

Using Resolution and Cutting Planes for Verification of Nonlinear Bit-Vector Properties

Paul Beame, Vincent Liew [CAV 2017, JACM 2020]

Vincent Liew, Paul Beame, Jo Devriendt, Jan Elffers,
Jakob Nordström [FMCAD 2020]

Bit-Vector Verification of Hardware/Software

Code:

```
int min(int x, int y) {  
    return y ^ ((x ^ y) & -(x < y));  
}
```

bit-level operations

arithmetic
operation

Specification:

$$[(s = x) \vee (s = y)] \wedge (s \leq x) \wedge (s \leq y)$$

Inputs: x, y Output: s

Step 1: Model

- $\text{int } x, y \mapsto$ 32-bit vectors x, y .
- Directly model int operators using:
 - **Arithmetic operations:** $+, -, \times, \%, <, >$
 - **Bit-level operations:** $\&, \wedge, \oplus, \circ, \ll, \gg$
- Write bit-vector formula ϕ asserting code does **not** follow specification

Step 2: Solve

- Send formula ϕ to **bit-vector solver** to prove UNSAT
 - Formula ϕ UNSAT \rightarrow Program follows specification

Challenge: Nonlinear arithmetic

Empirically, great success if all arithmetic is linear but...

Major problems with non-linear arithmetic

No bit-vector solver is close to working well in general at verifying:

- Hardware implementations that involve multiplier circuits
 - Though significant recent progress on multiplier circuits in isolation
[Kaufmann et al., 2017-2019]
- Software involving multiplication operations

Bit-Vector Verification of Hardware/Software

Hardware:

- Directly model circuits using Boolean logic and gate variables

Software:

- Directly model operations in bit-vector language
- Apply theories to simplify/prove via **pre-processing** (e.g. un-interpreted functions, arithmetic identities)

Core of the challenge: Mix of Boolean logic and arithmetic

- If no direct solution, “**bit-blast/flatten**” formulas to convert arithmetic to **fixed bit-width**, at least 32/64 bits
 - **Replace arithmetic operations using gate variables and constraints for circuits that evaluate them**
- Send resulting formula to SAT Solver.

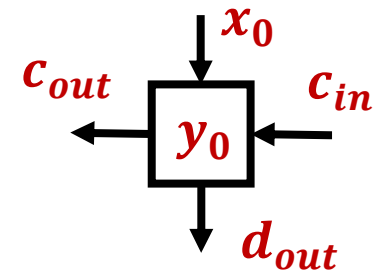
Circuits for $x + y$

Length 1:

- Use *full adder* circuit

$$c_{out} = MAJ(x_0, y_0, c_{in})$$

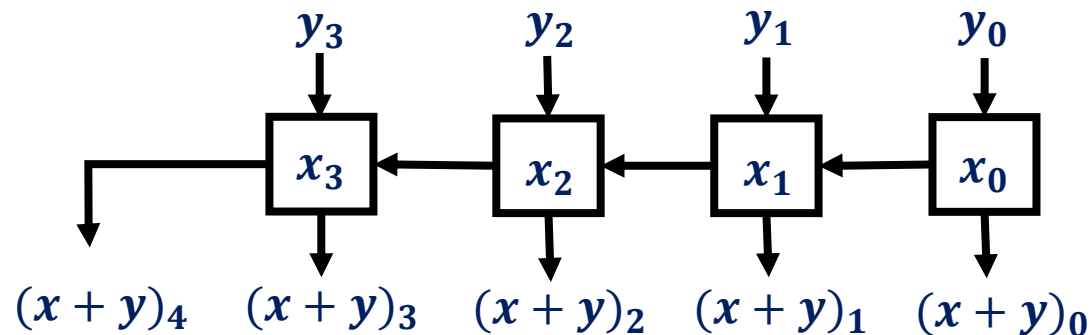
$$d_{out} = x_0 \oplus y_0 \oplus c_{in}$$



- Conservation of weight: $2c_{out} + d_{out} = x_0 + y_0 + c_{in}$

Length n :

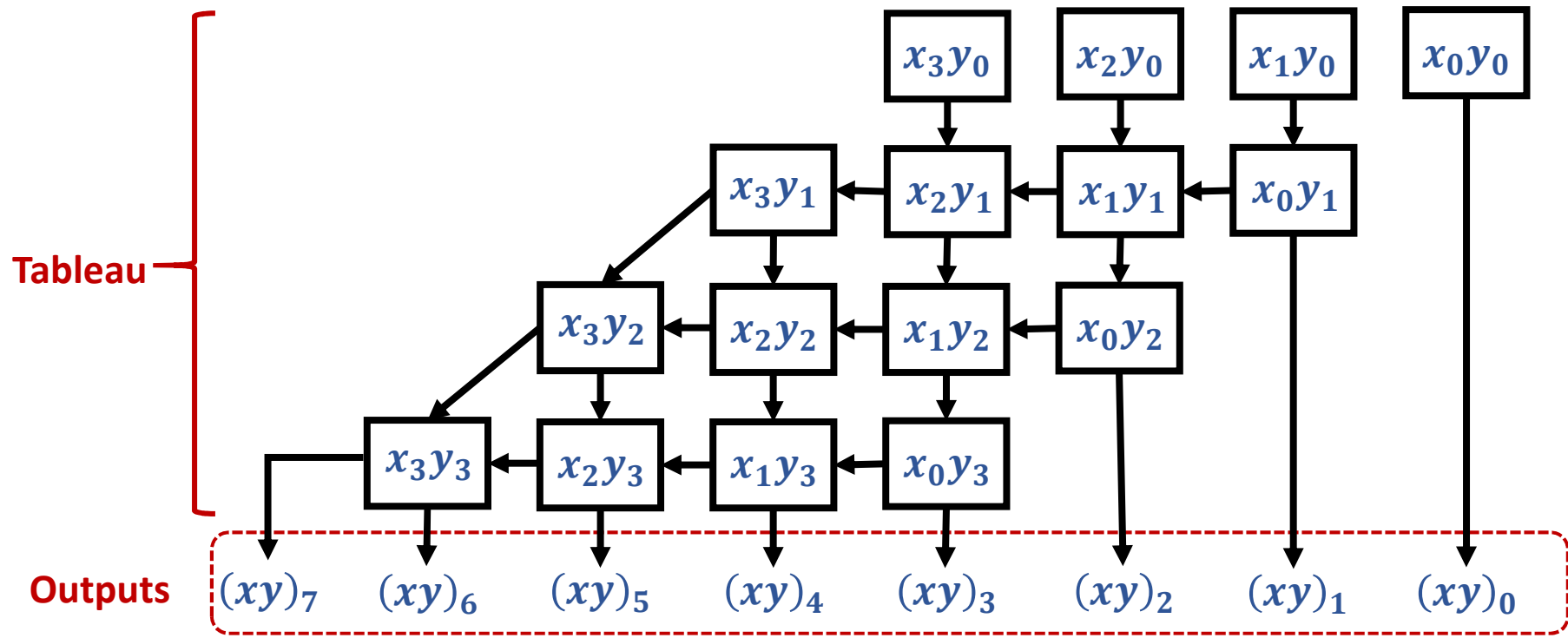
- Chain **full adders** to form *ripple-carry adder* circuit



