CHECKING CIRCUITS FOR INTEGER MULTIPLICATION USING GRÖBNER BASES

Manuel Kauers · Institute for Algebra · JKU · Linz, Austria.

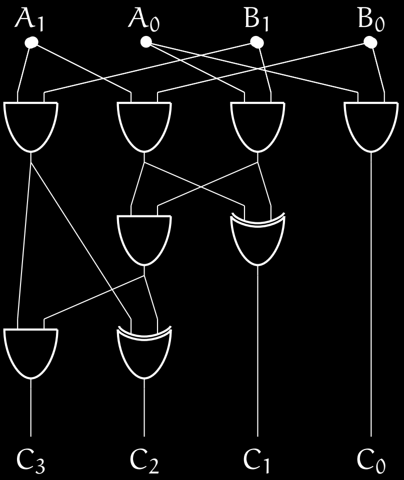
Joint work with Armin Biere and Daniela Ritirc

$$47495831406 \times 19167053557$$

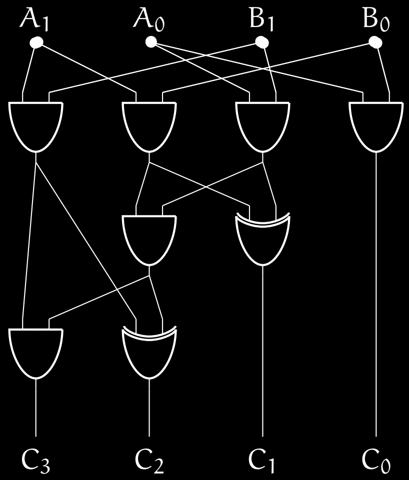
$$47495831406 \times 19167053557$$

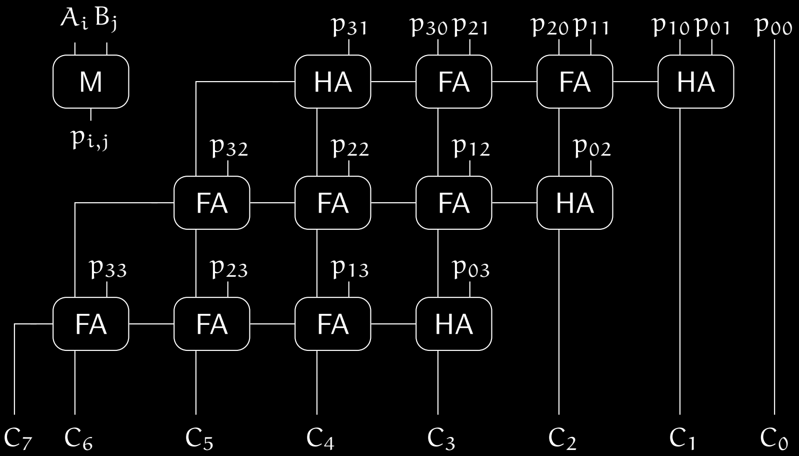
$$10001011001 \times 11011000001$$

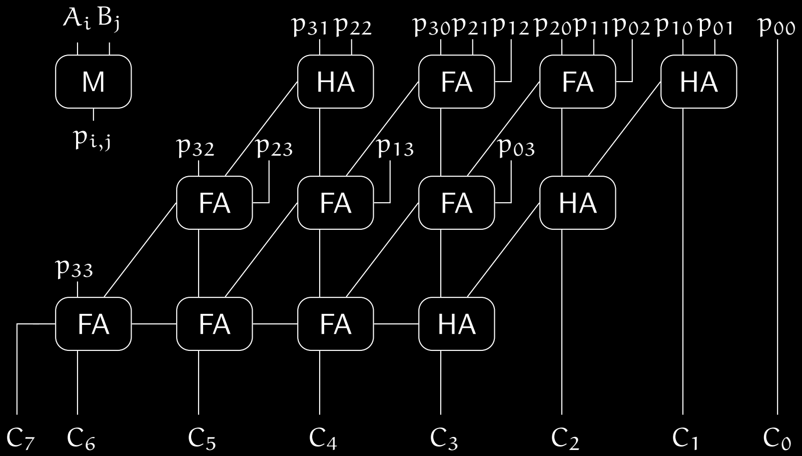
$$10001011001 \times 11011000001$$



A1
A0
B1
B0
C3
C2
C1
C0


A1
A0
B1
B0
C3
C2
C1
C0


A_i B_j
M
p_i,j
p31
p30 p21
p20 p11
p10 p01
p00
HA
FA
FA
HA
p32
p22
p12
p02
FA
FA
FA
HA
p33
p23
p13
p03
FA
FA
FA
HA
C7
C6
C5
C4
C3
C2
C1
C0



