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

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

[1]Johannes Kepler University, Linz, Austria
[2]University of Washington, Seattle, WA, USA
[3]Albert-Ludwigs-University Freiburg, Germany
[4]University of Copenhagen, Denmark & Lund University, Sweden

Design, Automation and Test in Europe Conference, 2022
Antwerp, Belgium & online

March 2022

✉ daniela.kaufmann@jku.at

Bugs in hardware are expensive!

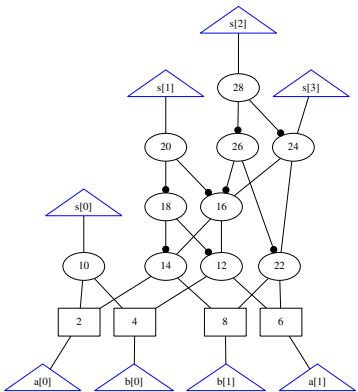
Circuit verification prevents issues like the famous Pentium FDIV bug.

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 run-time of solvers

Decision Diagrams

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

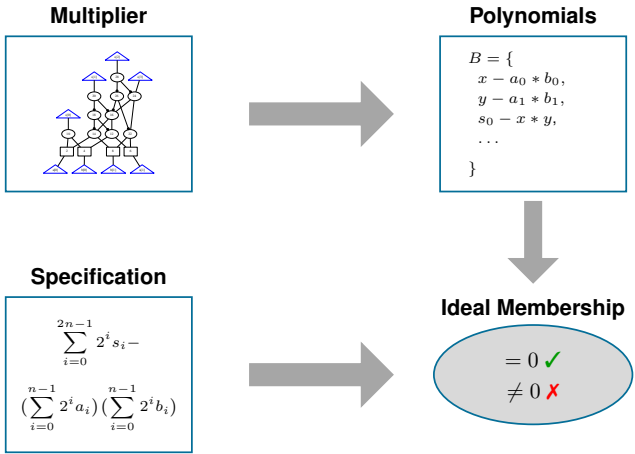
Theorem Proving

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

Algebraic Approach

- Great progress since 2015
- Polynomial encoding
- Works for non-trivial multiplier designs

Basic Idea of Algebraic Approach



From Circuits to Polynomials

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

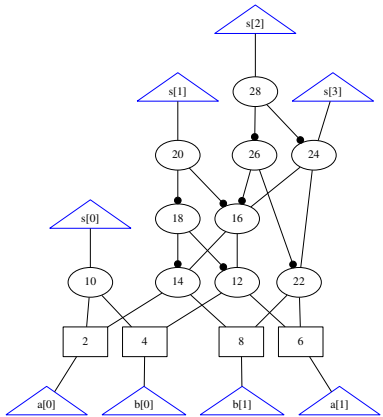
$$
\begin{array}{ll}
-s_3 + l_{24} & -l_{22} + a_1 b_1 \\
-s_2 + l_{28} & -l_{20} + l_{18} l_{16} - l_{18} - l_{16} + 1 \\
-s_1 + l_{20} & -l_{18} + l_{14} l_{12} - l_{14} - l_{12} + 1 \\
-s_0 + l_{10} & -l_{16} + l_{14} l_{12} \\
-l_{28} + l_{26} l_{24} - l_{26} - l_{24} + 1 & -l_{14} + a_0 b_1 \\
-l_{26}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

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

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

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

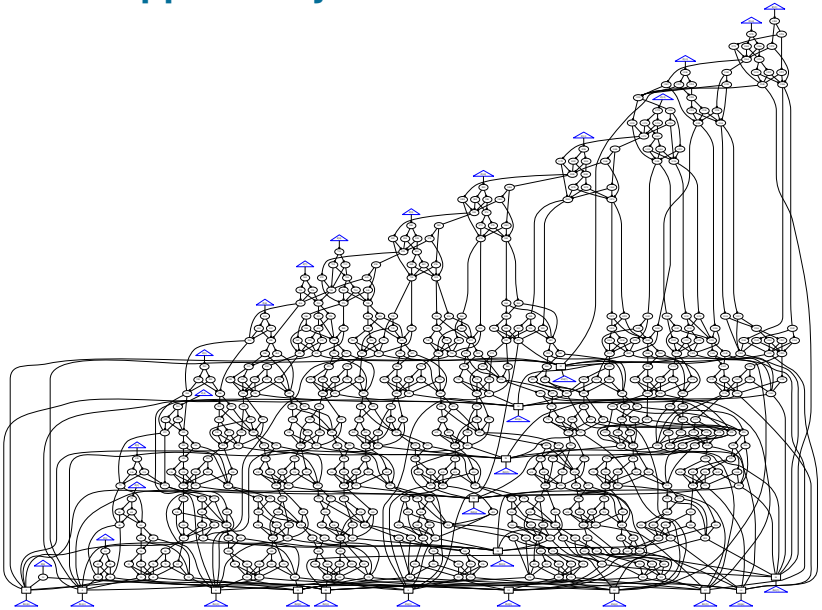
based on a fixed variable order until no further reduction is possible.