Example Circuits for $x \times y$

- Stack ripple-carry adders to make **array multiplier**

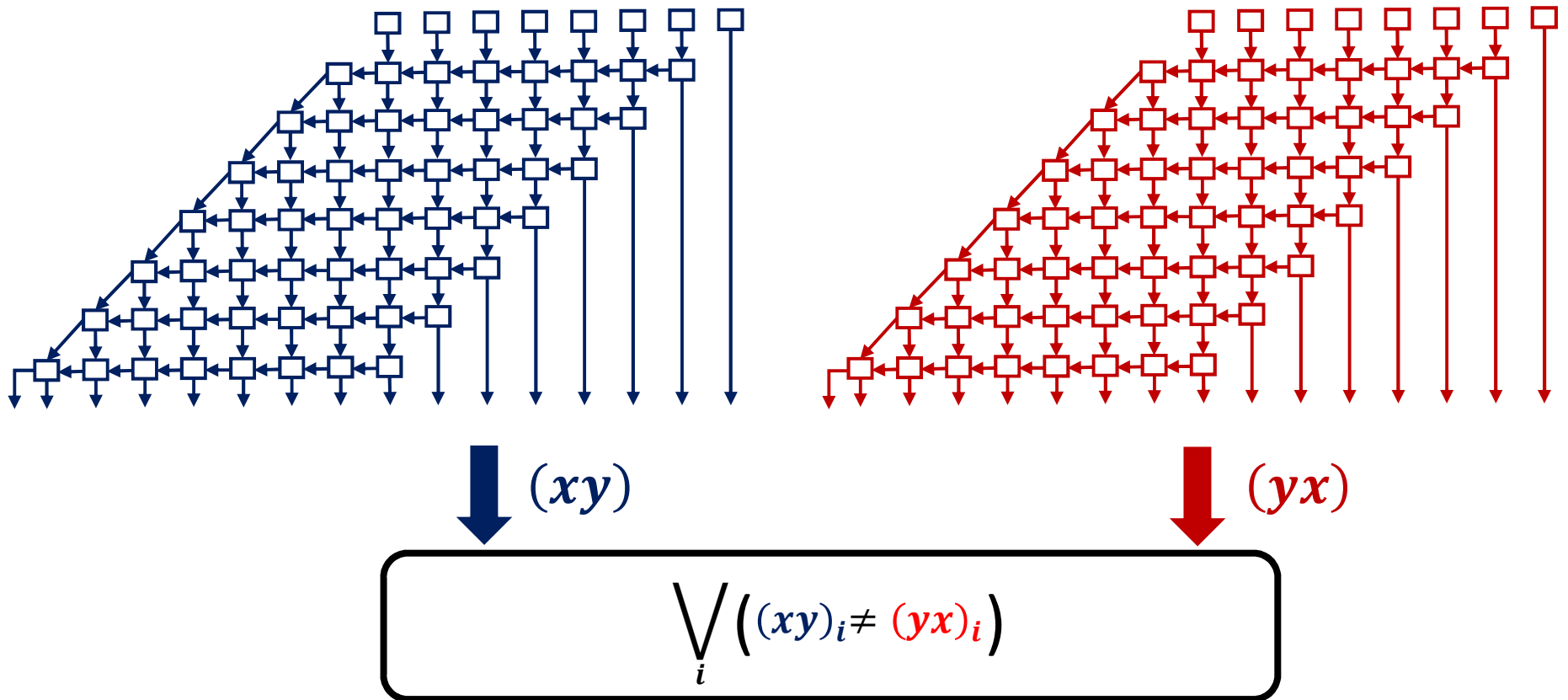
$$\begin{array}{r} x_3x_2x_1x_0 \\ \times y_3y_2y_1y_0 \\ \hline \end{array}$$



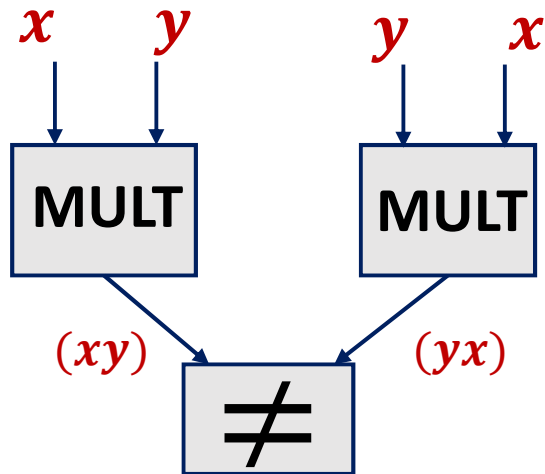
Example: Verifying array multiplier commutativity

Fix input bitwidth n .

Construct bit-blasted SAT formula encoding $xy \neq yx$ with array multipliers



Commutativity is hard for CDCL SAT solvers



- SAT formula: 100s-1000s of variables
- **SAT solvers cannot solve this with 16-bit-vectors for any multiplier circuit** [Biere, 2016].

