ADDING DUAL VARIABLES TO ALGEBRAIC REASONING FOR GATE-LEVEL MULTIPLIER VERIFICATION

Daniela Kaufmann[1] **Paul Beame**[2] **Armin Biere**[1,3] **Jakob Nordström**[4,5]
[1]Johannes Kepler University, Linz, Austria
[2]University of Washington, Seattle, WA, USA
[3]Albert-Ludwigs-University Freiburg, Germany
[4]University of Copenhagen, Denmark
[5]Lund University, Sweden

13th Pragmatics of SAT workshop
Haifa, Israel
August 1, 2022

✉ jn@di.ku.dk

ADDING DUAL VARIABLES TO ALGEBRAIC REASONING FOR GATE-LEVEL MULTIPLIER VERIFICATION

Daniela Kaufmann[1] **Paul Beame**[2] **Armin Biere**[1,3] **Jakob Nordström**[4,5]

[1]Johannes Kepler University, Linz, Austria

[2]University of Washington, Seattle, WA, USA

[3]Albert-Ludwigs-University Freiburg, Germany

[4]University of Copenhagen, Denmark

[5]Lund University, Sweden

13th Pragmatics of SAT workshop

Haifa, Israel

August 1, 2022

✉ `jn@di.ku.dk`

Thanks to Daniela for the slides!

Bugs in hardware are expensive!

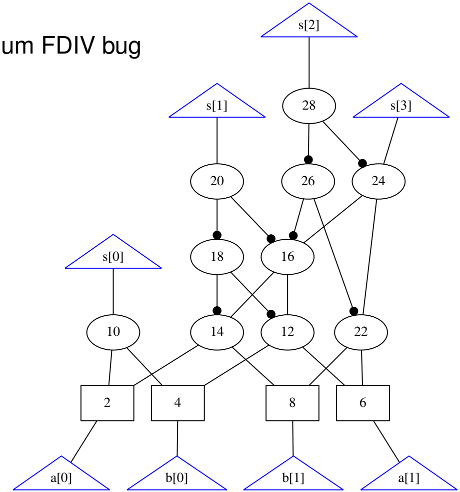
Circuit verification prevents issues like the famous Pentium FDIV bug

Multiplier verification

Given: Gate-level integer multiplier for fixed bit-width
Input format: AND-Inverter Graph

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

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



Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
- Exponential running time for solvers

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
- Exponential running time for solvers

Theorem Proving

- Used in industry
- Requires manual effort
- Automated techniques rely on hierarchical information

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
- Exponential running time for solvers

Decision Diagrams

- First technique to detect Pentium bug
- Rely on manual decomposition

Theorem Proving

- Used in industry
- Requires manual effort
- Automated techniques rely on hierarchical information

Formal Verification Techniques

Satisfiability Checking (SAT)

- SAT 2016 Competition
- Exponential running time for solvers

Decision Diagrams

- First technique to detect Pentium bug
- Rely on manual decomposition

Theorem Proving

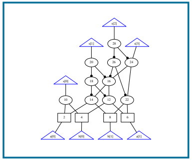
- Used in industry
- Requires manual effort
- Automated techniques rely on hierarchical information

Algebraic Approach

- Dramatic progress since 2015
- Polynomial encoding
- Automated approach
- 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

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

Ideal Membership

$= 0$ ✓

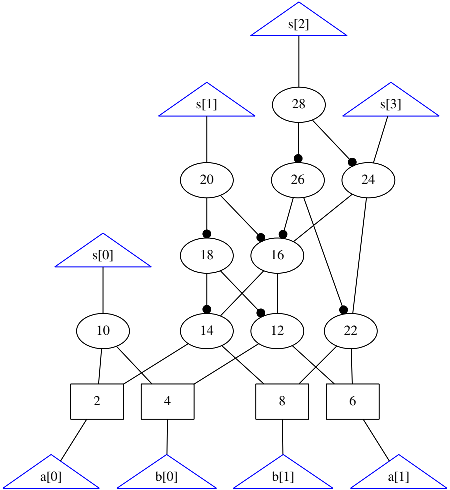
$\neq 0$ ✗

For more details on circuit verification using computer algebra, see, e.g., [Kaufmann, 2020]

From Circuits to Polynomials

Gate polynomials $G(C) \subseteq \mathbb{Z}[X]$

$$-s_3 + l_{24}$$
$$-s_2 + l_{28}$$
$$-s_1 + l_{20}$$
$$-s_0 + l_{10}$$
$$-l_{28} + l_{26}l_{24} - l_{26} - l_{24} + 1$$
$$-l_{26} + l_{22}l_{16} - l_{22} - l_{16} + 1$$
$$-l_{24} + l_{22}l_{16}$$
$$-l_{22} + a_1b_1$$
$$-l_{20} + l_{18}l_{16} - l_{18} - l_{16} + 1$$
$$-l_{18} + l_{14}l_{12} - l_{14} - l_{12} + 1$$
$$-l_{16} + l_{14}l_{12}$$
$$-l_{14} + a_0b_1$$
$$-l_{12} + a_1b_0$$
$$-l_{10} + a_0b_0$$



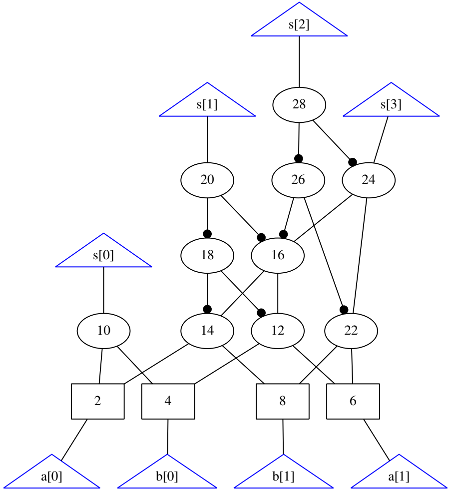
From Circuits to Polynomials

Gate polynomials $G(C) \subseteq \mathbb{Z}[X]$

$$
\begin{array}{ll}
-s_3 + l_{24} & -l_{22} + a_1 b_1 \\
-s_2 + l_{28} & -l_{20} + l_{18} l_{16} - l_{18} - l_{16} + 1 \\
-s_1 + l_{20} & -l_{18} + l_{14} l_{12} - l_{14} - l_{12} + 1 \\
-s_0 + l_{10} & -l_{16} + l_{14} l_{12} \\
-l_{28} + l_{26} l_{24} - l_{26} - l_{24} + 1 & -l_{14} + a_0 b_1 \\
-l_{26} + l_{22} l_{16} - l_{22} - l_{16} + 1 & -l_{12} + a_1 b_0 \\
-l_{24} + l_{22} l_{16} & -l_{10} + a_0 b_0
\end{array}
$$