Then C is a multiplier iff the final remainder is zero.

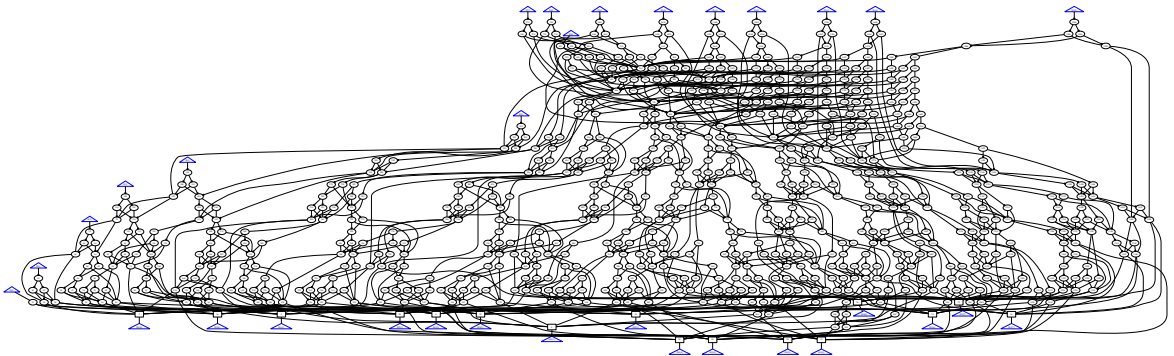
Easy: Multipliers containing a ripple-carry adder