Number of bits	Seconds to show $xy \neq yx$ unsat
5	0.01
6	0.2
7	0.5
8	11
9	43
10	743
11	Timeout

MiniSAT times

Arithmetic identities as indicators of complexity

Linear arithmetic

Easy to check:

$$x + y = y + x$$

(Commutativity)

$$(x + y) + z = x + (y + z)$$

(Associativity)

$$x \cdot 1 = x$$

(Multiplicative Identity)

Nonlinear arithmetic

Hard to check

$$x \cdot y = y \cdot x$$

(Commutativity)

$$x(y + z) = xy + xz$$

(Distributivity)

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

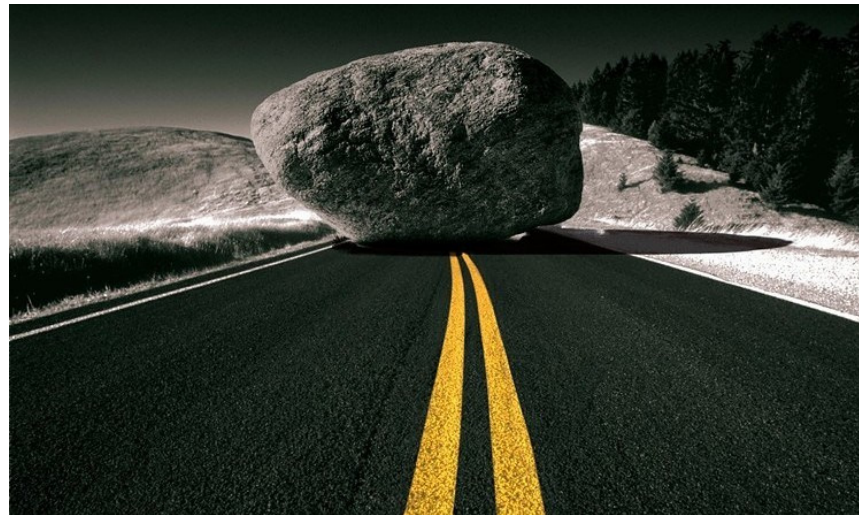
(Associativity)

Fundamental barrier? Or feasible with better SAT-solving?

Is resolution proof complexity a fundamental obstacle?

Conjecture: CDCL SAT solvers take **exponential time** to decide nonlinear arithmetic because resolution proofs require **exponential size**.

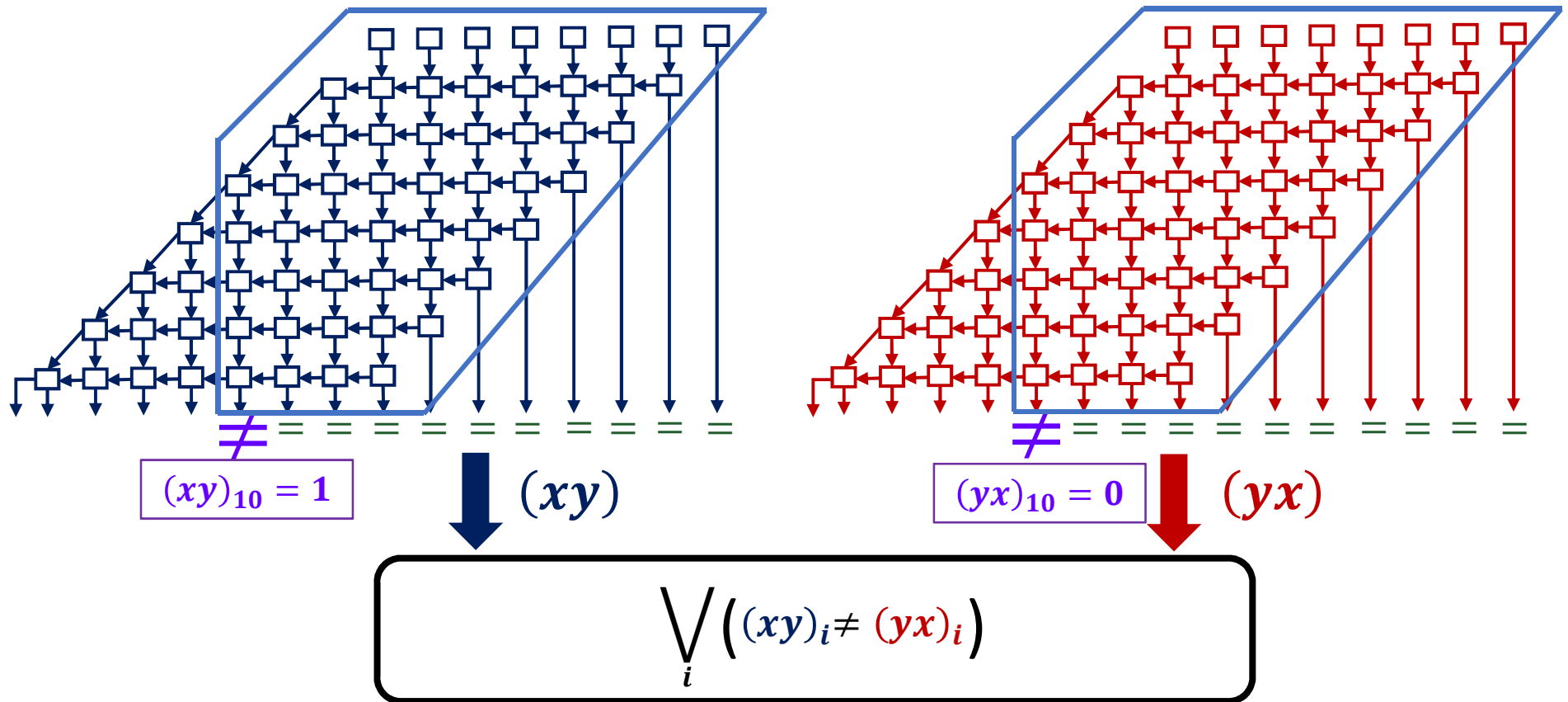
- [\[Biere SAT'16\]](#) [\[Slobodova SAT'16\]](#) [\[Tomb SAT'16\]](#) [\[Kalla FMCAD'15\]](#)