Boolean axioms / value constraints $B(C) \subseteq \mathbb{Z}[X]$

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



From Circuits to Polynomials

Gate polynomials $G(C) \subseteq \mathbb{Z}[X]$

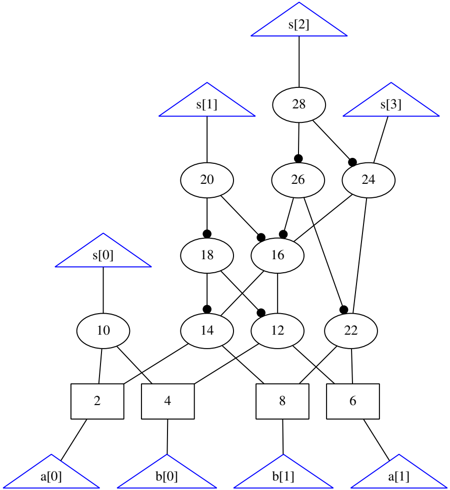
$-s_3 + l_{24}$	$-l_{22} + a_1 b_1$
$-s_2 + l_{28}$	$-l_{20} + l_{18} l_{16} - l_{18} - l_{16} + 1$
$-s_1 + l_{20}$	$-l_{18} + l_{14} l_{12} - l_{14} - l_{12} + 1$
$-s_0 + l_{10}$	$-l_{16} + l_{14} l_{12}$
$-l_{28} + l_{26} l_{24} - l_{26} - l_{24} + 1$	$-l_{14} + a_0 b_1$
$-l_{26} + l_{22} l_{16} - l_{22} - l_{16} + 1$	$-l_{12} + a_1 b_0$
$-l_{24} + l_{22} l_{16}$	$-l_{10} + a_0 b_0$

Boolean axioms / value constraints $B(C) \subseteq \mathbb{Z}[X]$

$$a_1, a_0 \in \mathbb{B} \qquad -a_1^2 + a_1,\ -a_0^2 + a_0,$$
$$b_1, b_0 \in \mathbb{B} \qquad -b_1^2 + b_1,\ -b_0^2 + b_0$$

Specification $\mathcal{S}_n \in \mathbb{Z}[X]$

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



Verification Technique

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 $G(C) \cup B(C)$

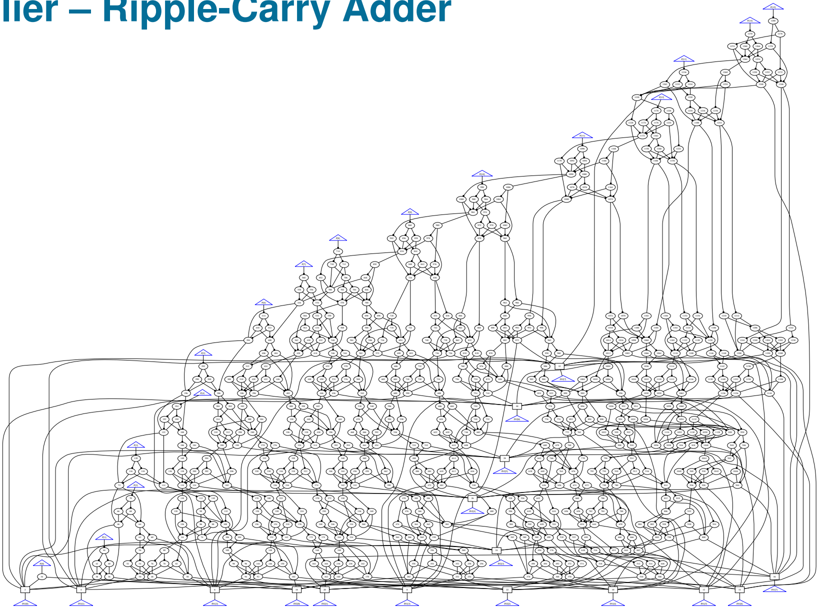
based on fixed variable order until no further reduction possible

Then: C is multiplier $\Leftrightarrow$ final remainder zero

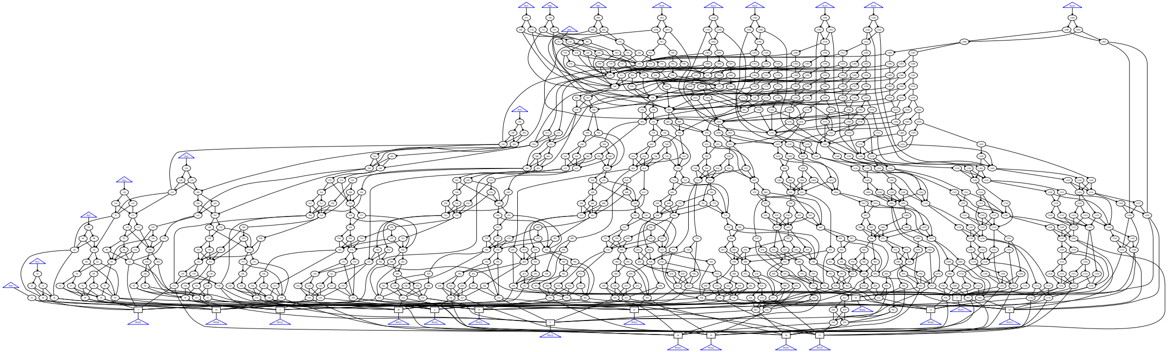
Easy: Multipliers containing a ripple-carry adder

Hard: Multipliers containing a generate-and-propagate adder, e.g., carry-lookahead adder

