

Proof-Guided SAT Encoding Selection for Bit-Vector Multiplication

Armin Biere^{1,2}, Martin Brain³, Daniel Kroening^{4,5}, Norbert Manthey¹ and Michael Tautschnig^{1,6}

¹Amazon Web Services

²University of Freiburg, Germany

³City St George's, University of London, United Kingdom

⁴Annapurna Labs, Amazon

⁵University of Oxford, United Kingdom

⁶Queen Mary University of London, United Kingdom

Abstract

We present a proof-guided case study in designing SAT encodings of integer multiplication, driven by proof trace analysis. By examining DRAT proofs, we identify structural bottlenecks. For example the top bottleneck variable in standard Comba encoding appears in 39% of all proof steps. We then use these findings to redesign the encoding. This methodology produced COMBA-CS, a carry-save Comba encoding that separates column reduction from carry propagation, achieving up to 64× speedup across four SAT solvers. We further show that carry propagation is the dominant driver of CDCL search difficulty for multiplication on standard encodings—even though polynomial-size resolution refutations of commutativity provably exist [1]. A controlled experiment holds the partial-product layout and accumulation topology fixed and varies only the accumulator (integer addition vs. GF(2) XOR): on CaDiCaL, integer commutativity is hundreds of times harder than GF(2) at bit-width 9, and structural variations (sequential vs. parallel-tree accumulation) change running time by less than 5×. The gap is between *existence* of short proofs and the ability of CDCL search to find them, and it is robust: neither supplying the known polynomial-size lemmas nor reordering the solver's decisions closes it. We instead address it by engineering encodings that guide CDCL toward those proofs. Comparing COMBA-CS with pre-existing alternative encodings (Booth radix-4, 4-bit blocks, sorting networks), we find that no single encoding dominates: Booth is up to 32× faster on strength-reduction benchmarks, while COMBA-CS is 2× faster on bfloat16 commutativity. The optimal encoding depends on problem structure, motivating adaptive and hybrid selection.

Keywords

SAT encoding, bit-vector SMT, multiplication, DRAT proof analysis, bounded variable elimination

1. Introduction

SMT solvers that support the theory of bit-vectors are widely used for tasks such as bounded and unbounded model checking, IC3-style invariant inference, hardware equivalence checking, cryptographic circuit analysis, and symbolic execution. Bounded model checkers such as CBMC [2] are a mainstay of this landscape: they reduce program-correctness questions—array-bounds, overflow, assertion and pointer safety—to bit-vector formulas discharged by a SAT solver, and are deployed routinely on production software and hardware. Their end-to-end performance is therefore gated by how efficiently the back-end solver handles the arithmetic these formulas contain, and multiplication is the dominant bottleneck. A popular technique is *bit-blasting* [3, 4]—reducing formulas in the theory of bit-vectors to SAT. For formulas involving integer multiplication—common in cryptography, DSP, overflow checking, and compiler verification—the encoding choice dramatically affects solver performance. Yet the interaction between encoding choices and modern SAT solver techniques (inprocessing [5], BVE [6]) is poorly understood. Prior work has focused on BDD-based complexity [7] and propagation completeness [8, 9], but not on systematic evaluation of SAT encodings across multiple solvers with solver-level proof analysis.

17th Pragmatics of SAT International Workshop 2026 (PoS'26)

0000-0001-7170-9242 (A. Biere); 0000-0003-4216-7151 (M. Brain); 0000-0002-6681-5283 (D. Kroening); 0000-0002-7947-983X (M. Tautschnig)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Consider verifying that 16-bit multiplication commutes: does $a \times b = b \times a$ hold for all 16-bit inputs a and b ? A bit-blasting SMT solver encodes this as a CNF formula with two multiplier circuits and an equality check. With the standard shift-and-add encoding, CADICAL [10] times out after 120 s. With our carry-save Comba encoding, it solves in 5.4 s—a speedup of at least $22\times$ from changing only the encoding (the actual ratio is larger because the shift-add baseline did not finish; we bound it conservatively by the timeout).

How did we arrive at this encoding? Not by theoretical analysis of circuit complexity, but by a systematic methodology: generate a DRAT proof of the existing encoding, identify which variables participate most heavily in the proof, trace those variables back to the encoding structure, and redesign to reduce the identified bottlenecks. This proof-guided approach is transferable to other encoding problems and is the methodological thread of this paper.