Example: Verifying array multiplier commutativity

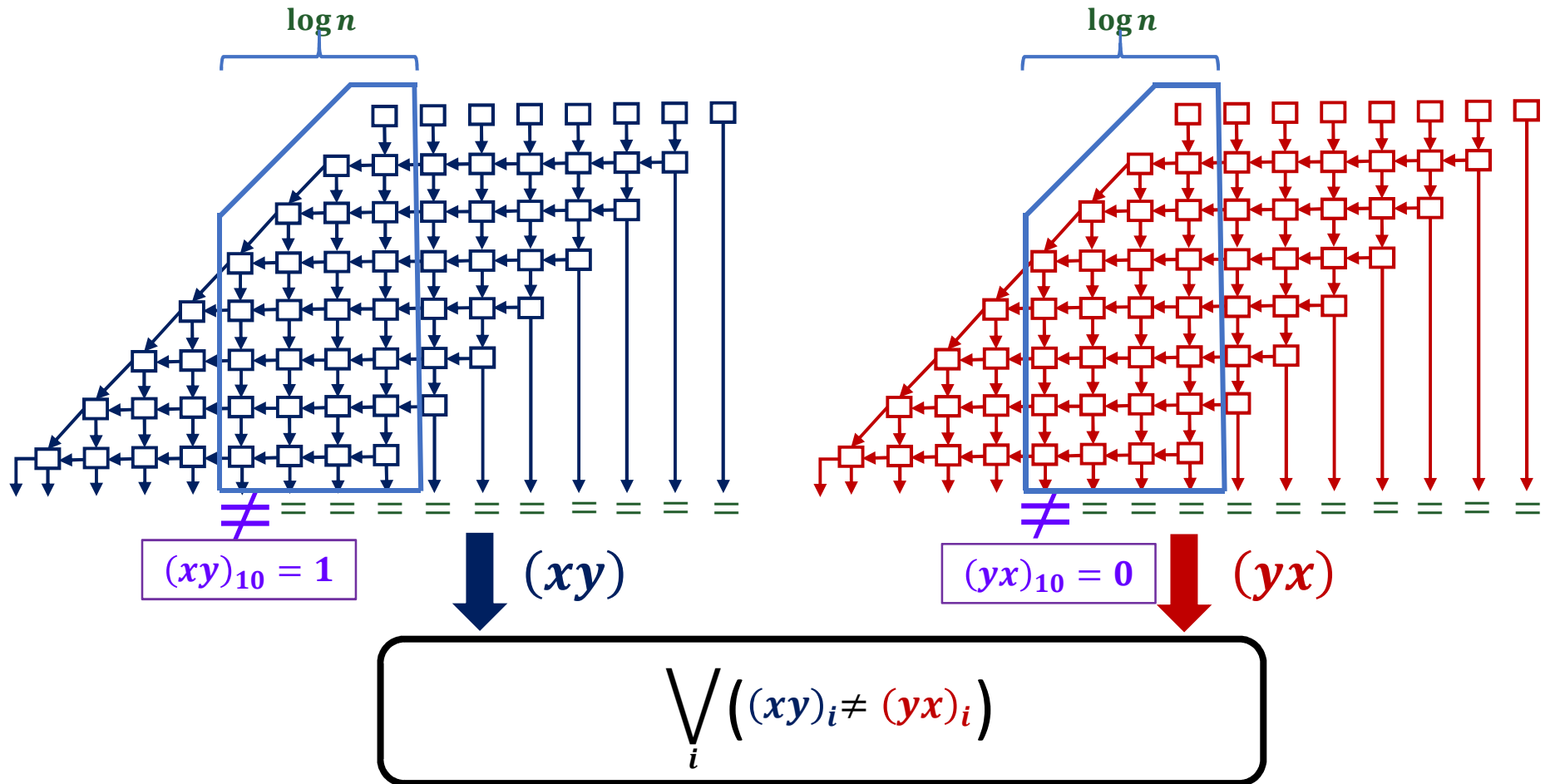
Start: branch to find first disagreeing output bit ($2n$ branches)

Issue: output sensitive to all previous tableau entries
so obvious proof is exponential



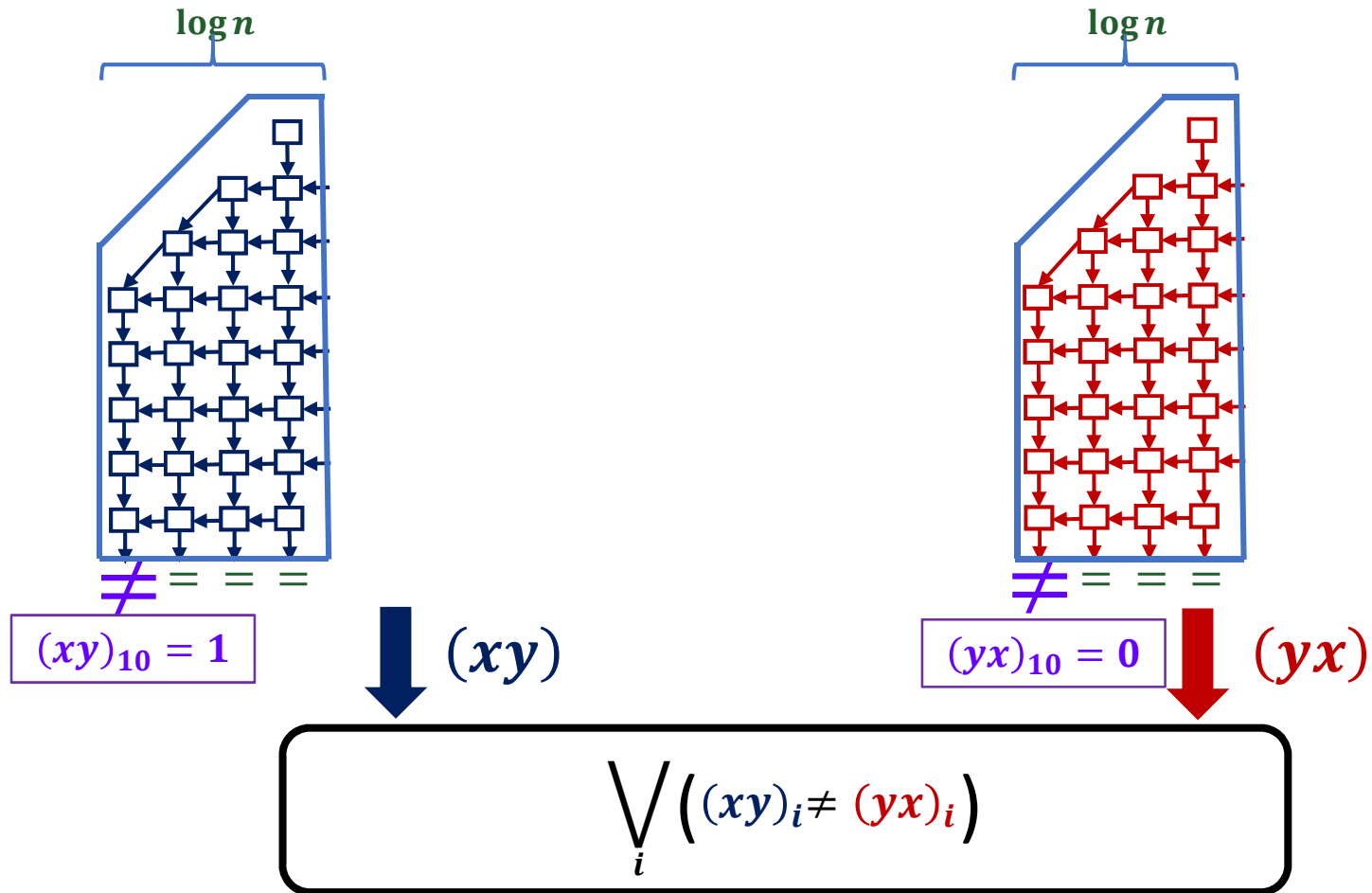
Example: Verifying array multiplier commutativity