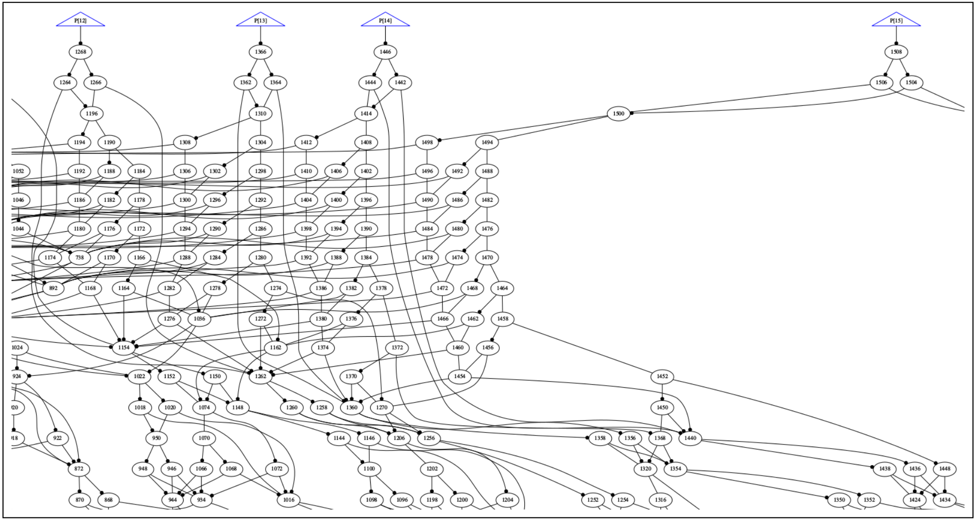
Multiplier – Ripple-Carry Adder



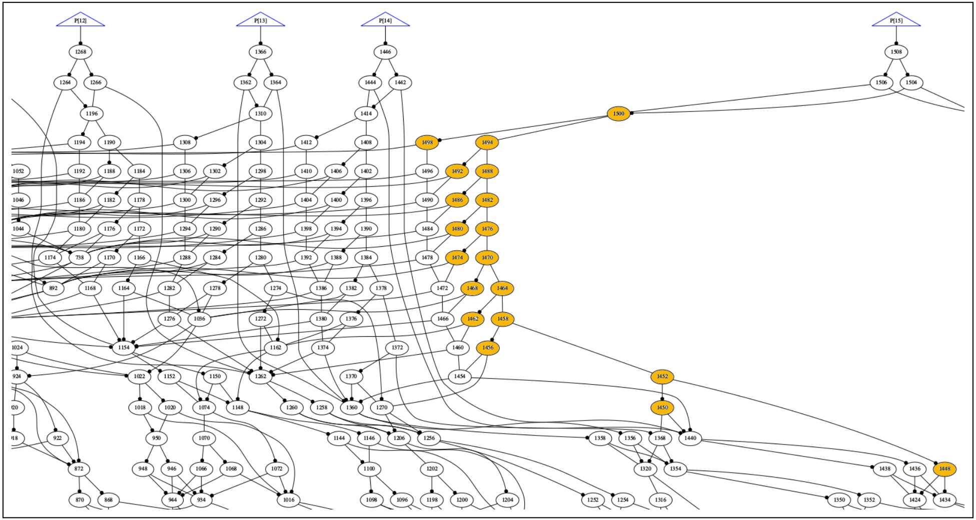
Multiplier – Carry-Lookahead Adder



Multiplier – Carry-Lookahead Adder

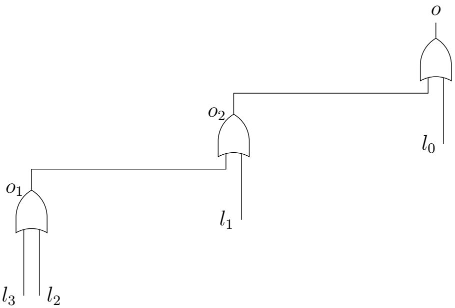


Multiplier – Carry-Lookahead Adder



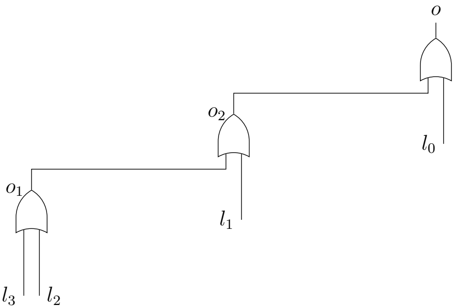
OR Gates

$$
\begin{aligned}
o &= o_2 \vee l_0 & -o + o_2 + l_0 - o_2 l_0, \\
o_2 &= o_1 \vee l_1 & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 &= l_3 \vee l_2 & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



OR Gates

$$
\begin{aligned}
o &= o_2 \vee l_0 & -o + o_2 + l_0 - o_2 l_0, \\
o_2 &= o_1 \vee l_1 & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 &= l_3 \vee l_2 & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$

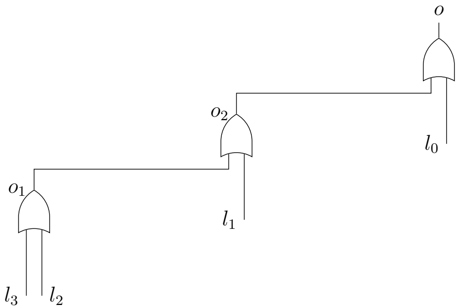


$$o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3$$

$15 = 2^4 - 1$ monomials

OR Gates

$$\begin{aligned} o = o_2 \vee l_0 \quad & -o + o_2 + l_0 - o_2 l_0, \\ o_2 = o_1 \vee l_1 \quad & -o_2 + o_1 + l_1 - o_1 l_1, \\ o_1 = l_3 \vee l_2 \quad & -o_1 + l_3 + l_2 - l_3 l_2 \end{aligned}$$