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

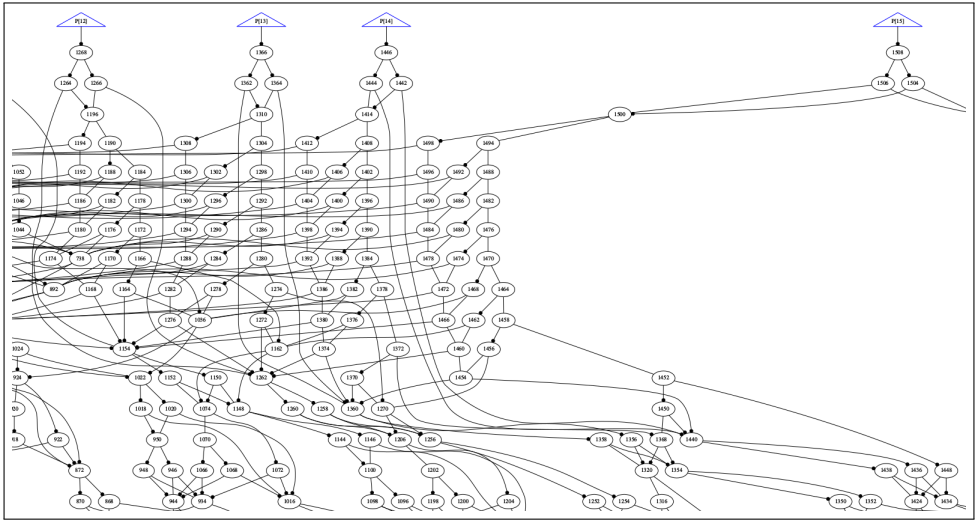
Multiplier – Ripple-Carry Adder



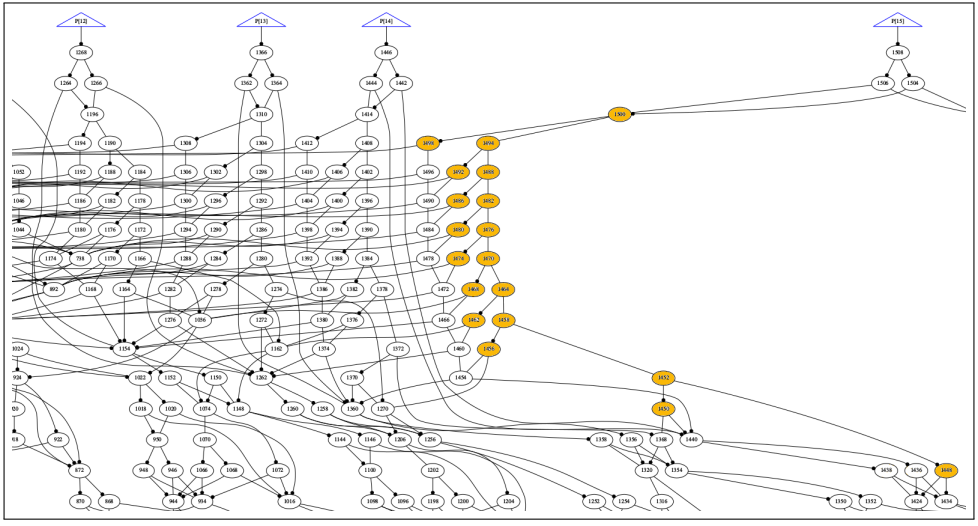
Multiplier – Carry-Lookahead Adder



Multiplier – Carry-Lookahead Adder

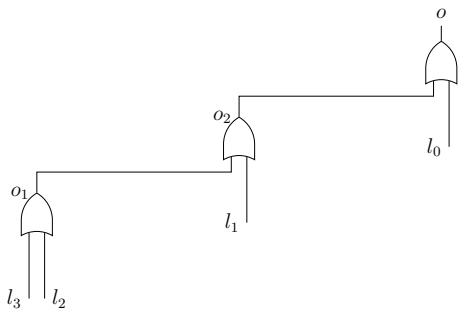


Multiplier – Carry-Lookahead Adder



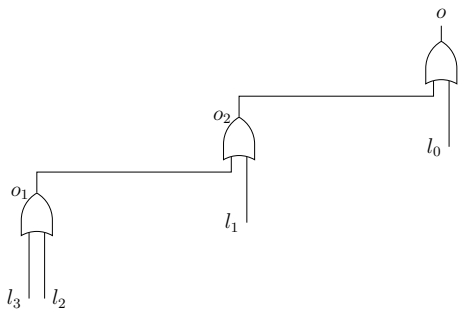
OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + l_0 - o_2 l_0, \\
o_2 = o_1 \vee l_1 \quad & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 = l_3 \vee l_2 \quad & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + l_0 - o_2 l_0, \\
o_2 = o_1 \vee l_1 \quad & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 = l_3 \vee l_2 \quad & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



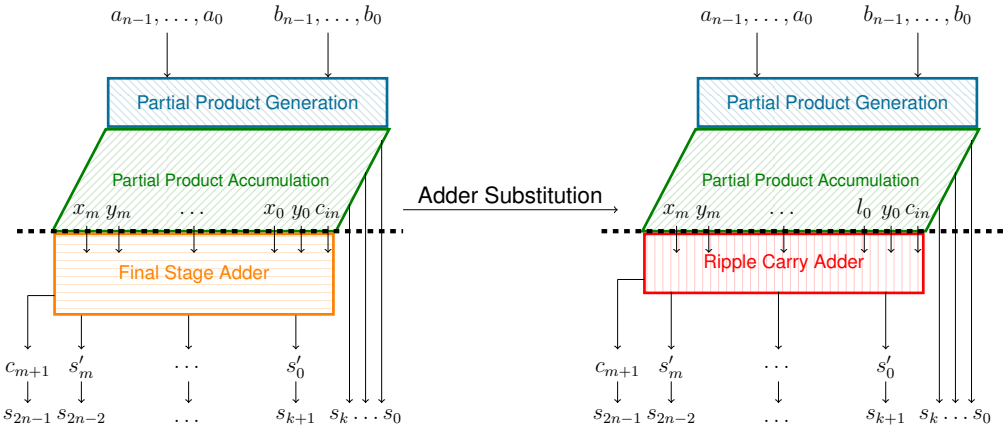
$$o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3$$