Key Idea 1: The *critical strip* of the prior $\log n$ columns suffices for UNSAT



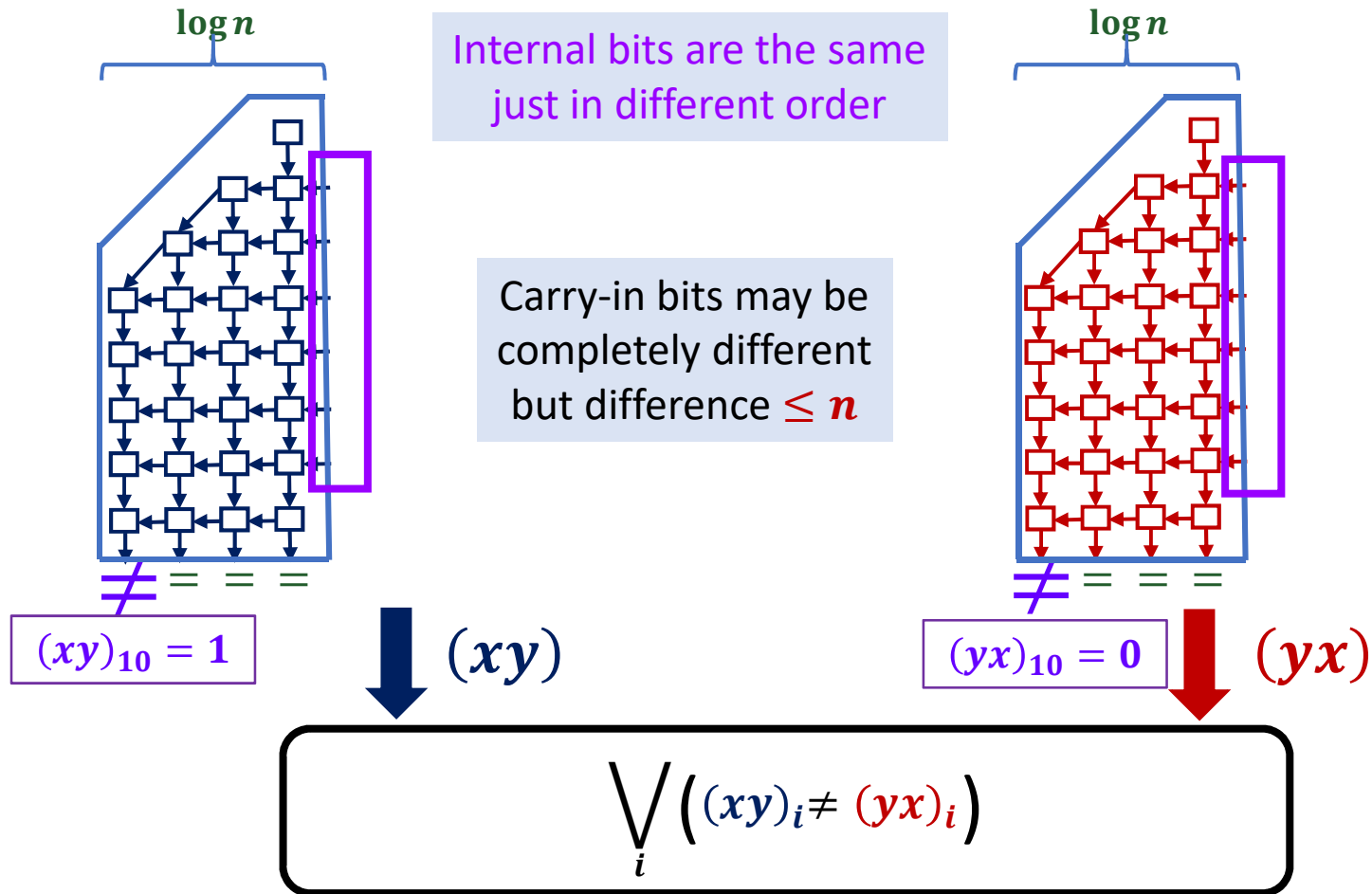
Example: Verifying array multiplier commutativity

Key Idea 1: The *critical strip* of the prior $\log n$ columns suffices for UNSAT



Example: Verifying array multiplier commutativity

Key Idea 1: The *critical strip* of the prior $\log n$ columns suffices for UNSAT

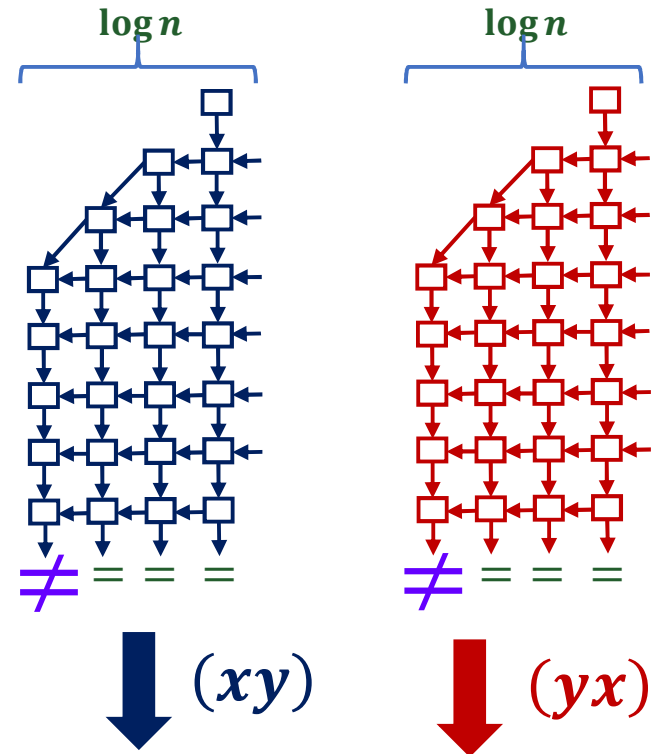


Example: Verifying array multiplier commutativity

- **Key Idea 2:** Each *critical strip* has **poly-size regular resolution refutations**

- **Why?**

- Follows from $O(\log n)$ pathwidth
- Resolution size at most exponential in pathwidth. [Dechter 1996]



Result: Polynomial-size resolution proofs

Theorem: For array, diagonal and Booth multipliers, there are polynomial size resolution proofs for any degree 2 identity.