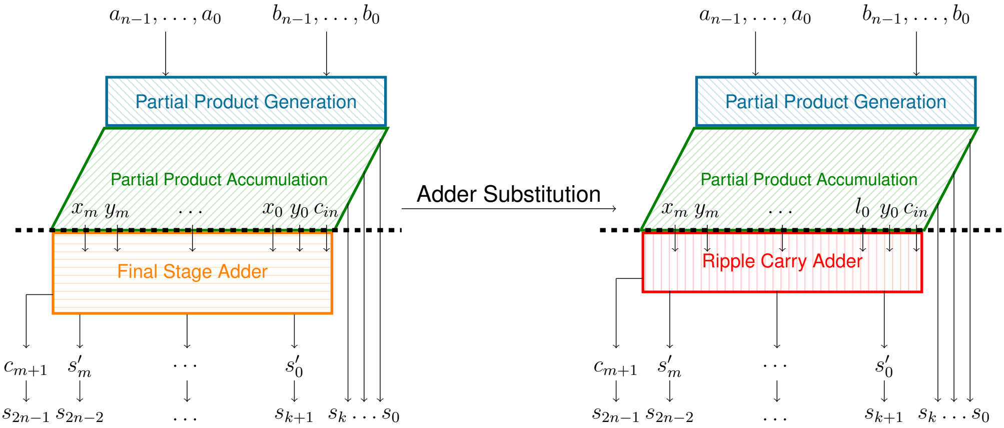
$$o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3$$

$15 = 2^4 - 1$ monomials

n OR Gates $\Rightarrow 2^{n+1} - 1$ monomials

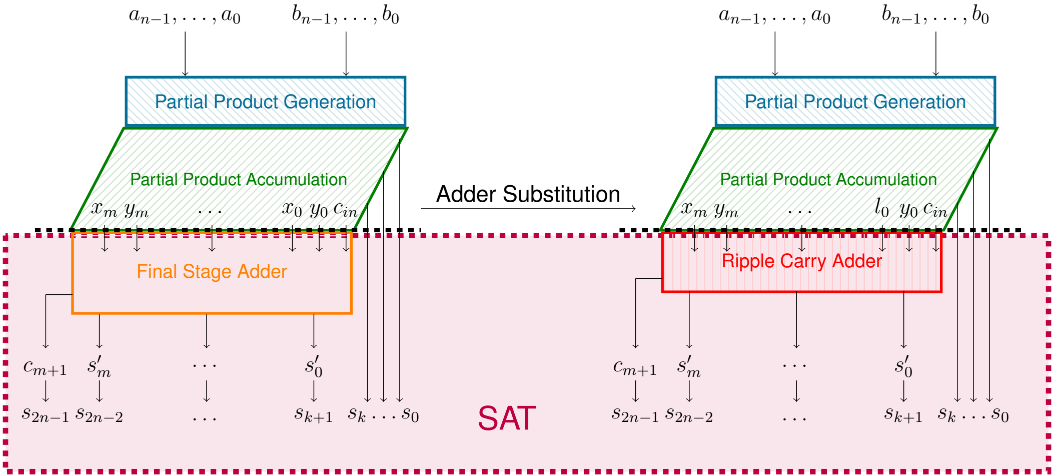
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]



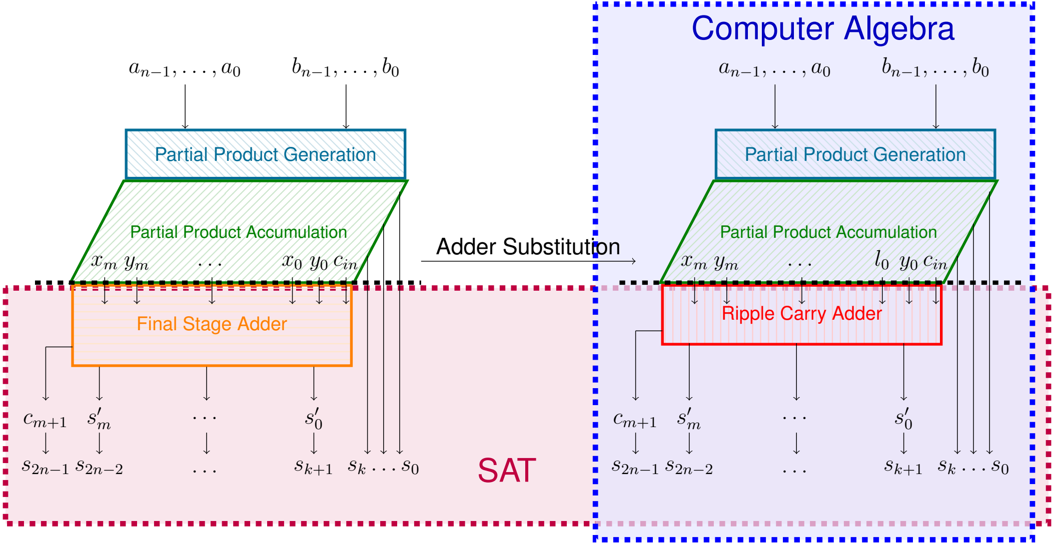
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]

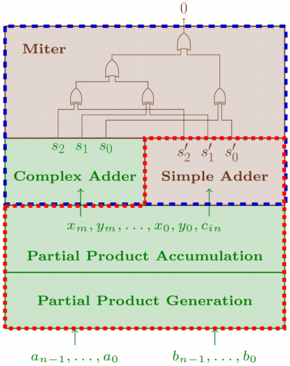


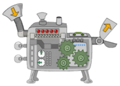
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]



Problem: Proof Certificates





SAT | **DRUP**

Computer Algebra | **Practical Algebraic Calculus**

Possible to simulate DRUP proofs in PAC, but does not scale [Kaufmann et al., 2020]

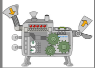
Contributions of Our *DATE ’22* Paper [Kaufmann et al., 2022]

Encoding

- Dual variables
- Compact representation of polynomials

Novel carry rewriting method

- Uses dual encoding
- Tail substitutions

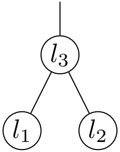


No need for SAT solver

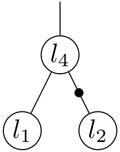
Uniform practical algebraic calculus (PAC) certificate