Our contributions are:

1. **A DRAT-guided design case study.** We apply DRAT proof-trace analysis to a single encoding-design case (standard Comba $\rightarrow$ COMBA-CS), following the sequence DRAT analysis $\rightarrow$ bottleneck identification $\rightarrow$ structural hypothesis $\rightarrow$ encoding redesign $\rightarrow$ cross-solver validation, and demonstrate the same analysis lens retrospectively on three pre-existing encodings (Booth, 4-bit blocks, sorting networks). This is a detailed case study, not a general methodology claim; the sequence is reproducible and transferable, and we find it sufficient to explain both successful and unsuccessful encodings in our evaluation (Section 4).
2. **Carry propagation as the causal driver of CDCL search difficulty.** We combine a GF(2) comparison with a controlled five-variant experiment holding topology fixed and varying only the accumulator operator: on CaDiCaL, integer multiplication is orders of magnitude harder than GF(2) on structurally identical formulas at $BW \geq 8$, while topology variations change running time by less than $5\times$. This isolates the cause of the *empirical* difficulty, rather than of proof-theoretic hardness—Beame and Liew have shown that polynomial-size regular resolution proofs of commutativity exist for array, diagonal, and Booth multipliers [1]. The practical gap is that CDCL does not find those proofs without encoding-level guidance, and we show this is robust—neither supplied lemmas nor reordered decisions close it (Sections 3, 8).
3. **Eight-encoding comparison with solver statistics.** We evaluate shift-add, Dadda, Wallace, Comba, COMBA-CS, Booth radix-4, 4-bit blocks, and sorting networks across four SAT solvers, reporting conflicts, propagations, BVE eliminations, fixed variables, and proof sizes. Results are presented with statistical variance, multi-layer ablation, and adaptive-heuristic ablation; no single encoding dominates, and Wallace and Dadda — both tree-based reductions — differ by up to $14\times$ in solve time (Sections 5, 6, 7).
4. **Negative results.** We catalogue 20+ approaches that failed, including subquadratic algorithms, XOR Gaussian elimination, and abstraction refinement (Section 8).

For this paper the algebraic pre-solver is disabled, isolating the encoding’s contribution. In the deployed configuration, the encoding matters primarily for problems the algebraic layer cannot handle: bit-wise operations, inequalities, floating-point, and structural mismatches. The same encoding techniques apply to division and remainder, which can be encoded as $a = b \times q + r$ with non-deterministic q, r .

2. Background

An N -bit unsigned multiplication $a \times b$ produces N partial products $pp[i] = a[i] \wedge b$, each shifted by i positions. Arranged in a $N \times 2N$ grid where row i is $pp[i]$ shifted into columns $[i, i + N)$, the multiplier must sum along the columns (diagonals of the original grid) to produce the $2N$ -bit product. The standard *shift-add* encoding adds partial products sequentially using a ripple-carry adder chain, creating a carry chain of length $O(N^2)$. *Dadda trees* [11] and the earlier *Wallace trees* [12] reduce the partial-product grid to two rows using carry-save full adders, followed by a single carry-propagate addition; Dadda differs from Wallace by deferring reductions until the column height forces them,

yielding fewer full and half adders than Wallace. *Comba* [13] reduces each column independently using popcount (parallel bit counting via the pop0 algorithm from [14]¹), propagating carries to adjacent columns during the same pass. Combining column-wise popcount with immediate carry propagation couples two otherwise independent sources of structure in the same set of clauses, which limits how aggressively downstream CDCL inprocessing can simplify the encoding. Section 5 introduces our carry-save Comba variant that decouples these two steps by first reducing every column to a small set of sum/carry bits independently, and only then propagating carries in a second pass.

Modern CDCL solvers employ techniques that interact with encoding design in ways we exploit throughout this paper. Bounded Variable Elimination (BVE) [6] eliminates variables by resolving all clauses containing them; it is effective when the resolvent count is small. CADICAL [10] performs BVE during search (*inprocessing*), while MINISAT/SatELite [6] performs it only during preprocessing. CADICAL also extracts XOR and AND gates from CNF and identifies congruent gate pairs between multiplier circuits, underpinning the congruence-closure benefits we exploit in Section 5.

Brain et al. [8] gave an algorithm for computing propagation-complete (PC) primitives—a property known as *arc consistency* in the constraint-programming community—and used it to construct a propagation-complete full-adder, which can be chained to form PC adder encodings. However, PC composition in general does not preserve PC, so this construction does not extend directly to PC multipliers. Brain [9] conjectured that a PC multiplier encoding would require a number of clauses that is exponential in the bit-width; indeed, a polynomial-size propagation-complete encoding of multiplication would run counter to common beliefs about the hardness of integer factoring. Rather than seeking propagation completeness, we optimise encoding structure for interaction with BVE and inprocessing.