[B, Liew CAV 2017, JACM 2020]

Identity	Proof size for bitwidth n
$x \cdot y = y \cdot x$	$\tilde{O}(n^6)$
$x \cdot (y + z) = x \cdot y + x \cdot z$	$\tilde{O}(n^6)$
$x \cdot (1 + x) = x^2 + x$	$\tilde{O}(n^{10})$

Compare with circuit size $O(n^2)$

Polynomial size → practical CDCL SAT solving?

Solvers:

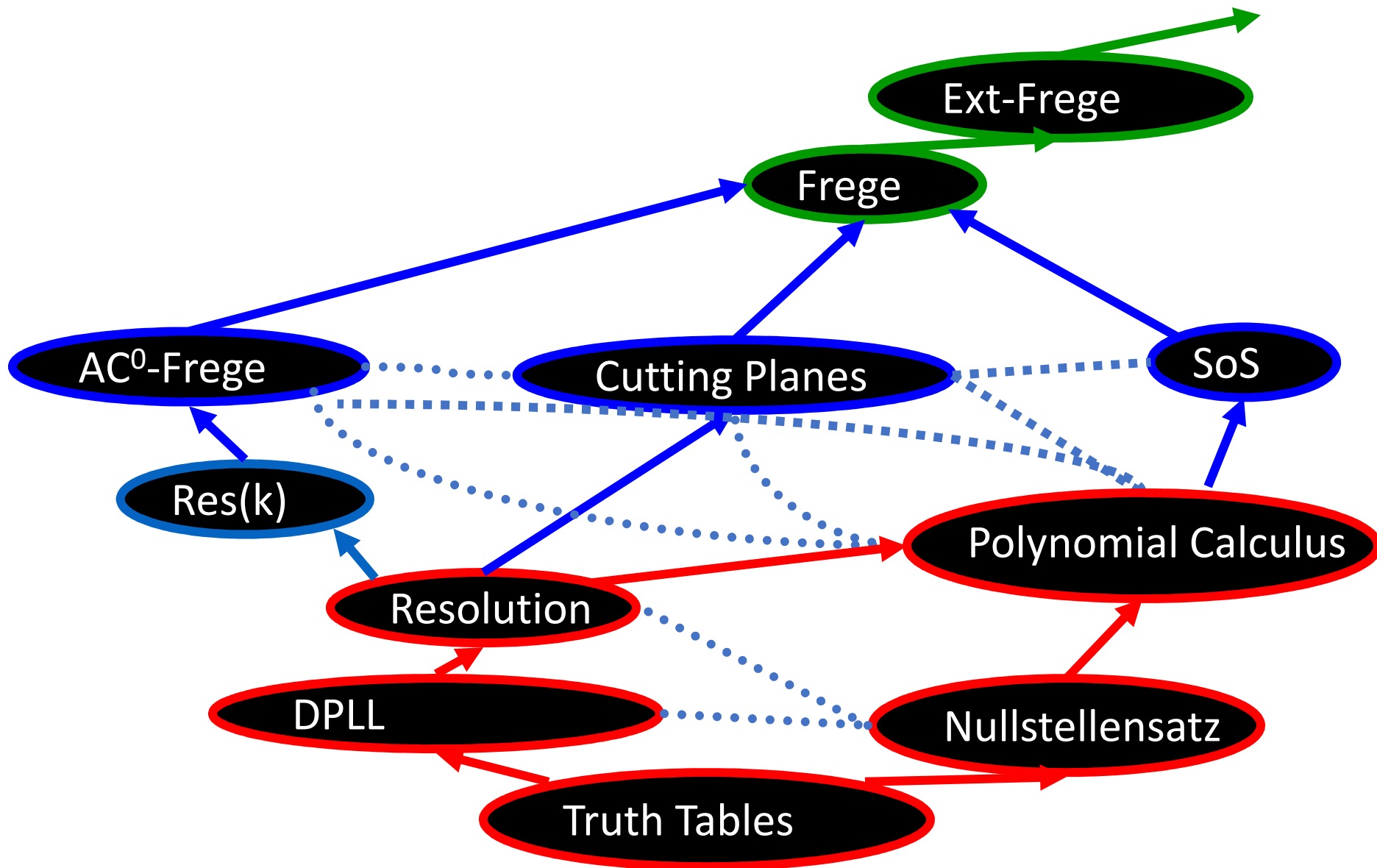
- Don't find these proofs even given the division into strips
- With the most extreme hand-holding (force-fed order, etc.) can't get any closer empirically than a factor of \sqrt{n}
- $\tilde{O}(n^6)$ proofs seem too large in any case.
- Target: **32** and **64** bits.

Stronger proof system?

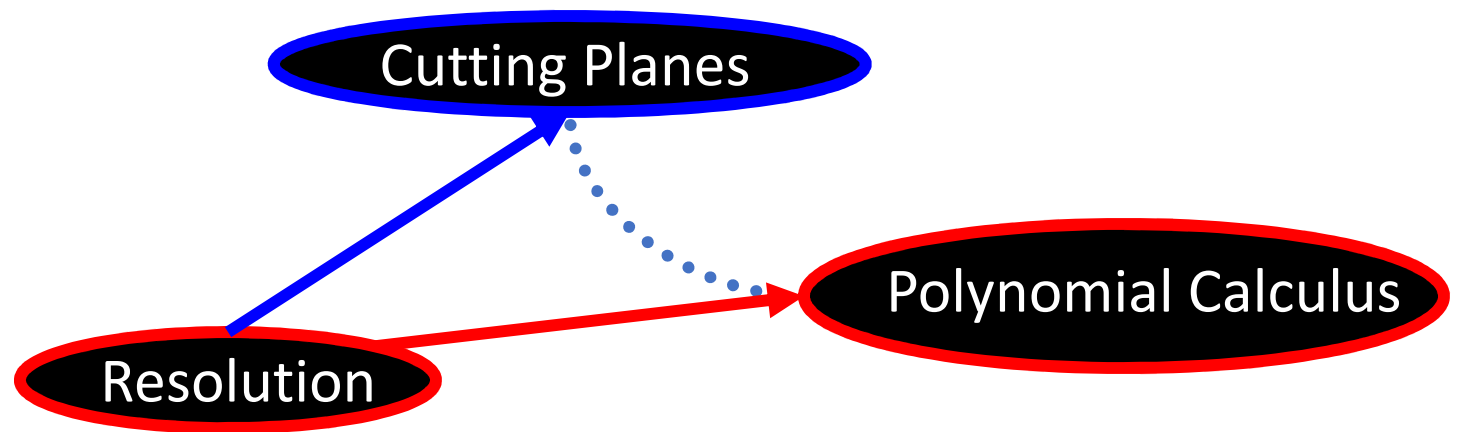
Number of bits	Strips $xy \neq yx$	Full $xy \neq yx$
5	0.01	0.01
6	0.1	0.2
7	0.7	0.5
8	3	11
9	26	43
10	146	743
11	1055	Timeout
12	5676	Timeout

MiniSAT: Strips vs Full

Stronger proof systems?