1st Contribution: Dual Variables

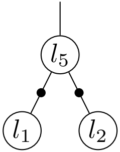
Provide more compact notation for inverters



$$l_3 = l_1 \wedge l_2$$
$$-l_3 + l_1 l_2$$



$$l_4 = l_1 \wedge \neg l_2$$
$$-l_4 - l_1 l_2 + l_1$$



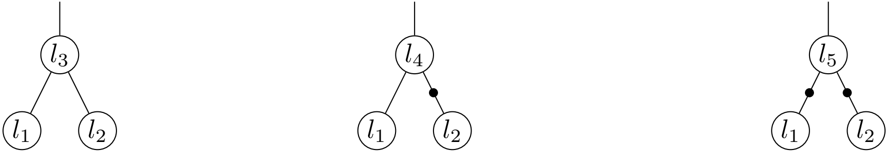
$$l_5 = \neg l_1 \wedge \neg l_2$$
$$-l_5 + l_1 l_2 - l_1 - l_2 + 1$$

1st Contribution: Dual Variables

Provide more compact notation for inverters

Dual variables

Whenever two variables $l_i, f_i \in \{0,1\}$ satisfy $f_i = 1 - l_i$, we have $f_i = \mathrm{dual}(l_i)$



$l_3 = l_1 \wedge l_2$	$l_4 = l_1 \wedge \neg l_2$	$l_5 = \neg l_1 \wedge \neg l_2$
$-l_3 + l_1 l_2$	$-l_4 - l_1 l_2 + l_1$	$-l_5 + l_1 l_2 - l_1 - l_2 + 1$
$-l_3 + l_1 l_2$	$-l_4 + l_1 f_2$	$-l_5 + f_1 f_2$

Detour: Proof Complexity

- Practical algebraic calculus well-studied under the name polynomial calculus [Clegg et al., 1996, Razborov, 1998, Impagliazzo et al., 1999, Buss et al., 2001, Alekhnovich and Razborov, 2003, Galesi and Lauria, 2010, Beck et al., 2013, Bonacina and Galesi, 2015, Mikša and Nordström, 2015] . . .

Detour: Proof Complexity

- Practical algebraic calculus well-studied under the name polynomial calculus [Clegg et al., 1996, Razborov, 1998, Impagliazzo et al., 1999, Buss et al., 2001, Alekhnovich and Razborov, 2003, Galesi and Lauria, 2010, Beck et al., 2013, Bonacina and Galesi, 2015, Mikša and Nordström, 2015] . . .
- Often with dual variables [Alekhnovich et al., 2000] to avoid annoying blow-up when encoding and reasoning with CNF formulas

Detour: Proof Complexity

- Practical algebraic calculus well-studied under the name polynomial calculus [Clegg et al., 1996, Razborov, 1998, Impagliazzo et al., 1999, Buss et al., 2001, Alekhnovich and Razborov, 2003, Galesi and Lauria, 2010, Beck et al., 2013, Bonacina and Galesi, 2015, Mikša and Nordström, 2015] . . .
- Often with dual variables [Alekhnovich et al., 2000] to avoid annoying blow-up when encoding and reasoning with CNF formulas
- Polynomial calculus without dual variables exponentially weaker [de Rezende et al., 2021]

Detour: Proof Complexity

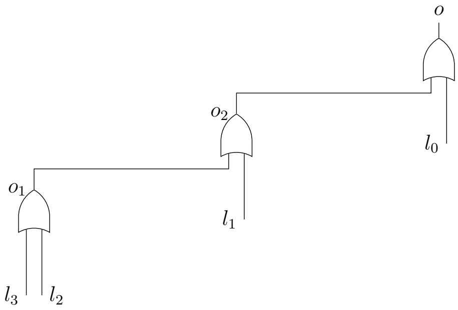
- Practical algebraic calculus well-studied under the name polynomial calculus [Clegg et al., 1996, Razborov, 1998, Impagliazzo et al., 1999, Buss et al., 2001, Alekhnovich and Razborov, 2003, Galesi and Lauria, 2010, Beck et al., 2013, Bonacina and Galesi, 2015, Mikša and Nordström, 2015] ...
- Often with dual variables [Alekhnovich et al., 2000] to avoid annoying blow-up when encoding and reasoning with CNF formulas
- Polynomial calculus without dual variables exponentially weaker [de Rezende et al., 2021]
- What we see here is **exactly this problem in practice**

Detour: Proof Complexity

- Practical algebraic calculus well-studied under the name polynomial calculus [Clegg et al., 1996, Razborov, 1998, Impagliazzo et al., 1999, Buss et al., 2001, Alekhnovich and Razborov, 2003, Galesi and Lauria, 2010, Beck et al., 2013, Bonacina and Galesi, 2015, Mikša and Nordström, 2015] . . .
- Often with dual variables [Alekhnovich et al., 2000] to avoid annoying blow-up when encoding and reasoning with CNF formulas
- Polynomial calculus without dual variables exponentially weaker [de Rezende et al., 2021]
- What we see here is **exactly this problem in practice**
- Theory suggests: **use dual variables!**

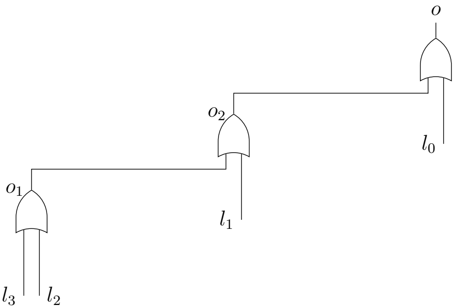
OR Gates

$$
\begin{aligned}
o &= o_2 \lor l_0 & -o + o_2 + l_0 - o_2 l_0, \\
o_2 &= o_1 \lor l_1 & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 &= l_3 \lor l_2 & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



OR Gates

$$
\begin{aligned}
o &= o_2 \lor l_0 & -o + o_2 + l_0 - o_2 l_0, \\
o_2 &= o_1 \lor l_1 & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 &= l_3 \lor l_2 & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



$$o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3$$

$$o = 1 - f_0 f_1 f_2 f_3$$

Practical Difficulty

Key Method for polynomial inference: Gröbner basis algorithm

Relies on reduction method based on fixed variable order that will immediately eliminate one of each pair of dual variables

Practice: During verification, always reduce specification by the dual constraint $-f_i - l_i + 1$ of a gate variable l_i before reducing by its gate constraint
Has the effect that all occurrences of f_i in the specification will be flipped to l_i before reducing l_i

Problem: Compact representation is unfolded

Practical Difficulty

Key Method for polynomial inference: Gröbner basis algorithm

Relies on reduction method based on fixed variable order that will immediately eliminate one of each pair of dual variables

