow 0$

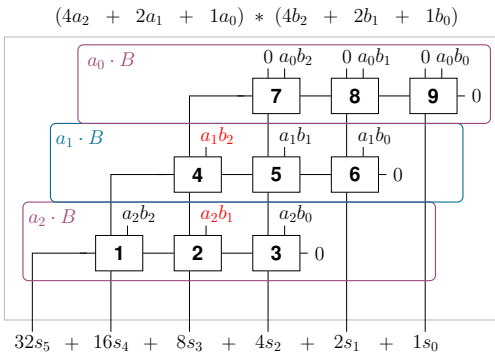
for $i \leftarrow 2n-1$ **to** 0

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

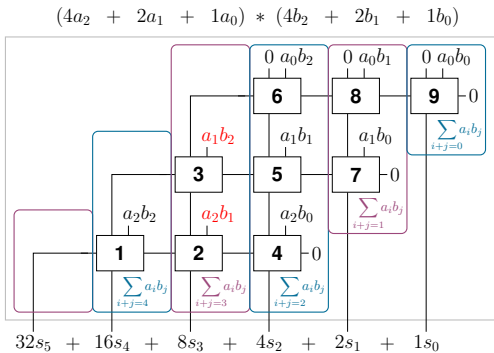
return $C_0 = 0$

Order

Row-Wise



Column-Wise



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 $C_0, \ldots, C_{2n}$ 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 Algorithm

Incremental Checking Algorithm.

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

$C_{2n} \leftarrow 0$

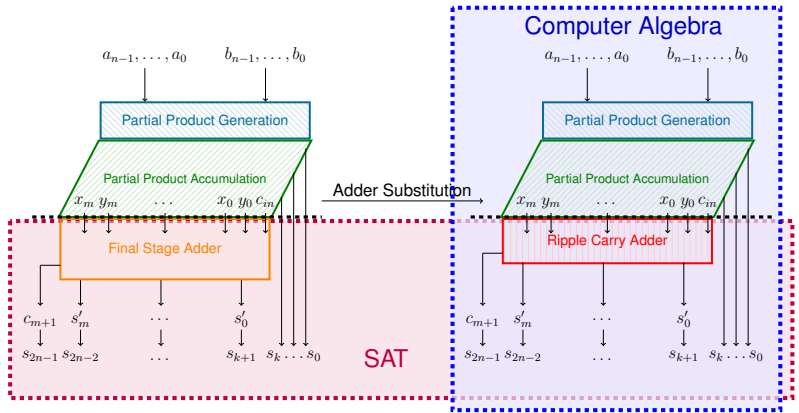
for $i \leftarrow 2n-1$ **to** 0

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

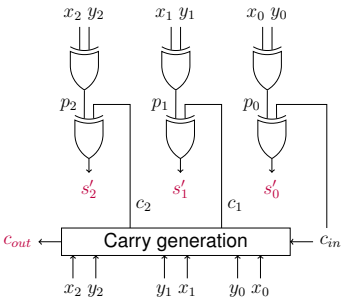
return $C_0 = 0$

SAT & Computer Algebra

[KaufmannBiereKauers FMCAD'19]

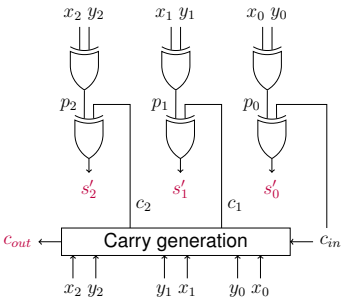


Adder Substitution



- $s'_i = p_i \oplus c_i$
- $p_i = x_i \oplus y_i$
- $c_i = (x_{i-1} \wedge y_{i-1}) \vee (c_{i-1} \wedge p_{i-1})$

Adder Substitution



- $s'_i = p_i \oplus c_i$
- $p_i = x_i \oplus y_i$
- $c_i = (x_{i-1} \wedge y_{i-1}) \vee (c_{i-1} \wedge p_{i-1})$

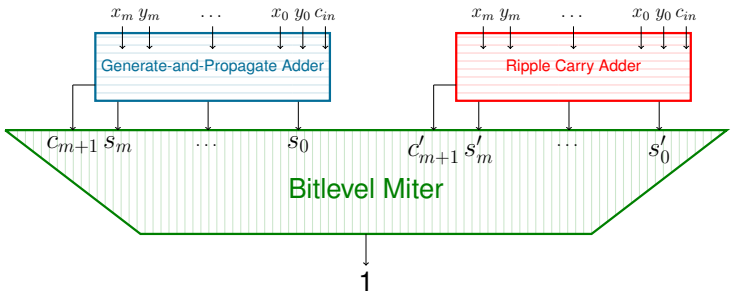
Algorithm: Identifying GP adders in AMULET