A_i B_j
M
$p_{i,j}$
p_{31} p_{22}
p_{30} p_{21} p_{12} p_{20} p_{11} p_{02} p_{10} p_{01}
p_{00}
HA
FA
FA
HA
p_{32}
p_{23}
p_{13}
p_{03}
FA
FA
FA
HA
p_{33}
FA
FA
FA
HA
C_7 C_6 C_5 C_4 C_3 C_2 C_1 C_0

Questions:

Questions:

- Are these circuits correct?

Questions:

- Are these circuits correct?
- Are these circuits really correct?

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Yes, because we understand the idea and don't see any bug.

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Yes, because we proved its correctness by computer algebra.

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:
Yes, because we constructed a proof by computer algebra and checked it with a proof checker.

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:

Questions:

- Are these circuits correct?
- Are these circuits really correct?
- Are these circuits really really correct?
- Are these circuits really really really correct?

Answers:
We don't know yet, because the proof checker we used is not formally verified.

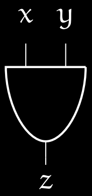
Are they really correct?

- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$

- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$
- A circuit is "correct" if it corresponds the right function

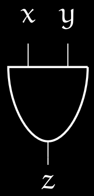
- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$
- A circuit is "correct" if it corresponds the right function
- The behaviour of a gate is described by a polynomial equation

- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$
- A circuit is "correct" if it corresponds the right function
- The behaviour of a gate is described by a polynomial equation

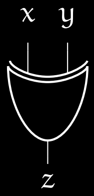


$$z = xy$$

- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$
- A circuit is "correct" if it corresponds the right function
- The behaviour of a gate is described by a polynomial equation

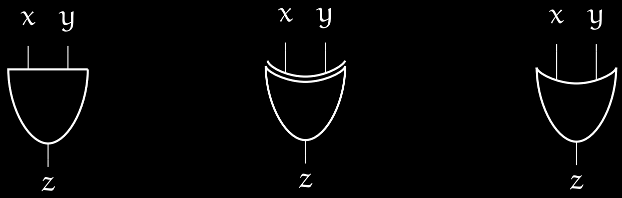


$z = xy$



$z = x + y - 2xy$

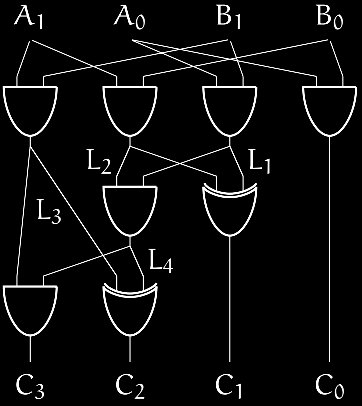
- Every circuit implements a certain function $\{0,1\}^n \to \{0,1\}^m$
- A circuit is “correct” if it corresponds the right function
- The behaviour of a gate is described by a polynomial equation



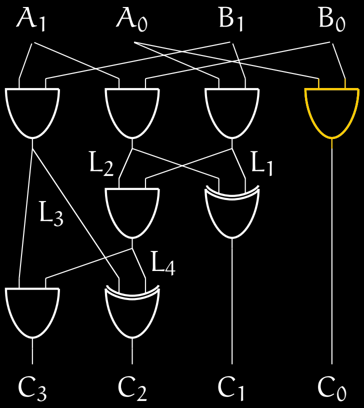
$z = xy$ $z = x + y - 2xy$ $z = x + y - xy$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.