These three elements—partial-product accumulation strategies, the inprocessing techniques of modern CDCL, and propagation completeness as a theoretical target—set the context for the rest of the paper. We do not design a new multiplier algorithm from scratch; instead we take the known column-wise reduction strategy and restructure its clause-level output so that inprocessing BVE and congruence-closure can exploit it. Section 3 first justifies why this is worth doing, by isolating carry propagation as the empirical driver of CDCL search cost for integer multiplication.

3. Carry Propagation Hardness

We compare integer multiplication commutativity ($a \times b = b \times a$) with GF(2) (carry-less) multiplication commutativity. Both have identical partial-product grids; the only difference is carry propagation (integer uses addition with carries, GF(2) uses XOR). To be precise, the SAT solver never reasons about “integers”: in both cases we bit-blast the corresponding bit-vector multiplier circuit—standard two’s-complement multiplication modulo 2^d in the integer case, carry-less (XOR-based) multiplication in the GF(2) case—to a CNF formula, and CADICAL solves the CNF. The two formulas differ only in how the partial products are summed. This comparison isolates carry propagation as a variable, though we note that the change in addition operation also affects clause structure and propagation properties.

Table 1

Integer vs. GF(2) multiplication commutativity (CADICAL 3.0.0), using the controlled sequential benchmarks (E0/E1 of Table 2): identical partial-product grids, differing only in bvadd vs. bvxor accumulation.

BW	GF(2)			Integer		
	Clauses	Conflicts	Time	Clauses	Conflicts	Time
7	708	1,452	0.02s	1,512	40,523	1.03s
8	920	2,875	0.04s	2,000	171,387	5.62s
9	1,160	3,686	0.05s	2,556	940,195	37.7s

¹Each of the $\lceil \log_2 N \rceil$ accumulation steps of pop0 is a ripple-carry bvadd; the masked field layout keeps the effective carry chains short, and inprocessing BVE eliminates most of the resulting auxiliary variables at the bit-widths we consider.

The gap widens sharply with bitwidth (Table 1): integer multiplication needs $28\times$ more conflicts than GF(2) at BW=7, $60\times$ at BW=8, and $255\times$ at BW=9, despite only $\sim 2.2\times$ more clauses. At $BW \geq 10$ the integer formula times out at 120 s while GF(2) stays under 0.1 s. GF(2) conflict counts grow gently over the bit-widths we tested while integer counts explode—and the DRAT proofs shrink correspondingly.

The gap is between proof existence and CDCL search. It is important to be clear about what this data does and does not mean. Beame and Liew [15, 1] prove that ring identities such as commutativity and distributivity on array, diagonal, and Booth multipliers admit *polynomial-size regular resolution refutations*—the very proof system that CDCL solvers operate in. A slightly weaker quasi-polynomial bound is known for Wallace tree multipliers. The gap we observe is therefore not that short proofs do not exist, but that current CDCL solvers do not find them on the standard shift-add encoding. Beame and Liew themselves remark on this gap and call it an open engineering problem: “the observed scaling of SAT solver performance on these problems suggests that they do not currently find proofs matching even these upper bounds. An important direction for improving SAT solvers is to find the right guiding information to add, either to the formulas derived from the circuits or to CDCL SAT solver heuristics, to help them find shorter proofs” [1]. Our paper is squarely in the “guiding information via encoding structure” direction: we engineer encodings (COMBA-CS, Booth, 4-bit blocks) that let CDCL find the short proofs on instances where the standard shift-add encoding does not.

The short proof exists but guidance does not reach it. Before acting on encoding structure, we tested the two routes Beame and Liew suggest, on the array-multiplier commutativity CNF with the same solver (CADICAL), at the bitwidths where it blows up (822k conflicts at $n=9$, timeout at $n \geq 10$). *Adding information to the formula:* asserting the output bit-equalities directly collapses the instance to zero conflicts, but the refutation never touches the multiplier—it holds verbatim when the two products are replaced by free variables, so it asserts the conclusion and proves nothing about search. Guidance that keeps the circuit in the proof does not help—the n^2 true partial-product correspondences do not reduce the conflict count, and even the restricted Beame–Liew strip—a local slice on which a polynomial refutation provably exists—still costs $\sim 10^5$ conflicts on a few hundred variables. *Guiding the heuristics:* renumbering the variables into strip order and forcing the decision phase changes runtime by at most $\sim 1.4\times$ and stays exponential. The short proof is real, yet neither added lemmas nor reordered decisions let CDCL find it (Section 8). One cannot fix a bad encoding by supplying it true lemmas or reordering its variables; this is what motivates acting on the encoding’s *structure*.