$15 = 2^4 - 1$ monomials

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

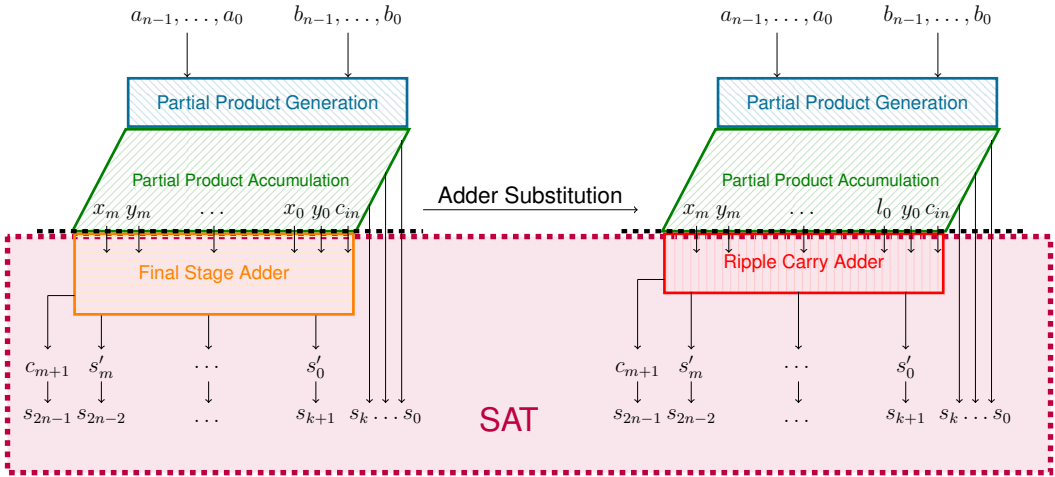
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]



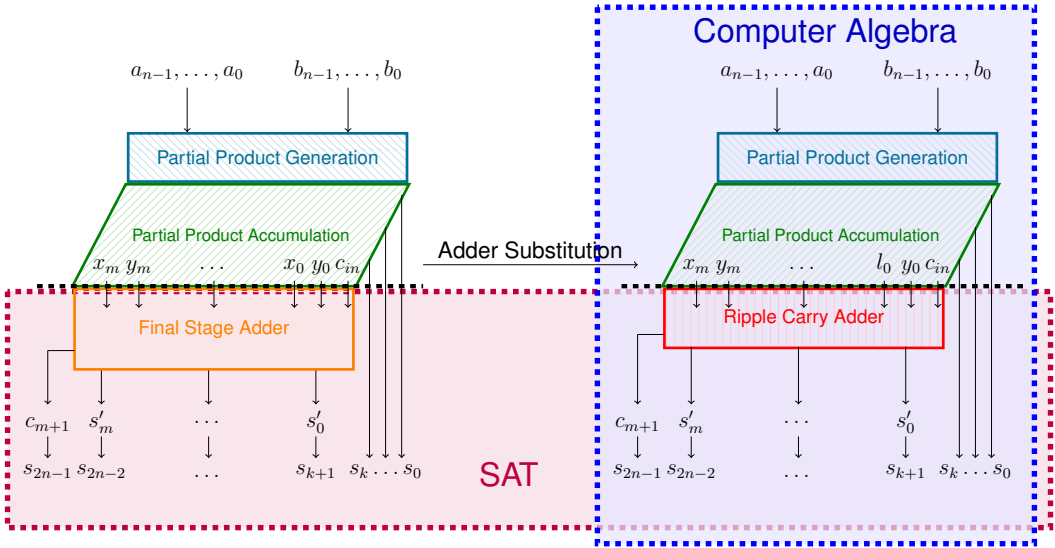
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]

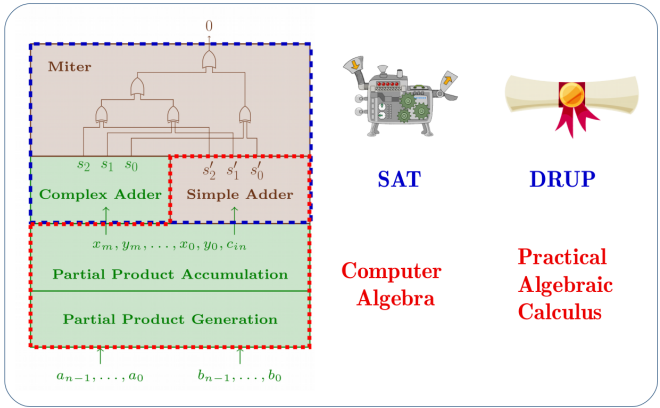


Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]



Problem: Proof Certificates



It is possible to simulate DRUP proofs in PAC, but it does not scale [Kaufmann et al., 2020].