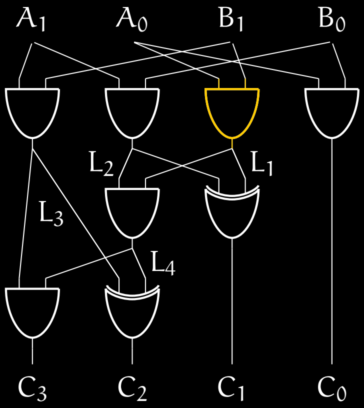


- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



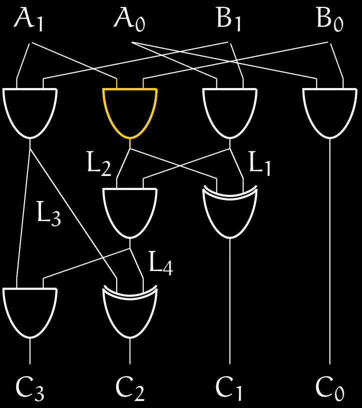
  - $C_0 = A_0 B_0$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



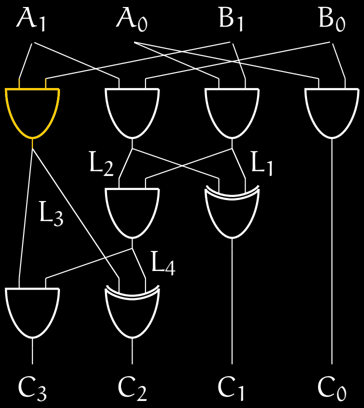
  - $C_0 = A_0B_0$
  - $L_1 = A_0B_1$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



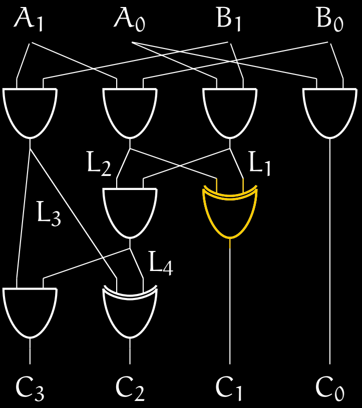
  - $C_0 = A_0B_0$
  - $L_1 = A_0B_1$
  - $L_2 = A_1B_0$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



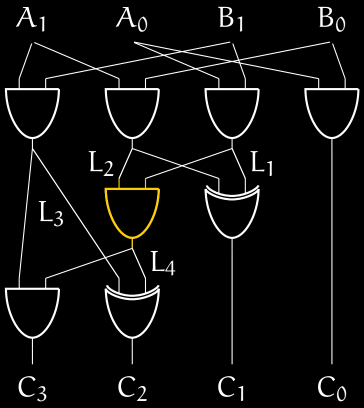
- $C_0 = A_0 B_0$
- $L_1 = A_0 B_1$
- $L_2 = A_1 B_0$
- $L_3 = A_1 B_1$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



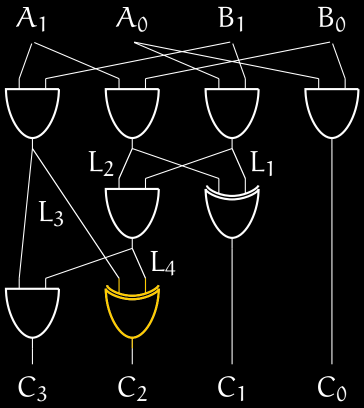
  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



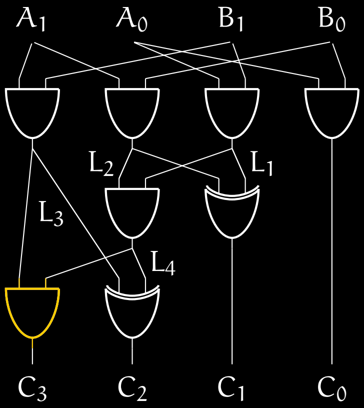
  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$
  - $L_4 = L_1 L_2$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



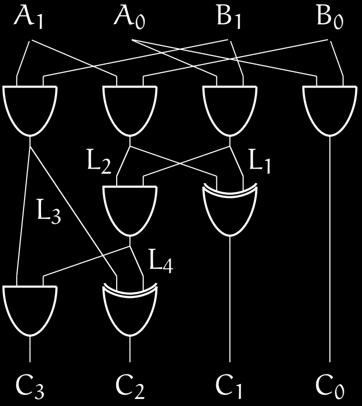
  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$
  - $L_4 = L_1 L_2$
  - $C_2 = L_3 + L_4 - 2L_3 L_4$

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$
  - $L_4 = L_1 L_2$
  - $C_2 = L_3 + L_4 - 2L_3 L_4$
  - $C_3 = L_3 L_4$

