Daniela Kaufmann\textsuperscript{1}  Paul Beame\textsuperscript{2}  Armin Biere\textsuperscript{1,3}  Jakob Nordström\textsuperscript{4}  \\
\textsuperscript{1}Johannes Kepler University, Linz, Austria  \\
\textsuperscript{2}University of Washington, Seattle, WA, USA  \\
\textsuperscript{3}Albert-Ludwigs-University Freiburg, Germany  \\
\textsuperscript{4}University of Copenhagen, Denmark & Lund University, Sweden  \\

\textbf{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 $a_i, b_i \in \mathbb{B}$:

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

Multiplier

Polynomials

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

Specification

Ideal Membership

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

$$= 0 \text{ ✓}$$

$$\neq 0 \times$$
From Circuits to Polynomials

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

\[-s_3 + l_{24} \quad -s_2 + l_{28} \quad -s_1 + l_{20} \quad -s_0 + l_{10} \quad -l_{28} + l_{26}l_{24} - l_{26} - l_{24} + 1 \quad -l_{26} + l_{22}l_{16} - l_{22} - l_{16} + 1 \quad -l_{24} + l_{22}l_{16}\]

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

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

Specification $S_n \in \mathbb{Z}[X]$.

\[8s_3 + 4s_2 + 2s_1 + s_0 - 4b_1a_1 - 2b_1a_0 - 2b_0a_1 - b_0a_0\]
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 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
OR Gates

\[ o = o_2 \lor x_0 \quad \neg o + o_2 + l_0 - o_2 l_0, \]
\[ o_2 = o_1 \lor l_1 \quad \neg o_2 + o_1 + l_1 - o_1 l_1, \]
\[ o_1 = l_3 \lor l_2 \quad \neg o_1 + l_3 + l_2 - l_3 l_2 \]
OR Gates

\[ o = o_2 \lor x_0 \quad -o + o_2 + l_0 - o_2 l_0, \]
\[ o_2 = o_1 \lor l_1 \quad -o_2 + o_1 + l_1 - o_1 l_1, \]
\[ o_1 = l_3 \lor l_2 \quad -o_1 + l_3 + l_2 - l_3 l_2 \]

\[ o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3 \]

15 = 2^4 − 1 monomials

\( n \) OR Gates \( \rightarrow 2^{n+1} - 1 \) monomials
Previous Approach: SAT & Computer Algebra

[Kaufmann et al., 2019]

Partial Product Generation

Partial Product Accumulation

Final Stage Adder

Adder Substitution

Ripple Carry Adder
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.

\[ l_3 = l_1 \land l_2 \]
\[ -l_3 + l_1 l_2 \]

\[ l_4 = l_1 \land \neg l_2 \]
\[ -l_4 - l_1 l_2 + l_1 \]

\[ l_5 = \neg l_1 \land \neg l_2 \]
\[ -l_5 + l_1 l_2 - l_1 - l_2 + 1 \]
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 $f_i = \text{dual}(l_i)$.

\[
\begin{align*}
l_3 &= l_1 \land l_2 \\
-l_3 + l_1 l_2 \\
-l_3 + l_1 l_2
\end{align*}
\]

\[
\begin{align*}
l_4 &= l_1 \land \neg l_2 \\
-l_4 - l_1 l_2 + l_1 \\
-l_4 + l_1 f_2
\end{align*}
\]

\[
\begin{align*}
l_5 &= \neg l_1 \land \neg l_2 \\
-l_5 + l_1 l_2 - l_1 - l_2 + 1 \\
-l_5 + f_1 f_2
\end{align*}
\]
OR Gates

\[ o = o_2 \lor x_0 \quad \neg o + o_2 + l_0 - o_2 l_0, \]
\[ o_2 = o_1 \lor l_1 \quad \neg o_2 + o_1 + l_1 - o_1 l_1, \]
\[ o_1 = l_3 \lor l_2 \quad \neg o_1 + l_3 + l_2 - l_3 l_2 \]
OR Gates

\[ o = o_2 \lor x_0 \quad \Rightarrow -o + o_2 + l_0 - o_2 l_0, \]
\[ o_2 = o_1 \lor l_1 \quad \Rightarrow -o_2 + o_1 + l_1 - o_1 l_1, \]
\[ o_1 = l_3 \lor l_2 \quad \Rightarrow -o_1 + l_3 + l_2 - l_3 l_2 \]

\[ o = l_0 + l_1 - l_0 l_1 + l_2 - l_0 l_2 - l_1 l_2 + l_0 l_1 l_2 + l_3 - l_0 l_3 - l_1 l_3 + l_0 l_1 l_3 - l_2 l_3 + l_0 l_2 l_3 + l_1 l_2 l_3 - l_0 l_1 l_2 l_3 \]

\[ o = 1 - f_0 f_1 f_2 f_3 \]
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 $\text{dual}(l_i) = f_i$ we have $l_i f_i = 0$.

“$l_i$ and $\text{dual}(l_i)$ cannot be 1 at the same time.”