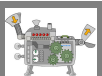
Contributions

Encoding

- Dual variables
- Compact representation of polynomials

Novel carry rewriting method

- Uses dual encoding
- Tail substitutions

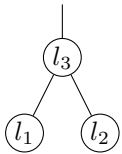


Eliminates necessity of a SAT solver

Uniform PAC certificate

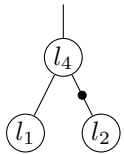
Dual Variables

Provide a shorthand notation for inverters.



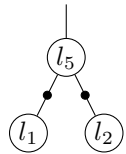
$$l_3 = l_1 \wedge l_2$$

$$-l_3 + l_1 l_2$$



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

$$-l_4 - l_1 l_2 + l_1$$



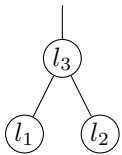
$$l_5 = \neg l_1 \wedge \neg l_2$$

$$-l_5 + l_1 l_2 - l_1 - l_2 + 1$$

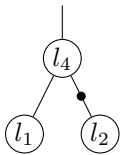
Dual Variables

Provide a shorthand notation for inverters.

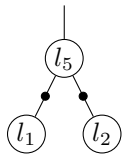
Dual variables.
Whenever two variables $l_i, f_i \in \{0, 1\}$ fulfill the relation $f_i = 1 - l_i$, we have $f_i = \mathrm{dual}(l_i)$.



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



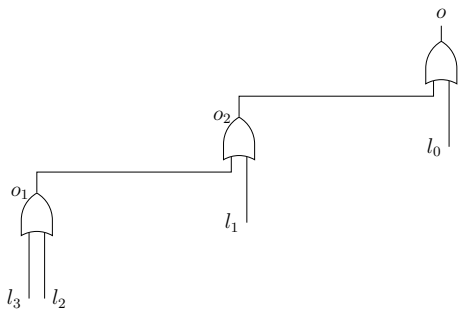
$$l_4 = l_1 \wedge \neg l_2$$
$$-l_4 - l_1 l_2 + l_1$$
$$-l_4 + l_1 f_2$$



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

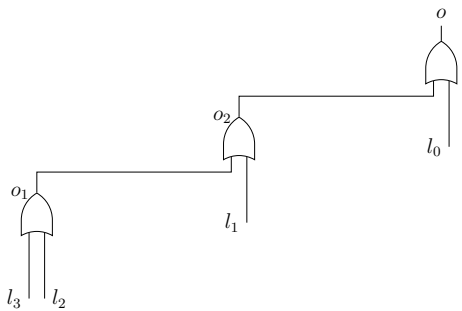
OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + l_0 - o_2 l_0, \\
o_2 = o_1 \vee l_1 \quad & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 = l_3 \vee l_2 \quad & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



OR Gates

$$
\begin{aligned}
o = o_2 \vee x_0 \quad & -o + o_2 + l_0 - o_2 l_0, \\
o_2 = o_1 \vee l_1 \quad & -o_2 + o_1 + l_1 - o_1 l_1, \\
o_1 = l_3 \vee l_2 \quad & -o_1 + l_3 + l_2 - l_3 l_2
\end{aligned}
$$



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

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

Practical Difficulty

Key Method for polynomial inference: Gröbner basis algorithm

Relies on a reduction method based on a fixed variable order that will immediately eliminate one of each pair of dual variables by re-expressing it using its partner.

Practice: During verification we always reduce the specification by the dual constraint $-f_i - l_i + 1$ of a gate variable l_i before reducing by its gate constraint. This 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 a reduction method based on a fixed variable order that will immediately eliminate one of each pair of dual variables by re-expressing it using its partner.

Practice: During verification we always reduce the specification by the dual constraint $-f_i - l_i + 1$ of a gate variable l_i before reducing by its gate constraint. This 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.

$\rightarrow$ We need dedicated preprocessing techniques to keep compact representation.