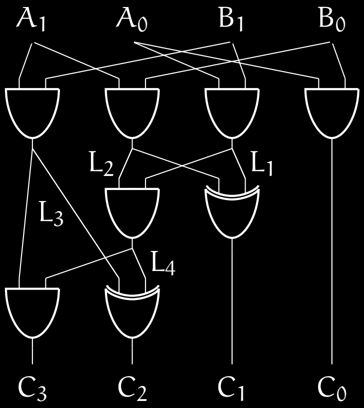
- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$
  - $L_4 = L_1 L_2$
  - $C_2 = L_3 + L_4 - 2L_3 L_4$
  - $C_3 = L_3 L_4$

- We also have polynomials for restricting the range of the variables to $\{0, 1\}$.

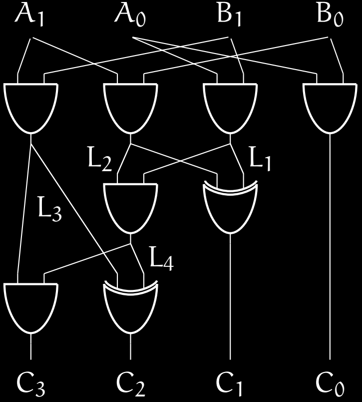
- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



  - $C_0 = A_0 B_0$
  - $L_1 = A_0 B_1$
  - $L_2 = A_1 B_0$
  - $L_3 = A_1 B_1$
  - $C_1 = L_1 + L_2 - 2L_1 L_2$
  - $L_4 = L_1 L_2$
  - $C_2 = L_3 + L_4 - 2L_3 L_4$
  - $C_3 = L_3 L_4$
  - $A_0(A_0 - 1) = 0$
  - $A_1(A_1 - 1) = 0$
  - $B_0(B_0 - 1) = 0$
  - $B_1(B_1 - 1) = 0$

- We also have polynomials for restricting the range of the variables to $\{0, 1\}$.

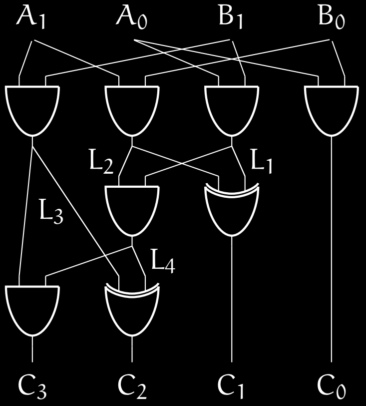
- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



  - $C_0 - A_0 B_0 = 0$
  - $L_1 - A_0 B_1 = 0$
  - $L_2 - A_1 B_0 = 0$
  - $L_3 - A_1 B_1 = 0$
  - $C_1 - L_1 - L_2 + 2L_1 L_2 = 0$
  - $L_4 - L_1 L_2 = 0$
  - $C_2 - L_3 - L_4 + 2L_3 L_4 = 0$
  - $C_3 - L_3 L_4 = 0$
  - $A_0(A_0 - 1) = 0$
  - $A_1(A_1 - 1) = 0$
  - $B_0(B_0 - 1) = 0$
  - $B_1(B_1 - 1) = 0$

- We also have polynomials for restricting the range of the variables to $\{0, 1\}$.

- For the whole circuit, we have one variable for each circuit input bit and each gate output, and one polynomial per gate.



  - $C_0 - A_0 B_0$
  - $L_1 - A_0 B_1$
  - $L_2 - A_1 B_0$
  - $L_3 - A_1 B_1$
  - $C_1 - L_1 - L_2 + 2L_1 L_2$
  - $L_4 - L_1 L_2$
  - $C_2 - L_3 - L_4 + 2L_3 L_4$
  - $C_3 - L_3 L_4$
  - $A_0(A_0 - 1)$
  - $A_1(A_1 - 1)$
  - $B_0(B_0 - 1)$
  - $B_1(B_1 - 1)$

- We also have polynomials for restricting the range of the variables to $\{0, 1\}$.

- The ideal generated by these polynomial contains all the polynomial relations implied by the circuit. The ideal is radical and has dimension zero.

- The ideal generated by these polynomial contains all the polynomial relations implied by the circuit. The ideal is radical and has dimension zero.
- The polynomials form a Gröbner bases for any lexicographic order such that $x_i < x_j$ whenever there is a gate that has x_i as input and x_j as output.

