# 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

Timişoara, Romania
September 28, 2017

# Equivalence Checking

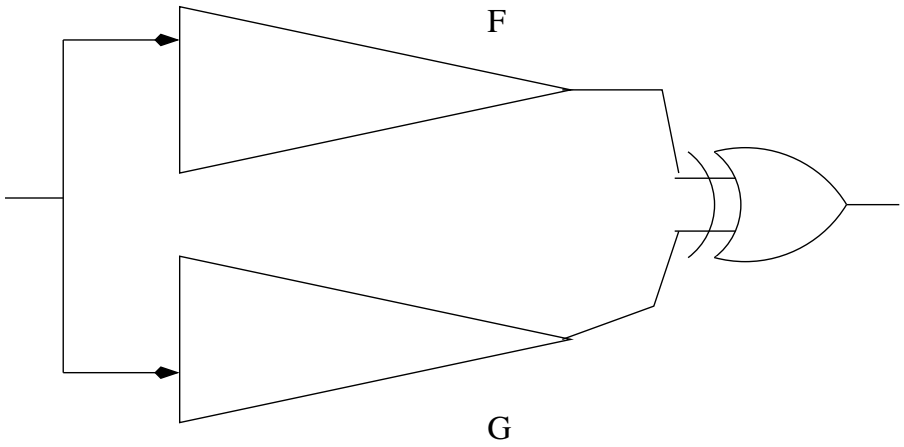
- compare low-level optimized versus high-level golden circuit
- reasons:
  - complex synthesis tool flow
  - engineering change order (manual optimizations)
- considered first successful industrial formal method
- since mid 90'ties
  - BDD sweepping
  - SAT sweeping
- combinational & sequential
  - co-NP versus PSPACE