Calculate 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

We call two monomials m_1 and m_2 **dual mergeable** iff $m_1 = cf_i\tau$ and $m_2 = cl_i\tau$ for c a constant, τ a term, and some index i. We call the 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);

Dual Mergeable

Example

Let $p = l_1 f_2 f_3 + l_1 f_2 l_3 + l_1 l_2 f_3 + f_1 f_2 + l_2 \in \mathbb{Z}[l_1, l_2, l_3, f_1, f_2, f_3]$. We write q_i to denote the polynomial q after iteration i and indicate the dual merges.

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

Tail Substitution

Allows to introduce sharing on larger topological levels.

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

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

We have to reduce p by polynomials $p_1, \dots, p_6$ to obtain $p = 0$.

$$\begin{aligned} f - g &\xrightarrow{p_1} \\ h_1 h_2 - g &\xrightarrow{p_2} \\ h_1 h_2 - h_3 h_4 g_0 g_5 &\xrightarrow{p_3} \\ g_0 g_1 g_2 h_2 - h_3 h_4 g_0 g_5 &\xrightarrow{p_4} \\ g_0 g_1 g_2 h_2 - g_1 g_2 h_4 g_0 g_5 &\xrightarrow{p_5} \\ g_0 g_1 g_2 g_3 g_4 g_5 - g_1 g_2 h_4 g_0 g_5 &\xrightarrow{p_6} \\ g_0 g_1 g_2 g_3 g_4 g_5 - g_0 g_1 g_2 g_3 g_4 g_5 &= 0 \end{aligned}$$

Tail Substitution

Allows to introduce sharing on larger topological levels.

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

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

We see that $\mathrm{tail}(p_4) \mid \mathrm{tail}(p_3)$ and $\mathrm{tail}(p_6) \mid \mathrm{tail}(p_5)$ and thus are able to derive:

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

In a second iteration we substitute the tails of p_3, p_5 in p_2:

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

Hence we have to reduce p only by p_1 and p_2 to derive $p = 0$.

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

1 $F \leftarrow$ Mark-final-stage-adder(C);
2 $G \leftarrow$ Dual-Polynomial-Encoding(F);
3 $H \leftarrow$ Polynomial-Encoding($C \setminus F$);
4 $G \leftarrow$ Eliminate-Pure-Positive-Variables(G);
5 $G \leftarrow$ Tail-Substitution(G);
6 $G \leftarrow$ Carry-Unfolding(G);
7 **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

The following polynomials are an excerpt of a carry-lookahead adder, with x_i, y_i being the i-th 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:

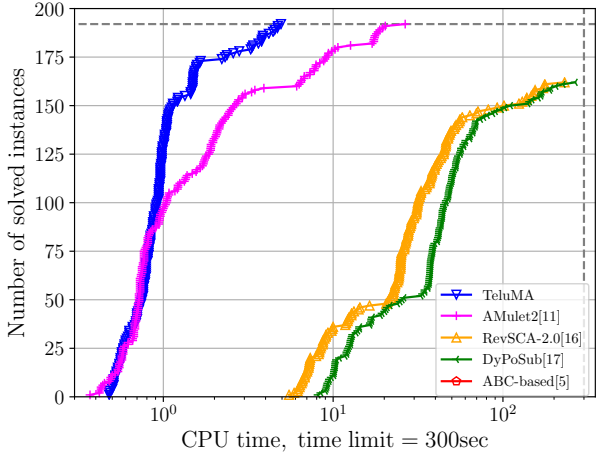
`http://fmv.jku.at/teluma`

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 are generated by the Arithmetic Model Generator [Homma et al., 2006]. TO = 3600 sec

Conclusion & Future Work

Conclusion.

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

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

Outlook.

- Generalize techniques to more general circuit verification.
- Determine minimal representations.

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

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

[1]Johannes Kepler University, Linz, Austria
[2]University of Washington, Seattle, WA, USA
[3]Albert-Ludwigs-University Freiburg, Germany
[4]University of Copenhagen, Denmark & Lund University, Sweden

Design, Automation and Test in Europe Conference, 2022
Antwerp, Belgium & online

March 2022

✉ daniela.kaufmann@jku.at

References I

[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.

[Kaufmann et al., 2019] Kaufmann, D., Biere, A., and Kauers, M. (2019).
Verifying Large Multipliers by Combining SAT and Computer Algebra.
In *FMCAD 2019*, pages 28–36. IEEE.

[Kaufmann et al., 2020] Kaufmann, D., Biere, A., and Kauers, M. (2020).
From DRUP to PAC and back.
In *DATE 2020*, pages 654–657. IEEE.