Practice: During verification, always reduce specification by the dual constraint $-f_i - l_i + 1$ of a gate variable l_i before reducing by its gate constraint
Has the effect that all occurrences of f_i in the specification will be flipped to l_i before reducing l_i

Problem: Compact representation is unfolded

$\Rightarrow$ Need dedicated preprocessing techniques to keep compact representation

Calculating with Dual Variables

Proposition 1.
For all Boolean variables l_i and their dual representation $\mathrm{dual}(l_i) = f_i$ we have $l_i f_i = 0$

"l_i and $\mathrm{dual}(l_i)$ cannot be 1 at the same time"

Proposition 2.
For all Boolean variables l_i and their dual representation $\mathrm{dual}(l_i) = f_i$ we have $l_i + f_i = 1$

"l_i and $\mathrm{dual}(l_i)$ add up to 1"

Dual Mergeable

Call m_1 and m_2 **dual mergeable** iff $m_1 = cf_i\tau$ and $m_2 = cl_i\tau$ for c constant, τ term
Call monomial $\mathrm{dmerge}(m_1, m_2) = c\tau$ their **dual merge**

Dual Mergeable

Call m_1 and m_2 **dual mergeable** iff $m_1 = c f_i \tau$ and $m_2 = c l_i \tau$ for c constant, τ term

Call monomial $\mathrm{dmerge}(m_1, m_2) = c\tau$ their **dual merge**

Algorithm: Merging monomials(p)

Input : Polynomial p

Output: Simplified polynomial r

$q \leftarrow$ sort-degree-lex(p); $r \leftarrow 0$;
while $q \neq 0$ **do**
    $q_l \leftarrow \mathrm{lm}(q)$; $t \leftarrow \mathrm{tail}(q)$; simplify $\leftarrow \bot$;
    while $t \neq 0$ and $\deg(q_l) = \deg(\mathrm{lt}(t))$ and $\neg$ simplify **do**
        $q_t \leftarrow \mathrm{lt}(t)$;
        if q_l and q_t are dual mergeable **then**
            $q \leftarrow q - q_l - q_t + \mathrm{dmerge}(q_l, q_t)$;
            simplify $\leftarrow \top$;
        else $t \leftarrow t - q_t$;
    if $\neg$ simplify **then** $r \leftarrow r + q_l$; $q \leftarrow q - q_l$;
return sort-lex(r);

Example of Dual Mergeable Monomials

Example

Let $p = l_1 f_2 f_3 + l_1 f_2 l_3 + l_1 l_2 f_3 + f_1 f_2 + l_2 \in \mathbb{Z}[l_1, l_2, l_3, f_1, f_2, f_3]$

Write q_i to denote polynomial q after iteration i (dual merges indicated)

$$
\begin{array}{llll}
q_0 &= l_1 f_2 f_3 + l_1 f_2 l_3 + l_1 l_2 f_3 + f_1 f_2 + l_2 & & r = 0 \\
q_1 &= l_1 l_2 f_3 + f_1 f_2 + \boxed{l_1 f_2} + l_2 & & r = 0 \\
q_2 &= f_1 f_2 + l_1 f_2 + l_2 & & r = l_1 l_2 f_3 \\
q_3 &= \boxed{f_2} + l_2 & & r = l_1 l_2 f_3 \\
q_4 &= \boxed{1} & & r = l_1 l_2 f_3 \\
q_5 &= 0 & & r = l_1 l_2 f_3 + 1
\end{array}
$$

2nd Contribution: Tail Substitution

Allows to introduce sharing on larger topological levels

Consider $p = f - g$ and $p_1, \ldots p_6$ in $\mathbb{Z}[X]$:

$$\begin{aligned} p_1 &:= -f + h_1 h_2 & p_2 &:= -g + h_3 h_4 g_0 g_5 \\ p_3 &:= -h_1 + g_0 g_1 g_2 & p_4 &:= -h_3 + g_1 g_2 \\ p_5 &:= -h_2 + g_3 g_4 g_5 & p_6 &:= -h_4 + g_3 g_4 \end{aligned}$$

2nd Contribution: Tail Substitution

Allows to introduce sharing on larger topological levels

Consider $p = f - g$ and $p_1, \dots p_6$ in $\mathbb{Z}[X]$:

$$\begin{aligned} p_1 &:= -f + h_1 h_2 & p_2 &:= -g + h_3 h_4 g_0 g_5 \\ p_3 &:= -h_1 + g_0 g_1 g_2 & p_4 &:= -h_3 + g_1 g_2 \\ p_5 &:= -h_2 + g_3 g_4 g_5 & p_6 &:= -h_4 + g_3 g_4 \end{aligned}$$

Have to reduce p by polynomials $p_1, \dots, p_6$ to obtain $p = 0$:

2nd Contribution: Tail Substitution

Allows to introduce sharing on larger topological levels

Consider $p = f - g$ and $p_1, \ldots p_6$ in $\mathbb{Z}[X]$:

$$\begin{aligned} p_1 &:= -f + h_1h_2 & p_2 &:= -g + h_3h_4g_0g_5 \\ p_3 &:= -h_1 + g_0g_1g_2 & p_4 &:= -h_3 + g_1g_2 \\ p_5 &:= -h_2 + g_3g_4g_5 & p_6 &:= -h_4 + g_3g_4 \end{aligned}$$

Have to reduce p by polynomials $p_1, \ldots, p_6$ to obtain $p = 0$:

$$\begin{aligned} f - g &\xrightarrow{p_1} \\ h_1h_2 - g &\xrightarrow{p_2} \\ h_1h_2 - h_3h_4g_0g_5 &\xrightarrow{p_3} \\ g_0g_1g_2h_2 - h_3h_4g_0g_5 &\xrightarrow{p_4} \\ g_0g_1g_2h_2 - g_1g_2h_4g_0g_5 &\xrightarrow{p_5} \\ g_0g_1g_2g_3g_4g_5 - g_1g_2h_4g_0g_5 &\xrightarrow{p_6} \\ g_0g_1g_2g_3g_4g_5 - g_0g_1g_2g_3g_4g_5 &= 0 \end{aligned}$$