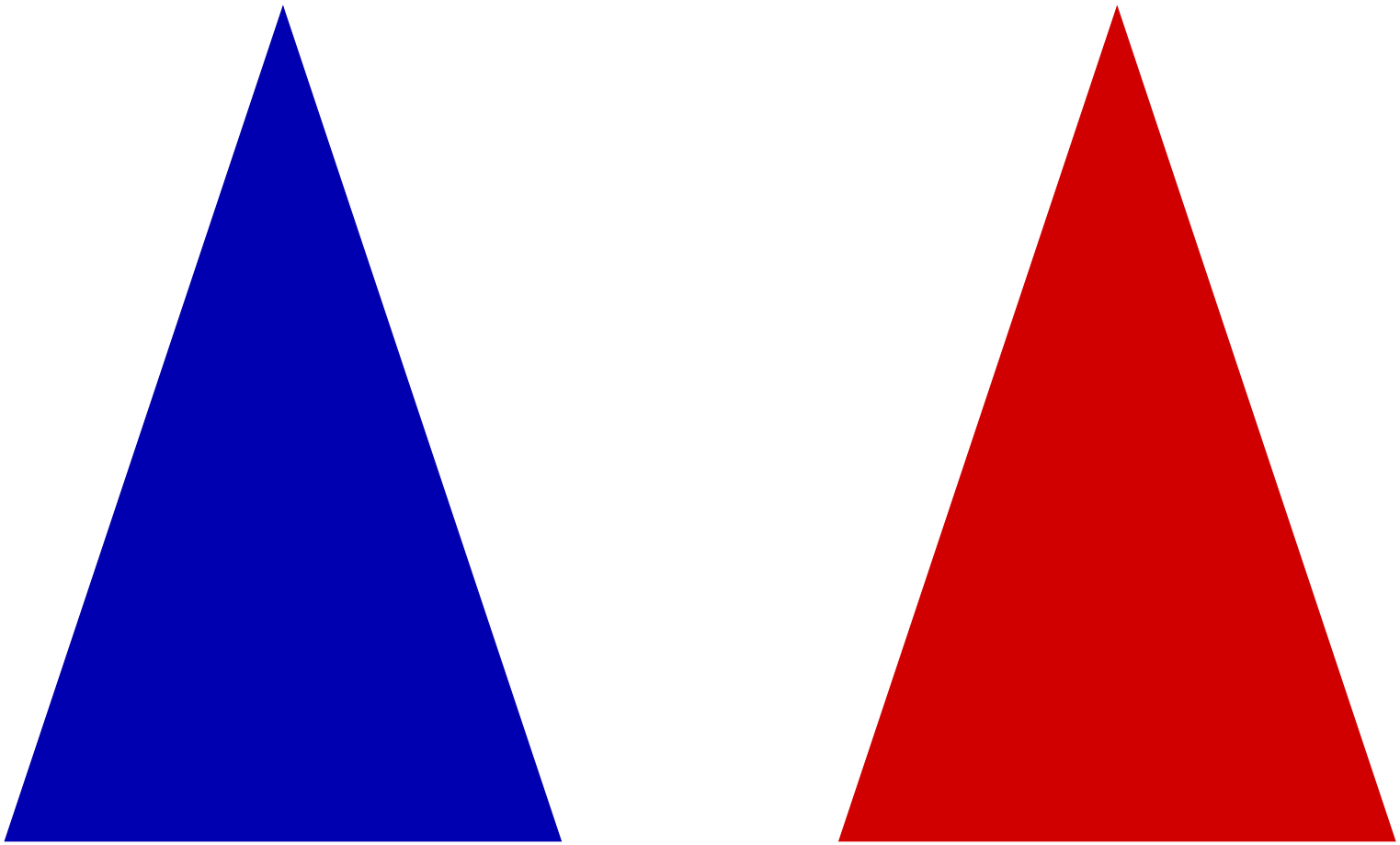
ICCAD'93

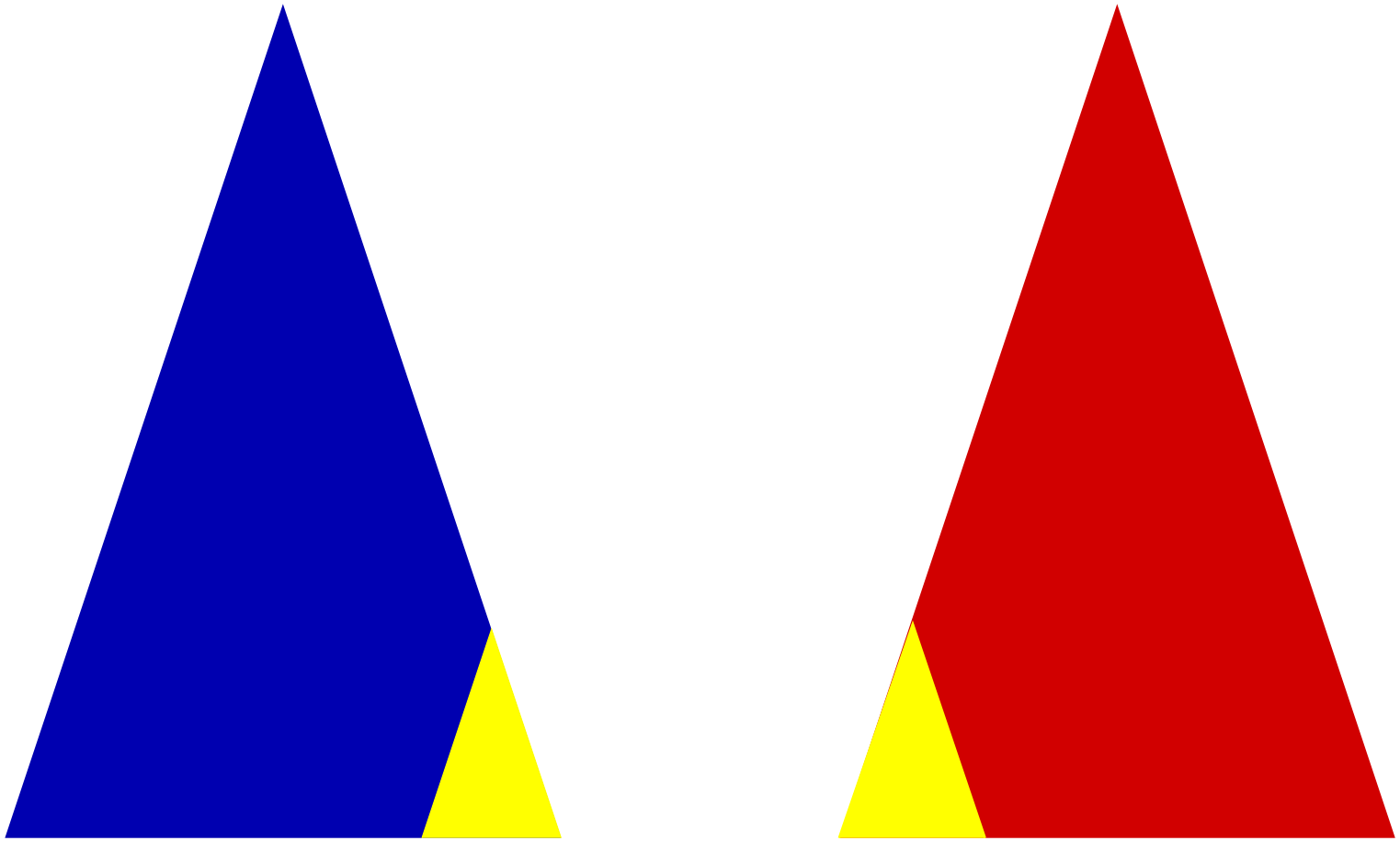
Verification of Large Synthesized Designs