From correlation to controlled experiment. The integer-vs-GF(2) comparison changes three things at once (operator, clause structure, variable dependencies). To isolate the operator, we built five SMT-LIB families with identical partial-product layouts and accumulation topologies, varying only the reduction operator on shifted partial products $pp_0, \dots, pp_{N-1}$ reduced to $2N$ bits: **E0** (integer sequential): left-to-right bvadd; **E1** (GF(2) sequential): same topology with bxor; **E2** (integer parallel-tree): balanced bvadd tree of depth $\lceil \log_2 N \rceil$; **E3** (GF(2) parallel-tree): same tree with bxor; **E4** (integer ternary): threes grouped sequentially, two dependent bvadds per group through the carry chain.

Table 2

Controlled accumulator experiment on commutativity (CADICAL, algebraic pre-solver disabled, 120 s T/O). Boldface marks the GF(2) winner in each row.

BW	E0 int-seq	E1 GF(2)-seq	E2 int-par	E3 GF(2)-par	E4 int-tern.
6	0.17 s	0.015 s	0.16 s	0.011 s	0.17 s
8	5.65 s	0.037 s	5.78 s	0.016 s	5.64 s
10	T/O	0.071 s	T/O	0.037 s	T/O
12	T/O	0.132 s	T/O	0.044 s	T/O
14	T/O	0.344 s	T/O	0.064 s	T/O
16	T/O	0.244 s	T/O	0.111 s	T/O

Results are in Table 2. At BW=8 integer sequential takes 5.65 s with 171K conflicts; GF(2) sequential solves the *structurally identical* formula in 37 ms with 2.9K conflicts ($153\times$ speedup), and at BW ≥ 10 integer times out at 120 s while GF(2) stays under 500 ms. Topology barely matters: sequential to parallel-tree changes integer by $<10\%$ (E0 vs E2) and helps GF(2) only $2\text{--}5\times$ (E1 vs E3), and ternary E4 tracks E0 within 1%—the solver sees a cascade of 2-input adds regardless of grouping. The ordering shift-add $<$ Dadda $<$ Comba $<$ COMBA-CS in Table 7 mirrors decreasing carry-propagation depth at constant partial-product count, reinforcing carry propagation as the dominant driver of CDCL search difficulty.

Scope. The experiment establishes that on commutativity and on CADICAL, carry presence dominates the choice of accumulation topology; it does not establish a proof-theoretic lower bound, given the Beame–Liew upper bounds cited above.

4. Designing COMBA-CS through DRAT Analysis

COMBA-CS was designed through a systematic methodology based on examining solver proof traces. We describe the procedure here both as a record of how the encoding was found and as a reproducible technique for future encoding work.

1. **Generate a DRAT proof** of the existing encoding on a representative hard benchmark (we used commutativity at BW=11 with standard Comba).
2. **Count variable participation:** for each variable, count the number of proof steps in which it appears as a literal. Rank variables by participation count.
3. **Trace the top variables** back to the multiplier circuit: which partial product bits, carry bits, or intermediate gates do they represent?
4. **Hypothesise a structural cause** for the concentration: what aspect of the encoding forces the solver to repeatedly reason about these variables?
5. **Propose an encoding modification** that breaks the identified dependency without losing correctness.
6. **Validate** the modification across multiple solvers and benchmarks to ensure the improvement is not solver-specific.

Applying this to standard Comba on commutativity BW=11, the top variable by proof participation appears in 39% of all DRAT steps.² Tracing it back to the encoding, we found it represents a carry bit at the boundary between adjacent columns—the point where standard Comba propagates the carry from one column’s popcount to the next during the same pass. This inter-column dependency forces the solver to reason about carry propagation across the entire width, creating a sequential bottleneck in an otherwise parallel structure.

The proposed fix—COMBA-CS—separates column reduction (first pass) from carry propagation (second pass), eliminating the inter-column dependency during the popcount phase. The validation step showed: 72% smaller proofs, 73% more unit-propagation-fixed variables during search, and up to $64\times$ speedup (Table 7) across four different SAT solvers.