Stronger proof systems?



Beyond resolution: polynomial calculus

Polynomial Calculus (PC):

- Each line is a polynomial equation $p = 0$
- **Addition rule:** $p_1 = 0, p_2 = 0 \rightarrow p_1 + p_2 = 0$
- **Multiplication rule:** $p = 0 \rightarrow pq = 0$ for any polynomial q

Models steps of *Groebner basis reduction (GBR)* algorithms

- Checks if spec polynomial $p = 0$ implied by polynomials $p_1 = 0, p_2 = 0, \dots$

Polynomial calculus stronger than resolution \Rightarrow GBR more efficient than SAT?

- No for most **non-algebraic** problems.
- Yes for certain **algebraic** problems.
 - [Sayed et al., 2016]: Verified 128-bit integer multipliers.
 - [Kaufmann et al., 2017-2019]: Verified 1024-bit integer multipliers

Proof size in polynomial calculus

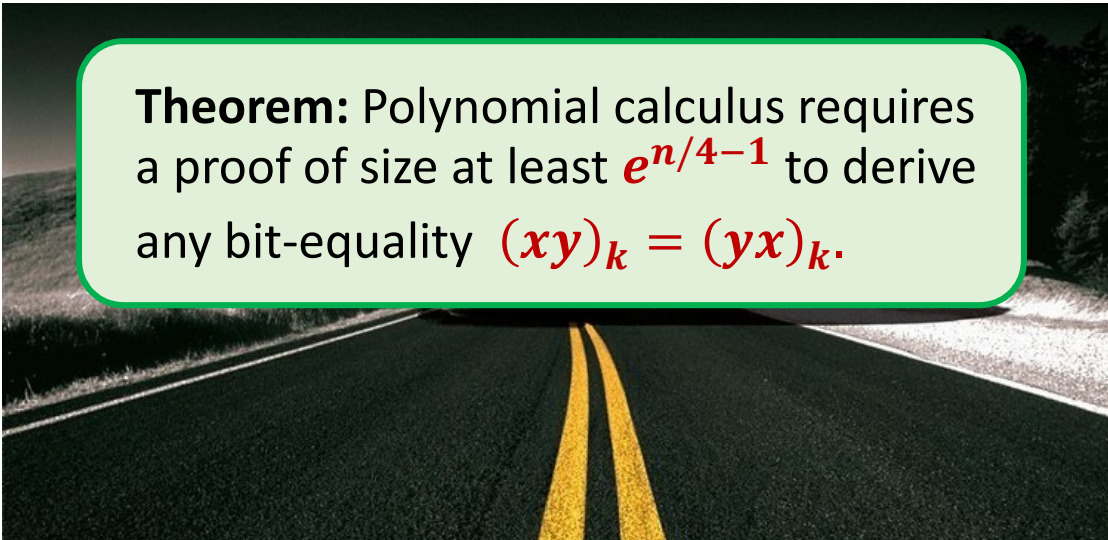
- [Kaufmann et al., 2019]: $O(n^2)$ length PC proof of *word-level* commutativity

$$\sum_{i=0}^{n-1} 2^i (xy)_i = \sum_{i=0}^{n-1} 2^i (yx)_i$$

- Idea generalizes to $O(n^2)$ length PC proofs of any word-level ring identity.

A Roadblock for Polynomial Calculus

[Liew, B, Devriendt, Elffers, Nordström, FMCAD 2020]



Theorem: Polynomial calculus requires a proof of size at least $e^{n/4-1}$ to derive any bit-equality $(xy)_k = (yx)_k$.

Beyond resolution: cutting planes

Cutting Planes Proofs:

- Each line l is Boolean linear inequality $\sum a_i x_i \geq b$
- Linear combination (non-negative):


$$l_1, l_2 \rightarrow \alpha l_1 + \beta l_2 \quad (\alpha, \beta \geq 0)$$

- Division:
$$\frac{\sum c a_i x_i \geq b}{\sum a_i x_i \geq \left\lceil \frac{b}{c} \right\rceil}$$

- Underlying proof system for the best *pseudo-Boolean solvers*

Cutting planes can extract bit-equalities!

Say we derive word-level equality $xy = yx$:

$$\sum_{i=0}^{n-1} 2^i (xy)_i = \sum_{i=0}^{n-1} 2^i (yx)_i$$


Two linear inequalities

Cutting planes can derive *all* n bit-equalities in $O(n)$ steps!

n	RoundingSat (Pseudo-Boolean)
32	.002
64	.009
128	.04
256	.2

n	Sat4j-Res (SAT)	NaPS (SAT)
16	3	2
20	81	39
24	TO	208
28		Error

And small cutting planes proofs at the word-level!

Theorem: There are $O(n^2)$ length cutting planes proofs for **word-level 2-colorable** ring identities. [Liew, et. al., FMCAD 2020]

2-colorable includes:

- $xy = yx$ (commutativity)
- $(x + y)z = xz + yz$ (distributivity)
- $(x + y)(w + z) = wx + yw + xw + zx$ (double distributivity)
- $x(y + z) + wz = xy + (x + w)z$ (distribute then factor)

Corollary: There are $O(n^2)$ length cutting planes proofs for **bit-level 2-colorable** ring identities. [Liew, et. al., FMCAD 2020]

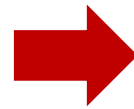
Key idea: we can do *nonlinear* reasoning within a *linear* proof system by using only a little nonlinearity at a time!