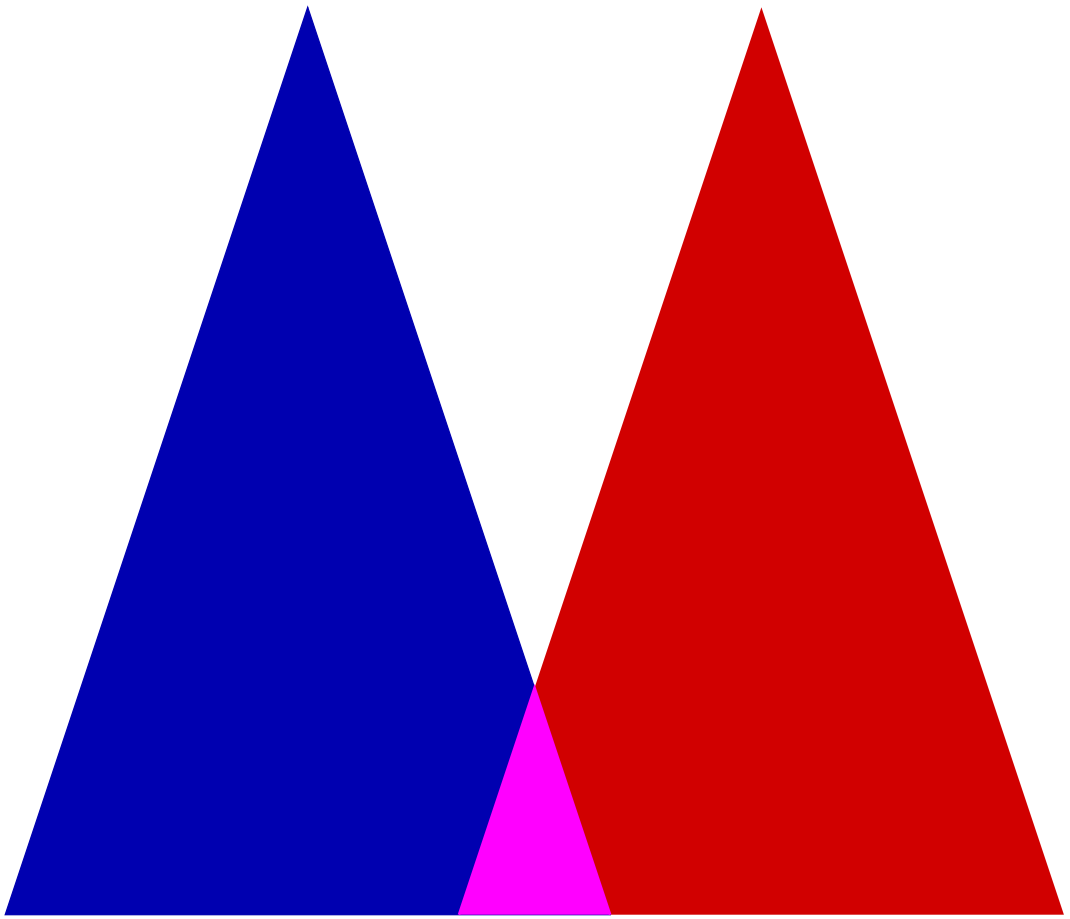
Daniel Brand

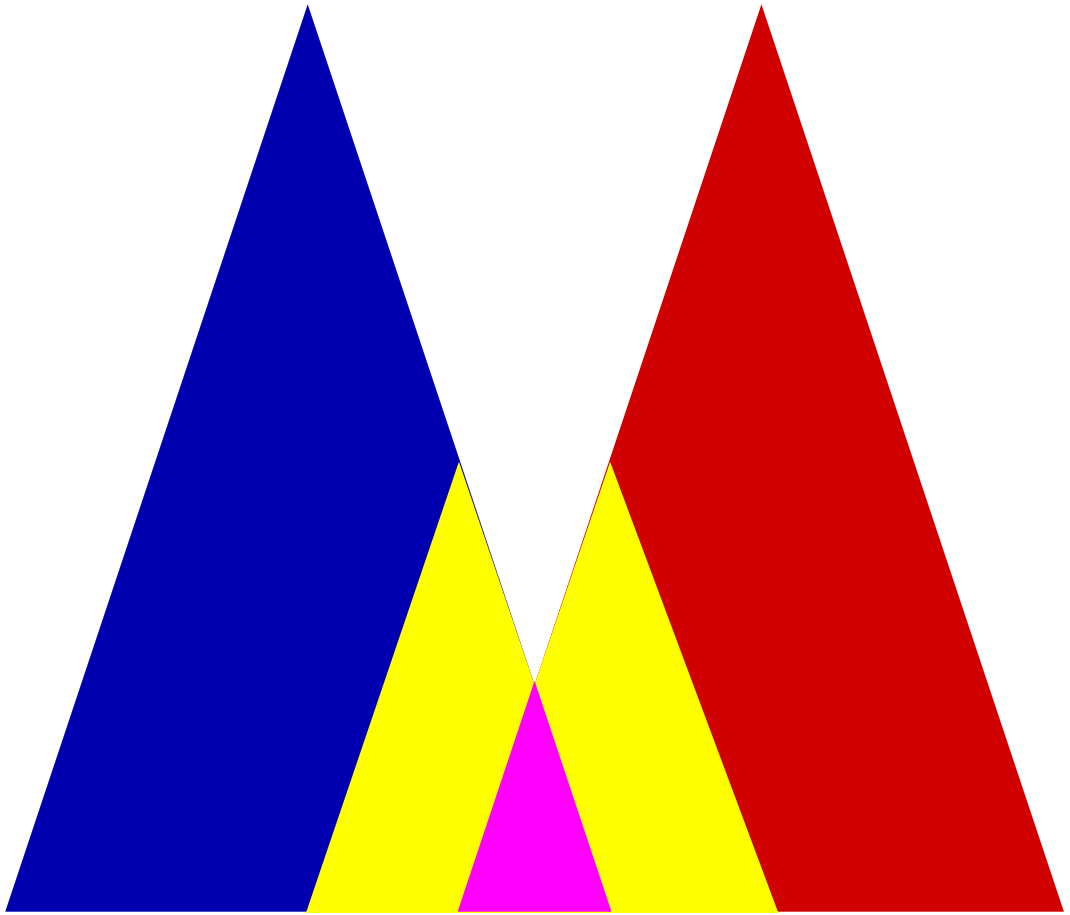
Abstract ....