Input : Circuit C in AIG format

Output: Determine whether C might contain a GP adder

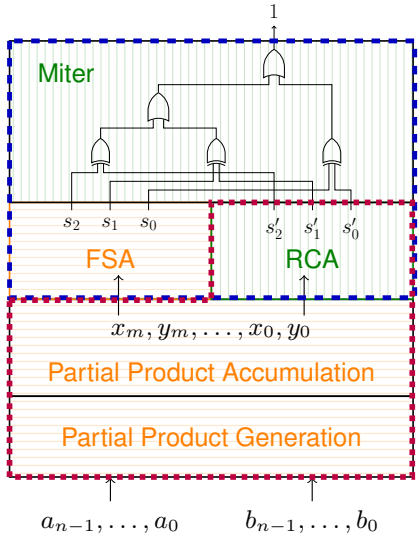
1 $j \leftarrow 2n-2$, $\tau \leftarrow 1$;
2 **while** τ **and** $j \geq 0$ **do**
3 $\quad \tau, c_j, p_j \leftarrow$ Check-if-XOR-and-Identify-p$_j$-and-c$_j$ (s_j);
4 $\quad x_j, y_j \leftarrow$Declare-Adder-Inputs (p_j, τ);
5 $\quad j \leftarrow j-1$;
6 $c_{in} \leftarrow c_j$;
7 **for** $i \leftarrow j$ **to** $2n-1$ **do**
8 $\quad m \leftarrow$ Follow-and-Mark-Paths(s_i);
9 **return** $m = 0$

SAT



- AIG is translated to CNF.
- CNF is given to SAT solver.
- To show correctness SAT solver needs to return UNSAT.

Generated Circuit



SAT

Computer algebra

References I

[ABC-based] M. J. Ciesielski, T. Su, A. Yasin and C. Yu. Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model. In IEEE TCAD, vol. 39, no. 6, pages 1346–1357, 2020.

[Biere SATComp'16] A. Biere. Collection of Combinational Arithmetic Miters Submitted to the SAT Competition 2016. In SAT Competition 2016, pages 65–66, Dep. of Computer Science Report Series B, University of Helsinki, 2016.

[Buchberger'65] B. Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD Thesis, University of Innsbruck, 1965.

[CleggEdmondsImpagliazzo STOC'96] M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability. In Proc. of STOC'96, pages 174–183, ACM, 1996.

[CiesielskiYuBrownLiuRossi DAC'15] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi. Verification of Gate-level Arithmetic Circuits by Function Extraction. In Proc. of DAC'15, pages 52:1–52:6, ACM, 2015.

[ChenBryant DAC'95] Y. Chen and R. Bryant. Verification of Arithmetic Circuits with Binary Moment Diagrams. In Proc. of DAC'95, pages 535–541, ACM, 1995.

References II

[KaufmannBiere TACAS'21] D. Kaufmann and A. Biere. AMulet 2.0 for Verifying Multiplier Circuits. Aceppted at TACAS'21, 2021.

[KaufmannBiereKauers FMCAD'19] D. Kaufmann, A. Biere and M. Kauers. Verifying Large Multipliers by Combining SAT and Computer Algebra. In Proc. of FMCAD'19, pages 28–36, IEEE, 2019.

[KaufmannBiereKauers FMSD'20] D. Kaufmann, A. Biere and M. Kauers. Incremental Column-Wise Verification of Arithmetic Circuits Using Computer Algebra. In FMSD, vol 56, pages 22–54, 2020.

[KaufmannBiereKauers Vampire'19] D. Kaufmann, A. Biere and M. Kauers. SAT, Computer Algebra, Multipliers. In Proc. of Vampire Workshop'19, EPiC series, vol. 71, pages 1–18, EasyChair, 2020.

[KaufmannFleuryBiere FMCAD'20] D. Kaufmann, M. Fleury and A. Biere. Pacheck and Pastèque Checking Practical Algebraic Calculus Proofs. In Proc. of FMCAD'20, pages 264–269, TU Vienna Academic Press, 2020.

[LvKallaEnescu TCAD'13] J. Lv, P. Kalla, F. Enescu. Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits. In IEEE TCAD, vol. 32, pages 1409–1420, 2013.

References III

[Mayr STACS'89] E. Mayr. Membership in polynomial ideals over Q is exponential space complete. In Proc. of STACS'89, pages 400–406, Springer, 1989

[MahzoonGroßeDrechsler ICCAD'18] A. Mahzoon, D. Große and R. Drechsler. PolyCleaner: Clean your Polynomials before Backward Rewriting to verify Million-gate Multipliers. In Proc. of ICCAD'18, pages 129:1–129:8, ACM, 2018.

[RevSCA] A. Mahzoon, D. Große and R. Drechsler. RevSCA: Using Reverse Engineering to Bring Light into Backwards Rewriting for Big and Dirty Multipliers. In Proc. of DAC'19, pages 185:1–185:6, ACM, 2019.

[RevSCA-2.0] A. Mahzoon, D. Große and R. Drechsler. RevSCA-2.0. `https://github.com/amahzoon/RevSCA-2.0`

[RitircBiereKauers DATE'18] D. Ritirc, A. Biere and M. Kauers. Improving and Extending the Algebraic Approach for Verifying Gate-Level Multipliers. In Proc. of DATE'18, pages 1556–1561, IEEE, 2018.

[RitircBiereKauers FMCAD'17] D. Ritirc, A. Biere and M. Kauers. Column-Wise Verification of Multipliers Using Computer Algebra. In Proc. of FMCAD'17, pages 23–30, IEEE, 2017.

References IV

[RitircBiereKauers SC2'18] D. Ritirc, A. Biere and M. Kauers. A Practical Polynomial Calculus for Arithmetic Circuit Verification. In Proc. of SC'2, pages 61–76, CEUR-WS, 2018.

[SayedGroßeKühneSoekenDrechsler DATE'16] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler. Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In Proc. of DATE'16, pages 1048–1053, IEEE, 2016.

[TemelSlobodovaHunt CAV'20] M. Temel, A. Slobodova and W. Hunt. Automated and Scalable Verification of Integer Multipliers. In Proc. of CAV'20, pages 485–507, Springer, 2020.