A nonlinear format for cutting planes proofs

Cutting Planes:

Linear inequality:

$$\sum a_i x_i \geq b$$



(k, d) -Cutting planes:

(k, d) -nonlinear inequality:

$$\underbrace{t_1 + t_2 + \dots + t_k}_{\text{Up to } k \text{ terms of degree } d \text{ or less}} + \sum a_i x_i \geq b$$

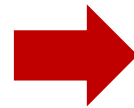
Up to k terms of degree d or less
(e.g. monomial, or $l_1 x_2 \dots x_d$)

- Linear combination rule:

$$l_1, l_2 \rightarrow \alpha l_1 + \beta l_2$$

- Division rule:

$$\frac{\sum c a_i x_i \geq b}{\sum a_i x_i \geq \left\lceil \frac{b}{c} \right\rceil}$$



- Linear combination rule:

Result must be (k, d) -nonlinear

- Division rule:

Generalizes immediately

- Multiply by variable rule:

$$5xy + w \geq b$$

$$\longrightarrow 5xyz + wz - bz \geq 0$$

Result must be (k, d) -nonlinear

Result: Simulating (k, d) -Cutting planes

Theorem: A (k, d) -cutting planes proof of s lines can be simulated by a standard cutting planes proof of at most $(k + 4)d^k \cdot s$ lines.

[Liew, et. al., FMCAD 2020]

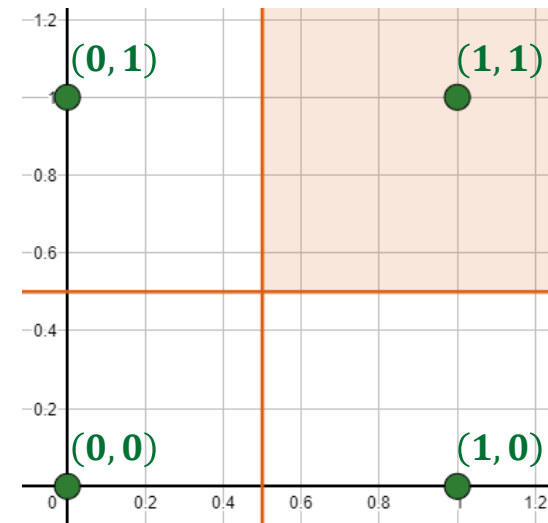
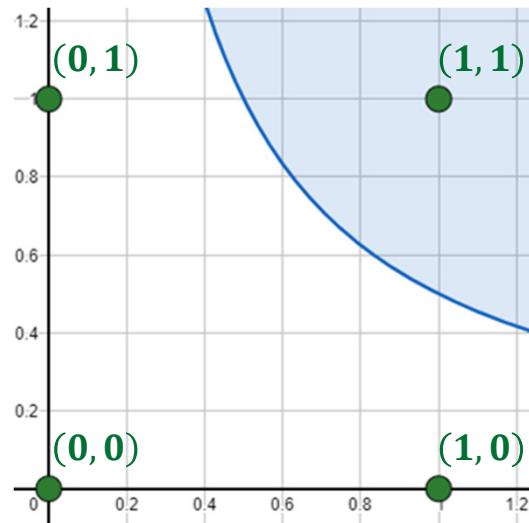
Simulation: Boolean (k, d) -nonlinear inequality $\leftrightarrow d^k$ linear inequalities.

E.g. $k = 1, d = 2$:

$$2xy \geq 1$$



$$\begin{aligned} 2x &\geq 1 \\ 2y &\geq 1 \end{aligned}$$



Simulating (k, d) -Cutting planes

Theorem: A (k, d) -cutting planes proof of s lines can be simulated by a standard cutting planes proof of at most $(k + 4)d^k \cdot s$ lines.

[Liew, et. al., FMCAD 2020]

Application: small proofs of 2-colorable identities

We give $O(n^2)$ length (k, d) -cutting planes proofs with k, d constant

Constant factor overhead simulation $\rightarrow O(n^2)$ proof in standard cutting planes

Finding cutting planes proofs via pB solvers

- **Pseudo-Boolean (PB) solvers Sat4j and RoundingSat**
 - **Sat4j**: Saturation-based, fast at proving word-level equalities
 - **RoundingSat**: Division-based, fast at extracting bit-level equalities.
- **Combination**: 256-bit commutativity for **bit-level!**
- 256-bit multiplier equivalence checking (e.g. array = diagonal)
- Requires value-based not clausal representation of 1-bit adders
- But cannot yet handle more complicated identities such as distributivity.

Number of bits	Array $xy = yx$	Array = Diagonal
32	21	15
64	43	34
128	117	91
256	419	338

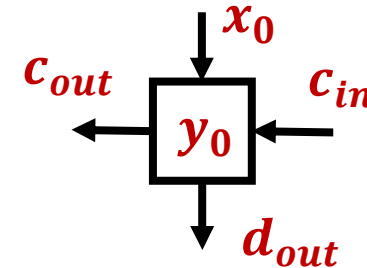
Sat4j + RoundingSat

Another approach to bit-blasting

The usual pseudo-Boolean advantage:

- Can represent full adder with equation:

$$2c_{out} + d_{out} = x_0 + y_0 + c_{in}$$



- Two inequalities instead of 14 clauses.

Even better pseudo-Boolean advantage:

- Can represent addition $x + y$ without a circuit:

$$\sum 2^i x_i + \sum 2^i y_i = \sum 2^i (x + y)_i$$

- Even further, can represent multiplication xy without a circuit!

n^2 Tableau constraints

$$t_{i,j} = x_i y_j$$

Tableau sum constraint

$$\sum 2^{i+j} t_{i,j} = \sum 2^i (xy)_i$$

i.e. $x_i - t_{i,j} \geq 0$; $y_i - t_{i,j} \geq 0$; $t_{i,j} - x_i - y_j \geq -1$

Beating bit-vector solvers

- With **algebraic representation** of multiplication, **RoundingSat** can outperform bit-vector solvers on crafted bit-vector inequalities.
- Inequalities mix multiplication and bit-wise operations.

Example: $kz \geq (x \& k)z$

Constant Bit-wise AND

bits	RoundingSat	Boolector	Z3
20	0.8	10	15
24	0.3	117	1154
28	0.5	TO	TO
32	0.6	TO	TO

Future directions

Pseudo-Boolean bit-vector solving

- Use preprocessing like CDCL/Bit-vector solvers
- Replace final SAT solver with PB solver.
- Algebraic bit-blasting
- **Can we get good performance on industrial benchmarks with multiplication?**

Improve cutting planes solving

- Pseudo-Boolean solving still young.
 - Could not solve more complicated 2-colorable identities.
- Crucial SAT solving improvements found over the last 25 years.
- **Can we get analogous improvements for pseudo-Boolean solvers?**