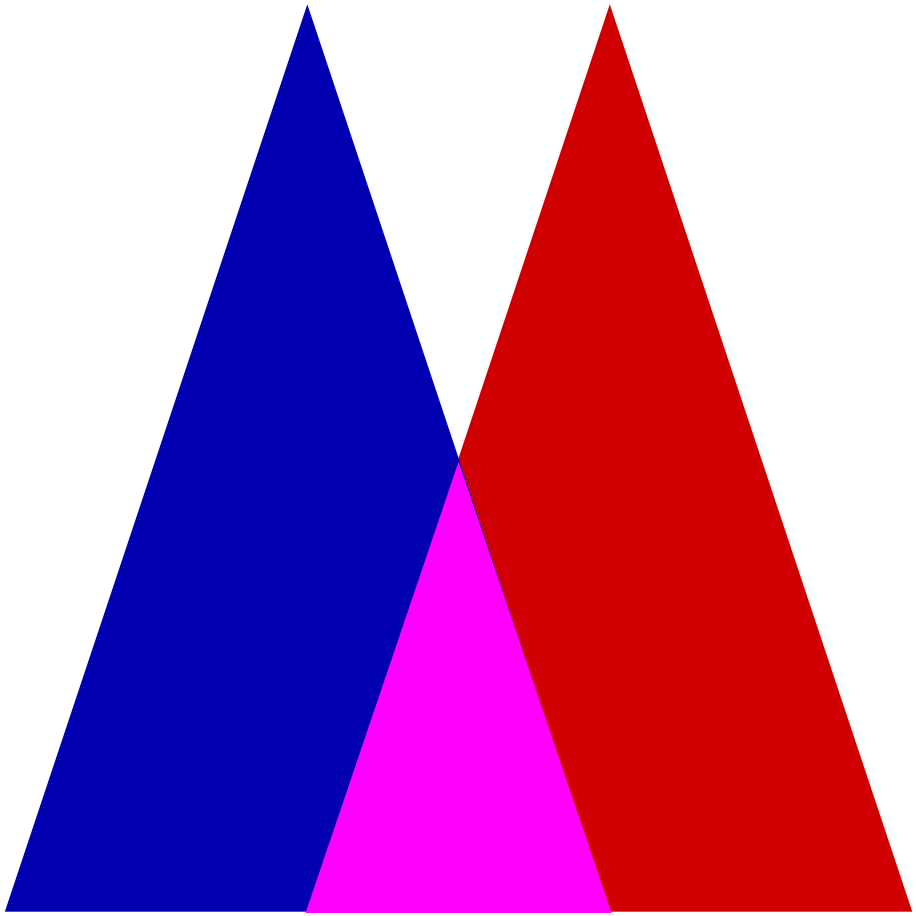


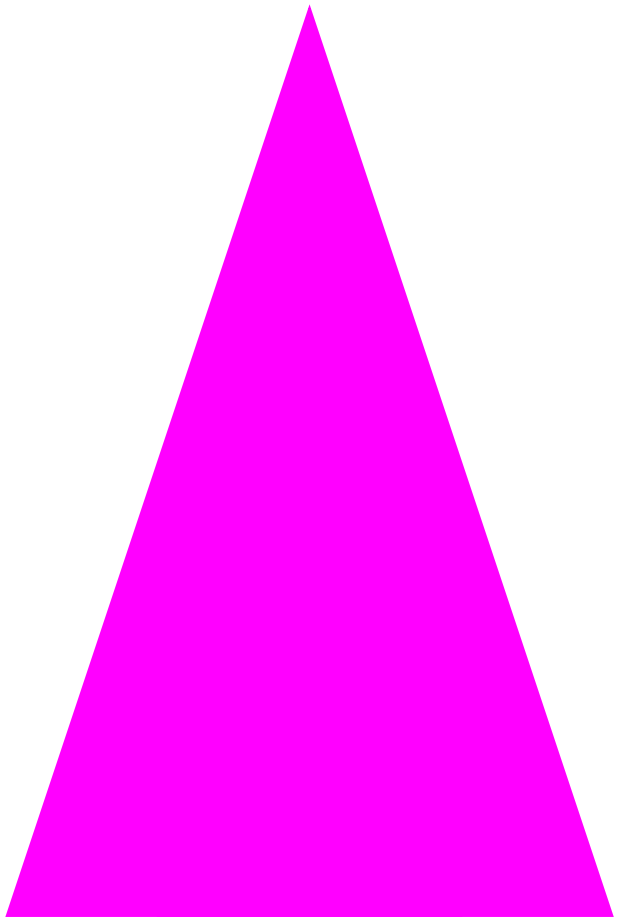










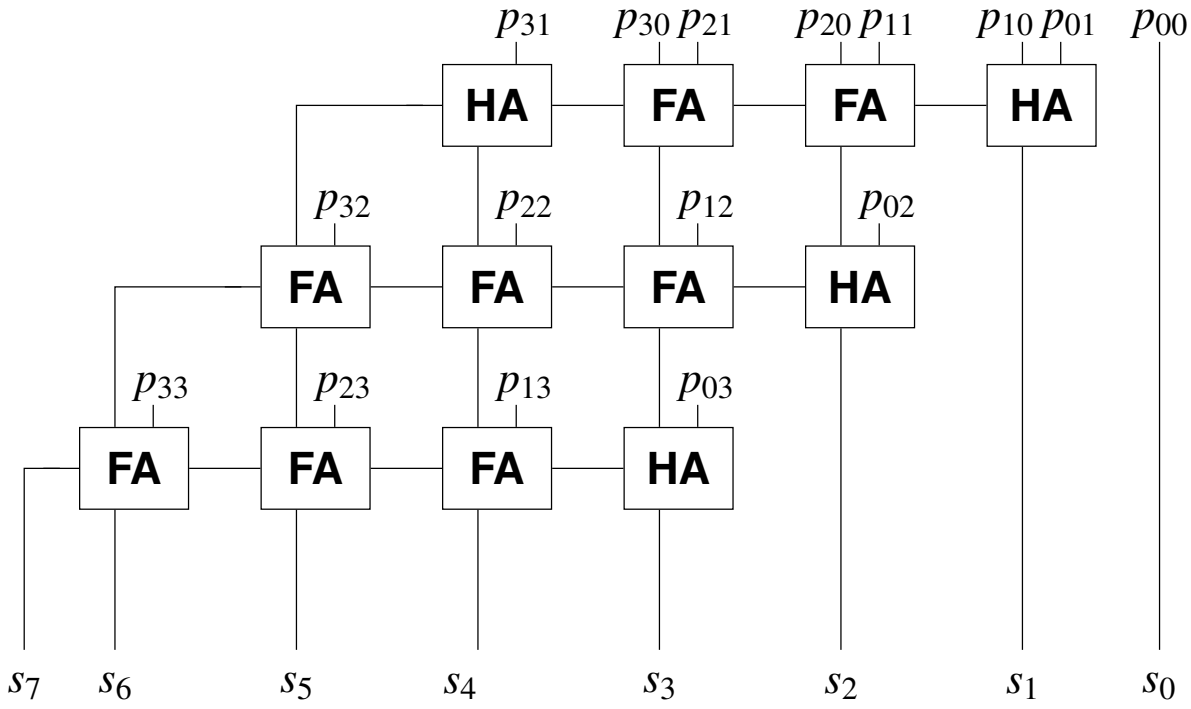


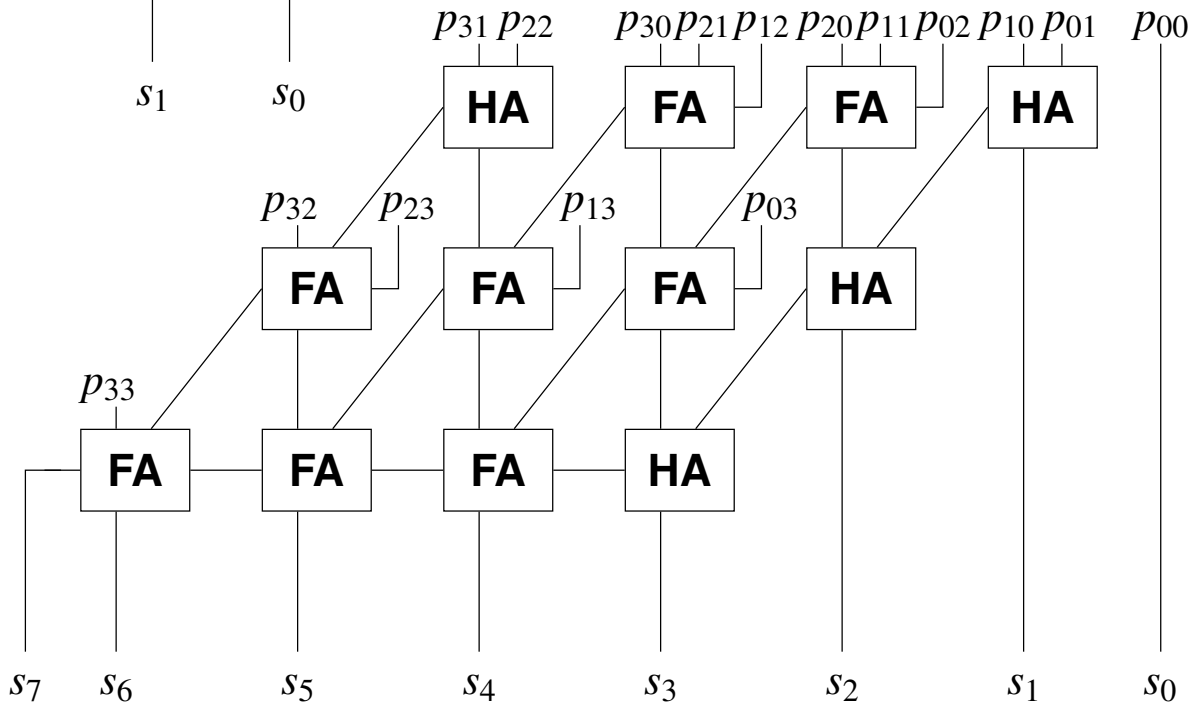
# Binary Multiplication

$$
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & \cdot & 1 & 1 & 0 & 1 & \\
\hline
 & & & & & 1 & 1 & 0 & 1 & \\
 & & & & 1 & 1 & 0 & 1 & & \\
 & & & 1 & 1 & 0 & 1 & & & \\
 & & 1_{2} & 1_{2} & 0_{2} & 1_{1} & 0 & 0 & & \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 &
\end{array}
$$