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

“$l_i$ and $\text{dual}(l_i)$ 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, \( \tau \) a term, and some index \( i \). We call the monomial \( \text{dmerge}(m_1, m_2) = c\tau \) their **dual merge**.

**Algorithm:** Merging monomials(\( p \))

**Input** : Polynomial \( p \)

**Output**: Simplified polynomial \( r \)

1. \( q \leftarrow \text{sort-degree-lex}(p); \ r \leftarrow 0; \)
2. **while** \( q \neq 0 \) **do**
   3. \( q_l \leftarrow \text{lm}(q); \ t \leftarrow \text{tail}(q); \ \text{simplify} \leftarrow \bot; \)
   4. **while** \( t \neq 0 \) and \( \deg(q_l) = \deg(\text{lt}(t)) \) and \( \neg \text{simplify} \) **do**
      5. \( q_t \leftarrow \text{lt}(t); \)
      6. **if** \( q_l \) and \( q_t \) are dual mergeable **then**
         7. \( q \leftarrow q - q_l - q_t + \text{dmerge}(q_l, q_t); \)
         8. \( \text{simplify} \leftarrow \top; \)
      9. **else** \( t \leftarrow t - q_t; \)
   10. **if** \( \neg \text{simplify} \) **then** \( r \leftarrow r + q_l, q \leftarrow q - q_l; \)
11. **return** \( \text{sort-lex}(r); \)
## 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.

<table>
<thead>
<tr>
<th>Iteration $i$</th>
<th>Polynomial $q_i$</th>
<th>Dual Merge</th>
<th>Remainder $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$l_1 f_2 f_3 + l_1 f_2 l_3 + l_1 l_2 f_3 + f_1 f_2 + l_2$</td>
<td>$r = 0$</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$l_1 l_2 f_3 + f_1 f_2 + l_1 f_2 l_3$</td>
<td>$r = 0$</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>$f_1 f_2 + l_1 f_2 + l_2$</td>
<td>$r = l_1 l_2 f_3$</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>$f_2 + l_2$</td>
<td>$r = l_1 l_2 f_3$</td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>$1$</td>
<td>$r = l_1 l_2 f_3$</td>
<td></td>
</tr>
<tr>
<td>$q_5$</td>
<td>$0$</td>
<td>$r = l_1 l_2 f_3 + 1$</td>
<td></td>
</tr>
</tbody>
</table>
Tail Substitution

Allows to introduce sharing on larger topological levels.

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

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

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

\[
\begin{align*}
f - g & \xrightarrow{p_1} \\
h_1 h_2 & \xrightarrow{p_2} \\
h_1 h_2 - h_3 h_4 g_0 g_5 & \xrightarrow{p_3} \\
g_0 g_1 g_2 h_2 & \xrightarrow{p_4} \\
g_0 g_1 g_2 h_2 - g_1 g_2 h_4 g_0 g_5 & \xrightarrow{p_5} \\
g_0 g_1 g_2 g_3 g_4 g_5 & \xrightarrow{p_6} \\
g_0 g_1 g_2 g_3 g_4 g_5 - g_0 g_1 g_2 g_3 g_4 g_5 & = 0
\end{align*}
\]
Tail Substitution

Allows to introduce sharing on larger topological levels.

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

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

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

\[
\begin{align*}
p_3 &:= -h_1 + h_3 g_0, & p_5 &:= -h_2 + h_4 g_5
\end{align*}
\]

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

\[
\begin{align*}
p_1 &:= -f + h_1 h_2, & p_2 &:= -g + h_1 h_2
\end{align*}
\]

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. $F \leftarrow \text{Mark-final-stage-adder}(C)$;
2. $G \leftarrow \text{Dual-Polynomial-Encoding}(F)$;
3. $H \leftarrow \text{Polynomial-Encoding}(C \setminus F)$;
4. $G \leftarrow \text{Eliminate-Pure-Positive-Variables}(G)$;
5. $G \leftarrow \text{Tail-Substitution}(G)$;
6. $G \leftarrow \text{Carry-Unfolding}(G)$;
7. return $G \cup H$
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 $\forall_{i=0}^{k} f_i = \text{dual}(l_i)$. 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 $x_i \oplus y_i$:

\[-c_{i+1} + f_4 f_5 f_6 f_7, \quad -c_i + f_1 f_2 f_3, \quad -l_7 + x_i y_i,
-l_6 + p_i l_3, \quad -l_5 + p_i l_2, \quad -l_4 + p_i l_1\]

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

![Graph showing number of solved instances vs CPU time]

- TeluMA
- AMulet2
- RevSCA-2.0
- DyPoSub
- ABC-based

CPU time, time limit = 300sec
### Evaluation - Proof Certificates

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

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\textsuperscript{1}  Paul Beame\textsuperscript{2}  Armin Biere\textsuperscript{1,3}  Jakob Nordström\textsuperscript{4}
\textsuperscript{1}Johannes Kepler University, Linz, Austria
\textsuperscript{2}University of Washington, Seattle, WA, USA
\textsuperscript{3}Albert-Ludwigs-University Freiburg, Germany
\textsuperscript{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