Retrospective analysis of known encodings. We did not design Booth radix-4, 4-bit blocks, or sorting networks via this methodology—these are pre-existing ideas (Booth from hardware arithmetic [16]; 4-bit blocks from earlier unpublished work by one of the authors; sorting networks from the synthesis literature). The same DRAT analysis explains why each wins or loses on specific patterns (Section 6).

²We measure participation over the *full* DRAT proof emitted by the solver, not over a trimmed proof core (the subset of steps a checker such as `drat-trim` retains as needed to derive the empty clause). The share over the core may differ; a core-level analysis is a natural refinement we leave to future work, and throughout the paper “proof steps” means full-proof steps.

5. Carry-Save Comba Encoding

Standard Comba processes columns left-to-right, propagating carry bits from each column’s popcount to the next column during the same pass. This creates inter-column dependencies that manifest as proof bottlenecks: the top bottleneck variable appears in 39% of all DRAT proof steps.

COMBA-CS separates the two concerns (Figure 1):³

1. **First pass:** Reduce each column independently via popcount. Carry bits are collected but *not* propagated to adjacent columns.
2. **Second pass:** Reduce the accumulated carry bits (which form much shorter columns, at most 4 for the bit-widths we test).

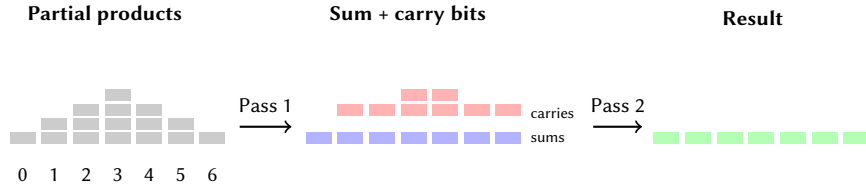


Figure 1: The COMBA-CS two-pass encoding for a 4-bit multiplier. Pass 1 reduces each column independently via popcount (no inter-column carries). Pass 2 reduces the short carry columns (max height 2 here; at most 4 even for the largest bit-widths we test).

Counterintuitively, COMBA-CS produces $\sim 2\times$ more clauses than Dadda yet is up to $\sim 9\times$ faster, solving bit-widths where Dadda times out: the popcount auxiliary variables act as “summary” abstractions (counts of 1-bits in groups of 2, 4, 8) the solver reasons about at a higher level. Formula size is not the right optimisation target—propagation structure is.

5.1. Adaptive Encoding Selection

The interaction between COMBA-CS and inprocessing Bounded Variable Elimination is pattern-dependent, which motivates an adaptive rather than one-shot choice. We observe three distinct regimes for COMBA-CS interacting with CDCL’s BVE (as measured on CADICAL):

- *Two-multiplication commutativity pattern* ($BW \leq 11$). Popcount intermediate variables are structurally aligned with the congruence-closure pairs CADICAL identifies between the two multiplier circuits. BVE cleans up the matched pairs and COMBA-CS benefits by large factors (up to $64\times$, §7).
- *Three-or-more multiplication patterns* (associativity, distributivity, matrix products). These benefit from the carry-chain cascading that *shift-add* exposes to BVE: *shift-add* reaches 120% BVE elimination on *assoc_8* (resolution creates new variables that are themselves eliminated in later rounds, so the cumulative count exceeds the original—the formula is eliminated entirely), whereas popcount-based carry-save encodings leave a substantial residual for search.
- *Wide single multiplication* ($BW \geq 11$). Inprocessing BVE attempts to resolve away COMBA-CS’s popcount variables but produces longer resolvents than the originals; we measure 4–140% overhead relative to BVE-disabled (Section 8.4). The adaptive heuristic does not disable BVE at solver-level because the resulting residual benchmark is still faster with COMBA-CS than with *shift-add*.

A pre-scan counts symbolic multiplications: the heuristic selects *popcount* when there are ≤ 2 of the same width (the commutativity pattern where congruence closure discovers equivalent gates between identical circuits) and *shift-add* otherwise (3+ multiplications, or differing widths). For constant multiplication with few partial products, COMBA-CS falls back to Dadda-style carry-save reduction.

³The complete clause-level encoding—partial-product layout, per-column popcount blocks, and the second carry-reduction pass—is implemented in CBMC’s `src/solvers/flattening/bv_utils.cpp` (`comba_carry_save`), which is the authoritative specification and reproducibility reference for the encoding described here.

The threshold is insensitive (1, 4, or 99 give identical results): polynomial benchmarks are already canonicalised by CBMC’s word-level simplification, and the non-polynomial ones in our suite have ≤ 2 symbolic multiplications.