$15 \cdot 13 = 195$

# Multipliers



# 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)
```

| bits | 1 core Glucose | 1 core Lingeling | 12 core cube-and-conquer March\|iLingeling | 12 core Treengeling |
|---|---|---|---|---|
| 01 | 0.00 | 0.00 | 0.00 | 0.01 |
| 02 | 0.00 | 0.00 | 0.00 | 0.01 |
| 03 | 0.00 | 0.00 | 0.00 | 0.01 |
| 04 | 0.00 | 0.00 | 0.02 | 0.03 |
| 05 | 0.00 | 0.01 | 0.05 | 0.13 |
| 06 | 0.02 | 0.03 | 0.36 | 0.31 |
| 07 | 0.14 | 0.27 | 0.63 | 0.72 |
| 08 | 1.18 | 1.98 | 1.38 | 2.47 |
| 09 | 7.85 | 10.98 | 2.63 | 4.65 |
| 10 | 37.16 | 41.49 | 5.02 | 10.86 |
| 11 | 147.62 | 214.98 | 15.72 | 21.96 |
| 12 | 833.62 | 649.49 | 56.57 | 61.48 |
| 13 | -- | -- | 238.10 | 263.44 |

limit of 900 seconds wall clock time

http://fmv.jku.at/datapath

# Related Work


1. Y.-A. Chen and R.E. Bryant. Verification of arithmetic circuits with binary moment diagrams. In DAC, pages 535–541, **1995**.

2. O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G.-M. Greuel. An algebraic approach for proving data correctness in arithmetic data paths. In CAV, volume 5123 of LNCS, pages 473–486. Springer, **2008**.

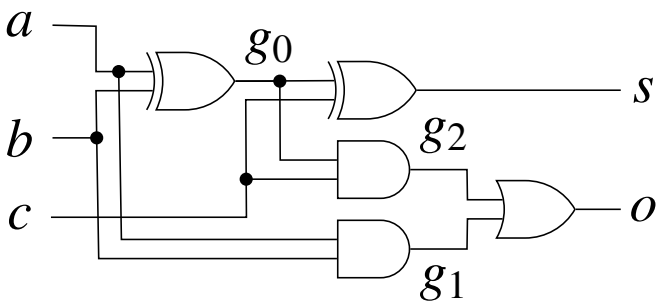
3. C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski. Formal verification of arithmetic circuits by function extraction. IEEE TCAD, 35(12):2131–2142, **2016**.

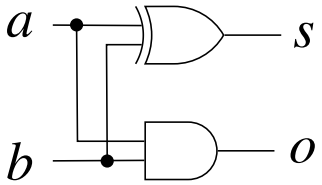
4. A.A.R. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler. Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In DATE, pages 1048–1053. IEEE, 2016.

5. P. Beame and V. Liew. Towards verifying nonlinear integer arithmetic. In CAV, volume 10427 of LNCS, pages 238–258. Springer, **2017**.

6. D. Ritirc, A. Biere, M. Kauers. Column-Wise Verification of Multipliers Using Computer Algebra. in FMCAD. IEEE, **2017**.

# Full-Adder and Half-Adder Gate Polynomials




$$
\begin{aligned}
g_0 &= a \oplus b \\
g_1 &= a \wedge b \\
g_2 &= c \wedge g_0 \\
s &= c \oplus g_0 \\
o &= g_1 \vee g_2
\end{aligned}
$$

$$
\begin{aligned}
s &= a \oplus b \\
o &= a \wedge b
\end{aligned}
$$

$$
\begin{aligned}
-g_0 &+ a+b-2ab \\
-g_1 &+ ab \\
-g_2 &+ cg_0 \\
-s &+ c+g_0-2cg_0 \\
-o &+ g_1+g_2-g_1g_2
\end{aligned}
$$

$$
\begin{aligned}
-s &+ a+b-2ab \\
-o &+ ab
\end{aligned}
$$

$$
-2o-s + a+b+c
$$

$$
-2o-s + a+b
$$

**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 Completness).** $J(C) = I(C)$.

# Multiplier Specification and a First Non-Incremental Algorithm

**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),
  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;
