Challenges in
Verifying Arithmetic Circuits Using Computer Algebra

Armin Biere
joint work with Daniela Ritirc and Manuel Kauers

SYNASC’17
19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

Timișoara, Romania
September 28, 2017
Equivalence Checking

- compare low-level optimized versus high-level golden circuit

- reasons:
  - complex synthesis tool flow
  - engineering change order (manual optimizations)

- considered first successful industrial formal method

- since mid 90’ties
  - BDD sweeping
  - SAT sweeping

- combinational & sequential
  - co-NP versus PSPACE
Binary Multiplication

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
\cdot & 1 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
1 & 2 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}
\]

\[15 \cdot 13 = 195\]
Multipliers

Diagram representing multipliers with nodes labeled HA and FA, and edges indicating connections.
Commutativity of Bit-Vector Multiplication

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

<table>
<thead>
<tr>
<th>bits</th>
<th>Glucose</th>
<th>Lingeling</th>
<th>cube-and-conquer</th>
<th>Treengeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>05</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.13</td>
</tr>
<tr>
<td>06</td>
<td>0.02</td>
<td>0.03</td>
<td>0.36</td>
<td>0.31</td>
</tr>
<tr>
<td>07</td>
<td>0.14</td>
<td>0.27</td>
<td>0.63</td>
<td>0.72</td>
</tr>
<tr>
<td>08</td>
<td>1.18</td>
<td>1.98</td>
<td>1.38</td>
<td>2.47</td>
</tr>
<tr>
<td>09</td>
<td>7.85</td>
<td>10.98</td>
<td>2.63</td>
<td>4.65</td>
</tr>
<tr>
<td>10</td>
<td>37.16</td>
<td>41.49</td>
<td>5.02</td>
<td>10.86</td>
</tr>
<tr>
<td>11</td>
<td>147.62</td>
<td>214.98</td>
<td>15.72</td>
<td>21.96</td>
</tr>
<tr>
<td>12</td>
<td>833.62</td>
<td>649.49</td>
<td>56.57</td>
<td>61.48</td>
</tr>
<tr>
<td>13</td>
<td>--</td>
<td>--</td>
<td>238.10</td>
<td>263.44</td>
</tr>
</tbody>
</table>

limit of 900 seconds wall clock time

http://fmv.jku.at/datapath
Related Work


Full-Adder and Half-Adder Gate Polynomials

\[ g_0 = a \oplus b \]
\[ g_1 = a \land b \]
\[ g_2 = c \land g_0 \]
\[ s = c \oplus g_0 \]
\[ o = g_1 \lor g_2 \]

\[ -g_0 + a + b - 2ab \]
\[ -g_1 + ab \]
\[ -g_2 + cg_0 \]
\[ -s + c + g_0 - 2cg_0 \]
\[ -o + g_1 + g_2 - 8182 \]

\[ -2o - s + a + b + c \]
\[ -2o - s + a + b \]
Definition 1. Let $C$ be an acyclic circuit with an $n$-bit multiplier signature, e.g.,

- inputs $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}$, outputs $s_0, \ldots, s_{2n-1}$, internal gates $g_1, \ldots, g_k$

$$X = a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, g_1, \ldots, g_k, s_0, \ldots, s_{2n-1}$$

- polynomial $p \in \mathbb{Q}[X]$ is a polynomial circuit constraint (PCC) for $C$ 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 gates of $C$ substitution of these values into $p$ gives zero.

- The set of all PCCs for $C$ is denoted by $I(C)$.

Definition 2. Let $G$ be the set of gate polynomials of $C$ (as in the example) joined with the set of input field polynomials $x(x-1)$ for inputs $x$.

Fix a topological order over $X$, with gate outputs larger than gate inputs. Let $J(C) = \langle G \rangle$.

Theorem 1. $G$ is a Gröbner basis.

Theorem 2 (Soundness and Completeness). $J(C) = I(C)$. 
Definition 3. A circuit $C$ as in Def. 1 is called a *multiplier* if

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

Algorithm 1.

Reduce polynomial in Def. 3 with $G$, then $C$ is a multiplier iff remainder vanishes.
ring R = 0, ( 
  s(3),
  s(2),
    Xor_2_2, 
    And_2_1, 
    And_2_0, 
  s(1),
    Xor_1_3, 
    And_1_2, 
    And_1_1, 
    And_1_0, 
  s(0),
    And_0_0, 
  b(1),
  b(0),
  b(0),
  a(1),
  a(0),
), lp;
ideal I0 = 
   -And_0_0 + a(0) * b(0),
   -s(0) + And_0_0

;

ideal I1 = 
   -And_1_0 + b(0) * a(1),
   -And_1_1 + a(0) * b(1),
   -And_1_2 + And_1_0 * And_1_1,
   -Xor_1_3 + And_1_0 + And_1_1 - 2 * And_1_0 * And_1_1,
   -s(1) + Xor_1_3

;

ideal I2 = 
   -And_2_0 + a(1) * b(1),
   -And_2_1 + And_1_2 * And_2_0,
   -Xor_2_2 + And_1_2 + And_2_0 - 2 * And_1_2 * And_2_0,
   -s(2) + Xor_2_2

;

ideal I3 =
   -s(3) + And_2_1

;
ideal F =
   -a(0) + a(0)^2, -a(1) + a(1)^2,
   -b(0) + b(0)^2, -b(1) + b(1)^2
;

poly spec =
   (a(0) + 2*a(1)) * (b(0) + 2*b(1))
   -
   (s(0) + 2*s(1) + 4*s(2) + 8*s(3))
;
reduce (spec, F + I0 + I1 + I2 + I3);
quit;
$\text{diff correct.singular incorrect.singular}$

$< -\text{And}_2\_1 + \text{And}_1\_2 \times \text{And}_2\_0,$

$---$

$> -\text{And}_2\_1 + \text{And}_1\_2 + \text{And}_2\_0 - \text{And}_1\_2 + \text{And}_2\_0,$

$\text{singular correct.singular}$

SINGULAR

... 0

Auf Wiedersehen.