- The ideal generated by these polynomial contains all the polynomial relations implied by the circuit. The ideal is radical and has dimension zero.
- The polynomials form a Gröbner bases for any lexicographic order such that $x_i < x_j$ whenever there is a gate that has x_i as input and x_j as output.
- Taking $\mathbb{Q}$ as ground field, a multiplier circuit is correct iff its ideal contains the polynomial

$$\left(\sum_{k=0}^{2^n-1} 2^k A_k\right)\left(\sum_{k=0}^{2^n-1} 2^k B_k\right) - \left(\sum_{k=0}^{2^{2n}-1} 2^k C_k\right)$$

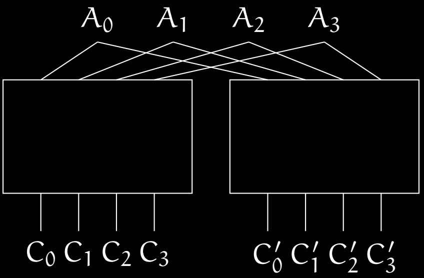
- The ideal generated by these polynomial contains all the polynomial relations implied by the circuit. The ideal is radical and has dimension zero.
- The polynomials form a Gröbner bases for any lexicographic order such that $x_i < x_j$ whenever there is a gate that has x_i as input and x_j as output.
- Taking $\mathbb{Q}$ as ground field, a multiplier circuit is correct iff its ideal contains the polynomial

$$\left(\sum_{k=0}^{2^n-1} 2^k A_k\right)\left(\sum_{k=0}^{2^n-1} 2^k B_k\right) - \left(\sum_{k=0}^{2^{2n}-1} 2^k C_k\right)$$

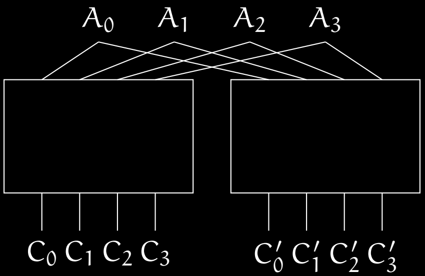
- Correctness thus reduces to ideal membership test.

If there is circuit which we know to be correct, we can also check whether another circuit implements the same function:

If there is circuit which we know to be correct, we can also check whether another circuit implements the same function:



If there is circuit which we know to be correct, we can also check whether another circuit implements the same function:



The circuits are equivalent iff $\sum_{k=0}^{n} 2^k(C_k - C'_k) \in I$.

- In theory, all this has been known for some time.

- In theory, all this has been known for some time.
- In practice, for nontrivial circuits, it's not as easy at it seems.

- In theory, all this has been known for some time.
- In practice, for nontrivial circuits, it's not as easy at it seems.
- As we get the Gröbner basis for free, we "just" have to compute a normal form.

- In theory, all this has been known for some time.
- In practice, for nontrivial circuits, it's not as easy at it seems.
- As we get the Gröbner basis for free, we "just" have to compute a normal form.
- For real world circuits (e.g., 64bit multipliers), this can be difficult.

- In theory, all this has been known for some time.
- In practice, for nontrivial circuits, it's not as easy at it seems.
- As we get the Gröbner basis for free, we "just" have to compute a normal form.
- For real world circuits (e.g., 64bit multipliers), this can be difficult.
- Some special purpose improvements we use in our code:
  - We divide the circuit into "slices" and do one reduction per slice. This prevents some bad choices during the reduction. [FMCAD'16]
  - We preprocess the Gröbner bases by eliminating some variables that only occur "locally". This prevents some amount of expression swell. [DATE'18]

Are they really really correct?

- Can we trust the computer algebra system and/or the implementation of our own improvements?

- Can we trust the computer algebra system and/or the implementation of our own improvements?
- Can we construct a checkable proof object rather than a yes/no answer?

- Can we trust the computer algebra system and/or the implementation of our own improvements?
- Can we construct a checkable proof object rather than a yes/no answer?
- Recall: $g \in \langle f_1, \ldots, f_m \rangle \iff g = p_1 f_1 + \cdots + p_m f_m$ for certain polynomials p_i.