```

```
$ diff correct.singular incorrect.singular
<    -And_2_1 + And_1_2 * And_2_0,
---
>    -And_2_1 + And_1_2 + And_2_0 - And_1_2 + And_2_0,

$ singular correct.singular
     SINGULAR
...
0
Auf Wiedersehen.

$ 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 $x$.

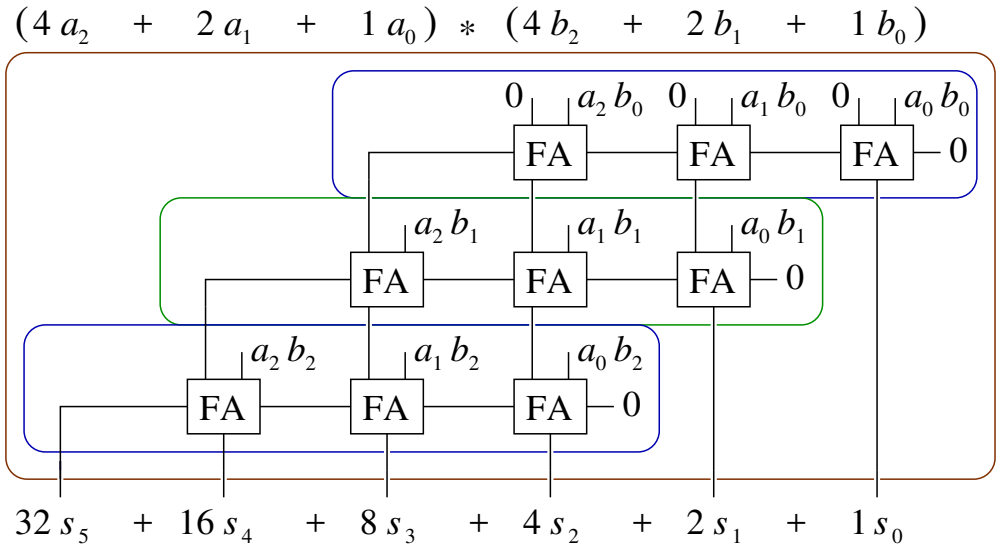
3. Leading coefficient $-1$ of all gate polynomials, thus computation stays in $\mathbb{Z}$.

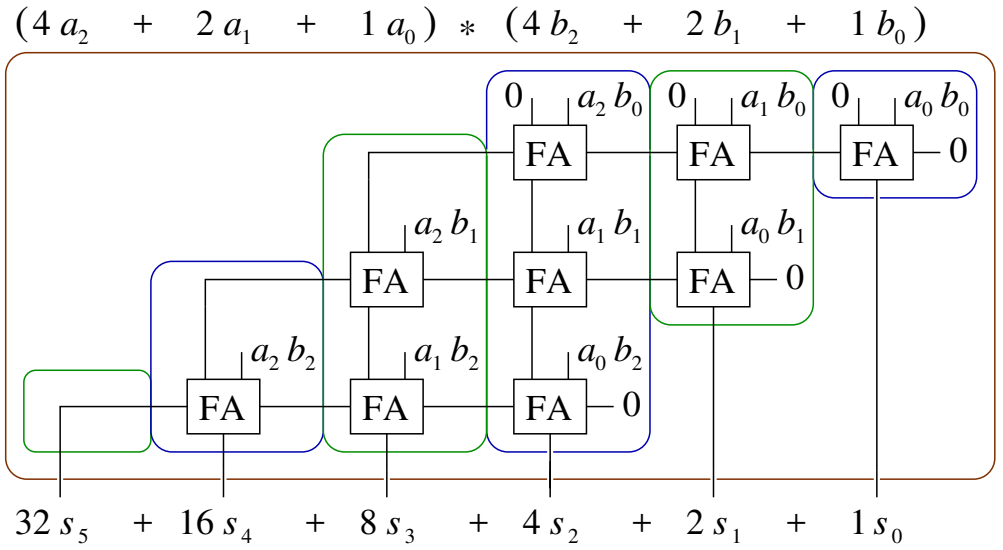
4. Still can use rational coefficients $\mathbb{Q}$ (important for Singular).

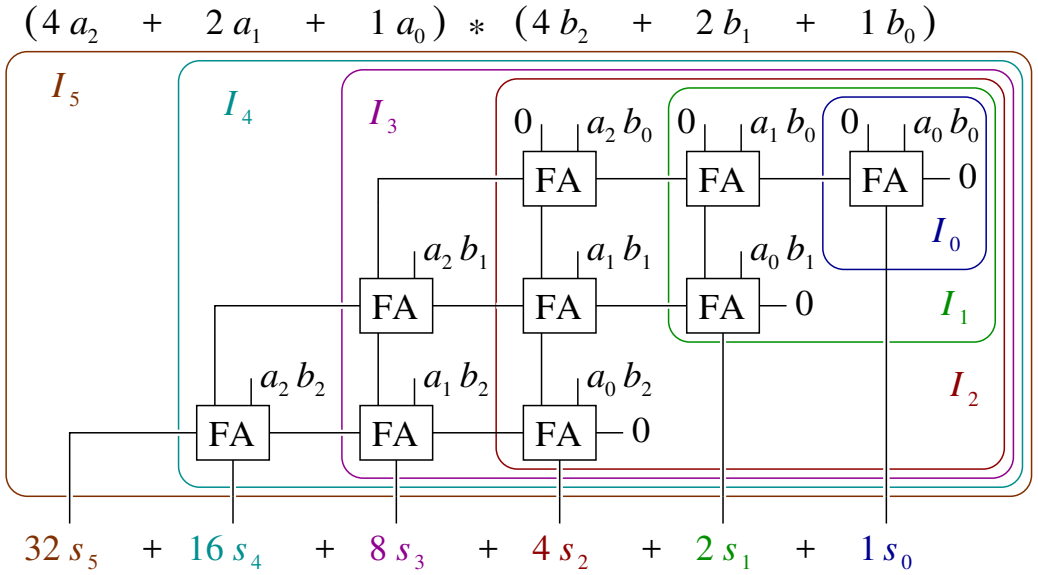
5. Ideal membership in $\mathbb{Q}[X]$ is co-NP hard even if Gröbner basis is given.

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

# Rows and Columns



# 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 $I_i$:

$$S_0 := I_0 \qquad S_{i+1} := I_{i+1} \setminus \bigcup_{j=0}^{i} S_j$$

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

# New Column-Wise Incremental Multiplier Checking Algorithm

**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** $0$

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

**return** $C_0 = 0$

# Results

| mult | $n$ | Mathematica | | | Singular | | |
|---|---|---|---|---|---|---|---|
| | | +inc | -inc | | +inc | -inc | |
| | | | col | row | | col | row |
| btor | 16 | 4 | 12 | 12 | 1 | 2 | 2 |
| btor | 32 | 35 | 531 | 491 | 16 | 53 | 58 |
| btor | 64 | 409 | MO | MO | MO | MO | MO |
| btor | 128 | TO | TO | TO | EE | EE | EE |
| sp-ar-rc | 16 | 7 | TO | TO | 1 | TO | TO |
| sp-ar-rc | 32 | 67 | TO | TO | 39 | TO | TO |
| sp-ar-rc | 64 | 841 | MO | MO | MO | MO | MO |

# 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 $a \wedge b \wedge (a \oplus b) = 0$
- 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 childen 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

# Column-Wise Verification of Multipliers Using Computer Algebra

Daniela Ritirc Armin Biere Manuel Kauers
Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
daniela.ritirc@jku.at armin.biere@jku.at manuel.kauers@jku.at

***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.

Supported by Austrian Science Fund (FWF), NFN S11408-N23 (RiSE), Y464-N18, SFB F5004.