Tail Substitution in Action

$$
\begin{array}{ll}
p_1 := -f + h_1 h_2 & p_2 := -g + h_3 h_4 g_0 g_5 \\
p_3 := -h_1 + g_0 g_1 g_2 & p_4 := -h_3 + g_1 g_2 \\
p_5 := -h_2 + g_3 g_4 g_5 & p_6 := -h_4 + g_3 g_4
\end{array}
$$

Reduce $p = f - g$ by polynomials $p_1, \ldots, p_6$ to obtain $p = 0$

Tail Substitution in Action

$$p_1 := -f + h_1h_2 \qquad p_2 := -g + h_3h_4g_0g_5$$
$$p_3 := -h_1 + g_0g_1g_2 \qquad p_4 := -h_3 + g_1g_2$$
$$p_5 := -h_2 + g_3g_4g_5 \qquad p_6 := -h_4 + g_3g_4$$

Reduce $p = f - g$ by polynomials $p_1, \ldots, p_6$ to obtain $p = 0$

Since $\mathrm{tail}(p_4) \mid \mathrm{tail}(p_3)$ and $\mathrm{tail}(p_6) \mid \mathrm{tail}(p_5)$, we can derive:

$$p_3 := -h_1 + h_3g_0 \qquad p_5 := -h_2 + h_4g_5$$

Tail Substitution in Action

$$p_1 := -f + h_1 h_2 \qquad p_2 := -g + h_3 h_4 g_0 g_5$$
$$p_3 := -h_1 + g_0 g_1 g_2 \qquad p_4 := -h_3 + g_1 g_2$$
$$p_5 := -h_2 + g_3 g_4 g_5 \qquad p_6 := -h_4 + g_3 g_4$$

Reduce $p = f - g$ by polynomials $p_1, \ldots, p_6$ to obtain $p = 0$

Since $\mathrm{tail}(p_4) \mid \mathrm{tail}(p_3)$ and $\mathrm{tail}(p_6) \mid \mathrm{tail}(p_5)$, we can derive:

$$p_3 := -h_1 + h_3 g_0 \qquad p_5 := -h_2 + h_4 g_5$$

Then substitute tails of p_3, p_5 in p_2:

$$p_1 := -f + h_1 h_2 \qquad p_2 := -g + h_1 h_2$$

Tail Substitution in Action

$$p_1 := -f + h_1 h_2 \qquad p_2 := -g + h_3 h_4 g_0 g_5$$
$$p_3 := -h_1 + g_0 g_1 g_2 \qquad p_4 := -h_3 + g_1 g_2$$
$$p_5 := -h_2 + g_3 g_4 g_5 \qquad p_6 := -h_4 + g_3 g_4$$

Reduce $p = f - g$ by polynomials $p_1, \ldots, p_6$ to obtain $p = 0$

Since $\mathrm{tail}(p_4) \mid \mathrm{tail}(p_3)$ and $\mathrm{tail}(p_6) \mid \mathrm{tail}(p_5)$, we can derive:

$$p_3 := -h_1 + h_3 g_0 \qquad p_5 := -h_2 + h_4 g_5$$

Then substitute tails of p_3, p_5 in p_2:

$$p_1 := -f + h_1 h_2 \qquad p_2 := -g + h_1 h_2$$

Hence we have to reduce p only by p_1 and p_2 to derive $p = 0$
Somewhat reminiscent of degree-bounded Gröbner basis reduction in [Clegg et al., 1996]

Carry Rewriting

Goal: Rewrite encoding of carry look-ahead unit into a ripple-carry unit, which can easily be verified using computer algebra

Algorithm: Carry-Rewriting

Input : Circuit C in AIG format

Output: Carry-rewritten Gröbner basis of C

$F \leftarrow$ Mark-final-stage-adder(C);
$G \leftarrow$ Dual-Polynomial-Encoding(F);
$H \leftarrow$ Polynomial-Encoding($C \setminus F$);
$G \leftarrow$ Eliminate-Pure-Positive-Variables(G);
$G \leftarrow$ Tail-Substitution(G);
$G \leftarrow$ Carry-Unfolding(G);
return $G \cup H$

Carry Unfolding

Proposition 3.
Let $-l_i + \sigma\tau_i$ for $1 \leq i \leq k$ be a given set of polynomials, with $l_i \in X$ and $\sigma, \tau_i \in [X]$. Assume $\forall_{i=0}^{k} f_i = \mathrm{dual}(l_i)$. Then $\prod_{i=0}^{k} f_i = 1 - \sigma(1 - \prod_{i=0}^{k}(1 - \tau_i))$.

Example

Excerpt of carry-lookahead adder, with x_i, y_i being the ith inputs of the adder, c_{i+1}, c_i denoting carries, and p_i being the polynomial encoding of $x_i \oplus y_i$:

$$\begin{array}{lll} -c_{i+1} + f_4 f_5 f_6 f_7, & -c_i + f_1 f_2 f_3, & -l_7 + x_i y_i, \\ -l_6 + p_i l_3, & -l_5 + p_i l_2, & -l_4 + p_i l_1 \end{array}$$

Using carry unfolding for c_{i+1}, we are able to derive

$$-c_{i+1} + f_7 p_i c_i - f_7 p_i + f_7, \quad -c_i + f_1 f_2 f_3 \quad -l_7 + x_i y_i$$

TeluMA

- Integration of dual variables into AMULET 2.0 [Kaufmann et al., 2019]
- Identifies final-stage adders
- Applies carry rewriting automatically
- On-the-fly generation of proof certificates in PAC format

Published version and experimental data available at:

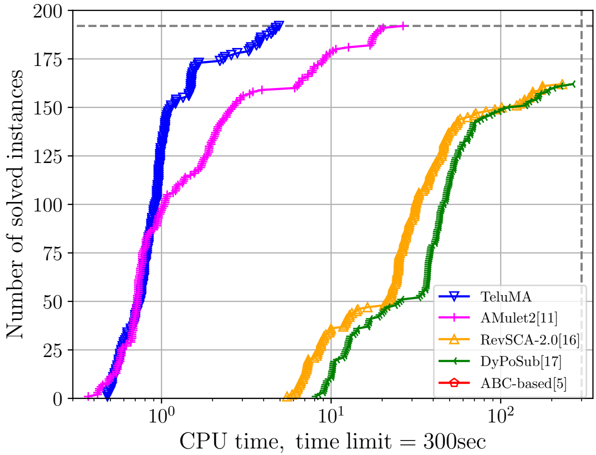
`http://fmv.jku.at/teluma`

Maintained version available at:

`https://github.com/d-kfmnn/teluma`

Evaluation: Multiplier Verification

Verification of 192 unsigned 64-bit multipliers



Evaluation: Proof Certificates

architecture	n	[Kaufmann et al., 2019]			[Kaufmann et al., 2020]		Our approach	
		DRUP	PAC	total (s)	PAC	total (s)	PAC	total (s)
		# rules	# rules		# rules		# rules	
sp-ar-cl	32	14 927	33 834	1	1 597 897	164	60 336	0
sp-bd-ks	32	17 528	34 958	1	817 956	28	54 116	0
sp-dt-lf	32	3 138	33 451	1	321 720	5	47 835	0
bp-ct-bk	32	2 276	27 312	1	217 128	3	36 356	0
bp-wt-cl	32	46 502	30 561	2	5 536 176	3 375	114 665	2
sp-ar-cl	64	65 317	139 338	8	-	TO	289 632	4
sp-bd-ks	64	44 921	142 138	6	1 440 943	74	214 378	3
sp-dt-lf	64	28 772	138 539	6	816 572	19	192 805	2
bp-ct-bk	64	19 891	105 579	5	459 262	15	136 703	2
bp-wt-cl	64	42 199	118 573	19	-	TO	774 044	24

All benchmarks generated by Arithmetic Model Generator [Homma et al., 2006]

TO = 3600 sec

Conclusion & Future Work

Contributions:

- Inclusion of dual variables
- Novel tail substitution scheme
- Carry rewriting technique

Results:

- Speed-up in verification of complex multiplier circuits
- Uniform PAC proof certificate

Future directions:

- Generalization to more general circuit verification
- Gröbner basis algorithm with dual variables?!
- Pseudo-Boolean solving for circuit verification? [Liew et al., 2020]
- More cross-fertilization between theory and practice!

Conclusion & Future Work

Contributions:

- Inclusion of dual variables
- Novel tail substitution scheme
- Carry rewriting technique

Results:

- Speed-up in verification of complex multiplier circuits
- Uniform PAC proof certificate

Future directions:

- Generalization to more general circuit verification
- Gröbner basis algorithm with dual variables?!
- Pseudo-Boolean solving for circuit verification? [Liew et al., 2020]
- More cross-fertilization between theory and practice!

Thank you for your attention!

References I

[Alekhnovich et al., 2000] Alekhnovich, M., Ben-Sasson, E., Razborov, A. A., and Wigderson, A. (2000).
Space complexity in propositional calculus.
In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 358–367.

[Alekhnovich and Razborov, 2003] Alekhnovich, M. and Razborov, A. A. (2003).
Lower bounds for polynomial calculus: Non-binomial case.
Proceedings of the Steklov Institute of Mathematics, 242:18–35.

[Beck et al., 2013] Beck, C., Nordström, J., and Tang, B. (2013).
Some trade-off results for polynomial calculus.
In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822.

[Bonacina and Galesi, 2015] Bonacina, I. and Galesi, N. (2015).
A framework for space complexity in algebraic proof systems.
Journal of the ACM, 62(3):23:1–23:20.

References II

[Buss et al., 2001] Buss, S. R., Grigoriev, D., Impagliazzo, R., and Pitassi, T. (2001).
Linear gaps between degrees for the polynomial calculus modulo distinct primes.
Journal of Computer and System Sciences, 62(2):267–289.

[Clegg et al., 1996] Clegg, M., Edmonds, J., and Impagliazzo, R. (1996).
Using the Groebner basis algorithm to find proofs of unsatisfiability.
In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183.

[de Rezende et al., 2021] de Rezende, S. F., Lauria, M., Nordström, J., and Sokolov, D. (2021).
The power of negative reasoning.
In *Proceedings of the 36th Annual Computational Complexity Conference (CCC '21)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:24.

[Galesi and Lauria, 2010] Galesi, N. and Lauria, M. (2010).
Optimality of size-degree trade-offs for polynomial calculus.
ACM Transactions on Computational Logic, 12(1):4:1–4:22.

References III

[Homma et al., 2006] Homma, N., Watanabe, Y., Aoki, T., and Higuchi, T. (2006).
Formal Design of Arithmetic Circuits Based on Arithmetic Description Language.
IEICE Transactions, 89-A(12):3500–3509.

[Impagliazzo et al., 1999] Impagliazzo, R., Pudlák, P., and Sgall, J. (1999).
Lower bounds for the polynomial calculus and the Gröbner basis algorithm.
Computational Complexity, 8(2):127–144.

[Kaufmann, 2020] Kaufmann, D. (2020).
Formal Verification of Multiplier Circuits using Computer Algebra.
PhD thesis, Informatik, Johannes Kepler University Linz.

[Kaufmann et al., 2022] Kaufmann, D., Beame, P., Biere, A., and Nordström, J. (2022).
Adding dual variables to algebraic reasoning for circuit verification.
In *Proceedings of the 25th Design, Automation and Test in Europe Conference (DATE '22)*, pages 1435–1440.

References IV

[Kaufmann et al., 2019] Kaufmann, D., Biere, A., and Kauers, M. (2019).

Verifying large multipliers by combining SAT and computer algebra.

In *Proceedings of the 19th Conference on Formal Methods in Computer-Aided Design (FMCAD '19)*, pages 28–36.

[Kaufmann et al., 2020] Kaufmann, D., Biere, A., and Kauers, M. (2020).

From DRUP to PAC and back.

In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE '20)*, pages 654–657.

[Liew et al., 2020] Liew, V., Beame, P., Devriendt, J., Elffers, J., and Nordström, J. (2020).

Verifying properties of bit-vector multiplication using cutting planes reasoning.

In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 194–204.

References V

[Mikša and Nordström, 2015] Mikša, M. and Nordström, J. (2015).

A generalized method for proving polynomial calculus degree lower bounds.

In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 467–487.

[Razborov, 1998] Razborov, A. A. (1998).

Lower bounds for the polynomial calculus.

Computational Complexity, 7(4):291–324.