- Can we trust the computer algebra system and/or the implementation of our own improvements?
- Can we construct a checkable proof object rather than a yes/no answer?
- Recall: $g \in \langle f_1, \dots, f_m \rangle \iff g = p_1 f_1 + \cdots + p_m f_m$ for certain polynomials p_i.
- These cofactors $p_1, \dots, p_m$ can serve as certificate of the ideal membership.

- Can we trust the computer algebra system and/or the implementation of our own improvements?
- Can we construct a checkable proof object rather than a yes/no answer?
- Recall: $g \in \langle f_1, \ldots, f_m \rangle \iff g = p_1 f_1 + \cdots + p_m f_m$ for certain polynomials p_i.
- These cofactors $p_1, \ldots, p_m$ can serve as certificate of the ideal membership.
- Again, this is well-known in theory, but not so easy in practice: the cofactors can be quite large.

- Translate the defining properties of ideals into a formal proof system:

- Translate the defining properties of ideals into a formal proof system:

$$\forall\, p, q \in I : p + q \in I$$

- Translate the defining properties of ideals into a formal proof system:

$$\forall\ p, q \in I : p + q \in I$$

$$\forall\ p \in \mathbb{K}[X]\ \forall\ q \in I : pq \in I$$

- Translate the defining properties of ideals into a formal proof system:

$$\forall\, p, q \in I : p + q \in I \qquad \rightsquigarrow \qquad \frac{p \quad q}{p + q}$$

$$\forall\, p \in \mathbb{K}[X]\ \forall\, q \in I : pq \in I \qquad \rightsquigarrow \qquad \frac{q}{pq}$$

- Translate the defining properties of ideals into a formal proof system:

$$\forall\, p, q \in I : p + q \in I \qquad \rightsquigarrow \qquad \frac{p \quad q}{p + q}$$

$$\forall\, p \in \mathbb{K}[X]\ \forall\, q \in I : pq \in I \qquad \rightsquigarrow \qquad \frac{q}{pq}$$

- We construct a formal proof by tracing the reduction process

```
⋮
* : -b+1-a,       a,              -a*b+a-a^2;
+ : -a*b+a-a^2,   a^2-a,          -a*b;
+ : -a*b,         -c+a*b,         -c;
* : -c,           -1,             c;
⋮
```

Observations: (for n-bit multipliers)

Observations: (for n-bit multipliers)

- Suppose that an ideal membership testing takes time X

Observations: (for n-bit multipliers)

- Suppose that an ideal membership testing takes time X
- Then proof generation costs $\approx 100X$

Observations: (for n-bit multipliers)

- Suppose that an ideal membership testing takes time X
- Then proof generation costs $\approx 100X$
- Verifying that the proof is correct costs $\approx X/100$

Observations: (for n-bit multipliers)

- Suppose that an ideal membership testing takes time X
- Then proof generation costs $\approx 100X$
- Verifying that the proof is correct costs $\approx X/100$
- Proof length seems to grow like $O(n^2)$

Observations: (for n-bit multipliers)

- Suppose that an ideal membership testing takes time X
- Then proof generation costs $\approx 100X$
- Verifying that the proof is correct costs $\approx X/100$
- Proof length seems to grow like $O(n^2)$
- Theoretical upper bound for resolution proof size $O(n^7 \log n)$ [Beame et al. 2017]

Are they really really really correct?

- Since the proof format is rather low-level, it doesn't take much to write a checker.

- Since the proof format is rather low-level, it doesn't take much to write a checker.
- We have written two checkers: one based on Python and Singular and another one purely in C.

- Since the proof format is rather low-level, it doesn't take much to write a checker.
- We have written two checkers: one based on Python and Singular and another one purely in C.
- But who will check the checkers? So far we have not made any efforts in this direction.

- Since the proof format is rather low-level, it doesn't take much to write a checker.
- We have written two checkers: one based on Python and Singular and another one purely in C.
- But who will check the checkers? So far we have not made any efforts in this direction.
- Also the script which turns the given circuit into polynomials might require verification.

- Since the proof format is rather low-level, it doesn't take much to write a checker.
- We have written two checkers: one based on Python and Singular and another one purely in C.
- But who will check the checkers? So far we have not made any efforts in this direction.
- Also the script which turns the given circuit into polynomials might require verification.
- No matter what we do: there is no absolute certainty, but we are reasonably sure that the circuits are correct.