The weak performance of SAT solvers on these benchmarks lead to the conjecture that verifying miters of multipliers and other ring properties after encoding them into CNF needs exponential sized resolution proofs [5], which would imply exponential run-time of CDCL SAT solvers. Surprisingly, however, this conjecture was recently answered negatively [2]. Such ring properties do admit polynomial resolution proofs. However, proof search is non-deterministic. Thus this theoretical result still needs to be transferred into practical SAT solving. The complexity bounds on proof size given in [2] involve polynomials of high degree too.

The first technique which was shown to be able to have prevented the Pentium bug was based on decision diagrams, precisely on binary moment diagrams (BMDs) [10] and variants [11]. While common (gate-level) BDDs are exponential in size for multipliers [6], BMDs remain linear in the number of the input bits of a multiplier (using edge weights). However, the BMD approach is not robust, in the sense that it still requires structural knowledge of the multipliers to determine the order in which BMDs are built, which has tremendous influence on performance. Actually only a row-wise backward substitution approach seems to be feasibly [9], which in addition assumes a simple carry-save-adder (CSA) design.

Recent algebraically inspired techniques [12], [28] based on so-called function-extraction also fail for even slightly optimized multiplier designs. On the positive side, this technique is able to handle very large clean multipliers.

In even more recent work [24] substantial progress was made. The authors use a dedicated polynomial reduction engine and also gave various optimizations (discussed further down), which made their algebraic technique scale to large non-trivial multiplier designs of various architectures [16] (called AOKI benchmarks in the following) even with and without Booth reencoding. It is still unclear however, whether their technique is robust under synthesis or technology mapping. Their arguments for soundness and completeness are rather imprecise. Their tool is not available, nor details about the experiments. Benchmarks have not been published either.

There is a substantial amount of previous work for arithmetic circuit verification. We focus on comparing our approach to the currently most successful techniques for verifying multipliers, which all are using some form of algebraic reasoning [28], [24]. For an up-to-date discussion of related work and a more comprehensive list see the recent article [28].