$\text{singular incorrect.singular}$

... 8*b(1)*b(0)*a(1)*a(0) - 8*b(1)*a(1)

...
Implications

1. $J(C)$ is a radical ideal thus no radical membership necessary.

2. We can add the set $F$ of all field polynomials $x(x - 1)$ of all variables $x$.

3. Leading coefficient $-1$ of all gate polynomials, thus computation stays in $\mathbb{Z}$.

4. Still can use rational coefficients $\mathbb{Q}$ (important for Singular).

5. Ideal membership in $\mathbb{Q}[X]$ is co-NP hard even if Gröbner basis is given.

6. Completeness proof allows to derive concrete input assignment if $C$ is incorrect.
Rows and Columns

\[(4a_2 + 2a_1 + 1a_0) \times (4b_2 + 2b_1 + 1b_0)\]

\[FA \quad FA \quad FA \quad FA \quad 0\]

\[32s_5 + 16s_4 + 8s_3 + 4s_2 + 2s_1 + 1s_0\]
Slices

For each output bit \( s_i \) we determine its input cone

\[
I_i := \{ \text{gate } g \mid g \text{ is in input cone of output } s_i \}
\]

We define slices \( S_i \) as the difference of consecutive cones \( I_i \):

\[
S_0 := I_0 \quad S_{i+1} := I_{i+1} \setminus \bigcup_{j=0}^{i} S_j
\]

Definition 4 (Sliced Gröbner Bases).

Let \( G_i \) be the set of polynomial representations of the gates in slice \( S_i \).

Definition 5 (Partial Products). Let \( P_k = \sum_{k=i+j} a_i b_j \).
Algorithm 2.

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 $0$

$$C_i \leftarrow \text{Remainder} \left( 2C_{i+1} + s_i - P_i, \ G_i \cup F \right)$$

return $C_0 = 0$
## Results

<table>
<thead>
<tr>
<th>mult</th>
<th>$n$</th>
<th>Mathematica</th>
<th>Singular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>+inc</td>
<td>-inc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>col</td>
<td>row</td>
</tr>
<tr>
<td>btor</td>
<td>16</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>btor</td>
<td>32</td>
<td>35</td>
<td>531</td>
</tr>
<tr>
<td>btor</td>
<td>64</td>
<td>409</td>
<td>MO</td>
</tr>
<tr>
<td>btor</td>
<td>128</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>16</td>
<td>7</td>
<td>TO</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>32</td>
<td>67</td>
<td>TO</td>
</tr>
<tr>
<td>sp-ar-rc</td>
<td>64</td>
<td>841</td>
<td>MO</td>
</tr>
</tbody>
</table>
Optimizations

- common rewriting
  - auto-reduce $G$ by gates with one parent
  - only if parent is in the same slice
- vanishing constraints
  - add simplifying relations among carry variables
  - for instance $a \land b \land (a \oplus b) = 0$
- pattern match meta gates
  - XOR rewriting
  - full- and half-adder rewriting
- shrinking support of “carry polynomials”
  - *merge* single parent gates of children in lower slice to that slice
  - *promote* non-carry parent gates to slice of children if children are carries
Open Problems

- more complex multipliers
- wallace trees
- booth encoding
- synthesis
- equivalence checking
- other arithmetic circuits
- modular multipliers
- shift, division, ...
- floating point operators
- complexity results
- proofs

Abstract—Verifying arithmetic circuits, and most prominently multipliers, is an important problem but in practice still requires substantial manual effort. Recent work tries to solve this issue using techniques from computer algebra. The most effective approach uses polynomial reasoning over pseudo boolean polynomials. In this paper we give a rigorous formalization of this approach and present a new column-wise verification technique for the correctness of gate-level multipliers which does not require the reduction of a full word-level specification. We formally prove soundness and completeness of our technique, making use of our precise formalization. Our experiments show that simple multipliers can be verified efficiently by using off-the-shelf computer algebra tools, while more complex and optimized multipliers require more sophisticated techniques. Further, our paper independently confirms the effectiveness of previous related work. We make all benchmarks and tools publicly available.

I. INTRODUCTION

Formal verification of arithmetic circuits is motivated by the necessity to avoid issues like the famous Pentium FDIV bug, which is reported to have cost Intel almost half a billion dollar. There have been many attempts since then to verify such circuits, but even today verifying designs with arithmetic parts is not considered to be fully automated. For instance, a common approach is to black-box multipliers and then verify them separately. This might also require insight into the multiplier design, which has to be communicated to the verification tool. Commercial tools can not fully automatically handle full-sized multipliers [24] or huge multipliers occurring in cryptographic circuits. In this paper we will focus, as a first step, on the simplest but also most important arithmetic circuit verification problem of verifying multipliers.

This lack of automation was a common conclusion in three plenary talks at the joint FMCAD’15 and SAT’15 conferences in Austin in 2015, by Anna Slobodova on formal verification of processors, Aaron Tomb on verifying cryptographic circuits, and, from the academic side, Priyank Kalla on methods for data path verification. In order to stimulate research into this direction, particularly the development of fast SAT solving techniques for arithmetic circuit verification, we collected a large set of such benchmarks, generated and submitted CNF encodings of these problems to the SAT 2016 competition and made them publicly available [4]. The competition results confirmed that miters of even small multipliers pose a real challenge to SAT solvers.