Scope of the classifier. The classifier keys only on the count and width of the symbolic multiplications, leaving all other gates (bitwise, comparisons, ITE) untouched, so correctness is independent of the rest of the formula. It is a deliberately simple count/width baseline; richer features (constant detection, structural matching) are plausible but untested. Its logic is solver-agnostic, but its integration is not—it fires inside CBMC’s bit-blasting layer, and porting it to Bitwuzla or cvc5 would require those solvers to expose the multiplier-encoding choice, which they currently do not.

Ablation of the adaptive heuristic. An ablation against the alternatives “popcount always” and “shift-add always” on commutativity, associativity, and distributivity benchmarks (algebraic pre-solver disabled) confirms the two design choices: on commutativity (two symbolic multiplications) popcount is decisive—at BW=11 it is $217\times$ faster than shift-add and at BW=12 it avoids a timeout; on three-multiplication problems (assoc, distrib) the three configurations perform identically, confirming that the heuristic’s gain on the targeted pattern is matched by zero cost elsewhere. DRAT proof analysis at BW=11 (Table 3) confirms the benefit of column independence at the proof level.

At BW=11, COMBA-CS produces a 72% smaller DRAT proof—and a 74% smaller *proof core* (the subset of lemmas `drat-trim` retains to derive the empty clause)—while doing substantially less search: 77% fewer conflicts ($86,405 \rightarrow 19,931$) and 76% fewer decisions ($160,713 \rightarrow 38,627$). The 73% increase in unit-propagation-fixed variables ($84 \rightarrow 145$) lets the solver determine more variable values through unit propagation alone, shrinking the search space. The effect is not specific to BW=11: Table 3 tracks the same DRAT measurement across BW=7–15, and the proof shrinks at every bitwidth, in both the full proof (26% at BW=7 to 84% at BW=15, widening monotonically) and the proof core (34% to 67%, peaking at 76% at BW=13). The unit-propagation/fixed-variable advantage, by contrast, is noisy across bitwidths and we read no trend into it. This confirms that column independence reduces inter-column reasoning.

Table 3

DRAT proof-size scaling, commutativity BW=7–15 (CADiCAL 3.0.0): full-proof lemmas and proof-core lemmas, standard Comba vs. COMBA-CS.

BW	Proof steps (lemmas)			Proof core	
	Std. Comba	COMBA-CS	Change	Std. Comba	COMBA-CS
7	4,295	3,171	−26%	3,225	2,123
9	15,904	9,146	−42%	10,354	4,908
11	106,465	30,246	−72%	47,597	12,212
13	205,989	55,482	−73%	77,062	18,382
15	779,927	127,457	−84%	133,151	44,368

5.2. Multi-Encoding as Hybrid Selection

We also evaluated a *multi-encoding* mode that emits two encodings of the same multiplication with equality constraints tying their outputs, letting CDCL arbitrate. It is slower than the best singleton on 58% of 180 configurations (median 30% overhead)—not a selection strategy but effectively another encoding, dominated by the singletons it combines.

6. Alternative Encodings

Three alternative encodings from the literature were implemented in CBMC and evaluated alongside COMBA-CS: Booth radix-4 from hardware arithmetic [16], 4-bit block multiplication (unpublished,

proposed by one of the authors), and sorting networks adapted from counting-based synthesis. For each, we use the same DRAT proof analysis that guided COMBA-CS’s design to *understand* why the encoding helps or hurts on specific benchmark patterns. These encodings were not designed via proof analysis; they are pre-existing ideas whose interaction with SAT solvers we characterise here.

6.1. Booth Radix-4

Booth encoding [16] examines multiplier bits in pairs, generating partial products with values in $\{-2, -1, 0, +1, +2\} \times$ multiplicand. This halves the number of partial products at the cost of two’s complement correction. We initially expected Booth to excel broadly on constant multiplication, but a sweep over two families (algebraic solver disabled, Table 4) gave a more nuanced picture:

Table 4

Booth vs. shift-add with algebraic solver disabled, on representative benchmarks (CADICAL).

Pattern	Benchmark	shift-add	Booth	Ratio
<i>Commutativity of constant mul: $x \cdot c = c \cdot x$</i>				
	const_mul_16_3	0.007s	0.132s	0.05×
	const_mul_16_15	0.014s	0.206s	0.07×
	const_mul_16_255	0.389s	0.268s	1.5×
	const_mul_32_15	0.056s	T/O	$\ll 1 \times$
<i>Strength reduction: $x \cdot (2^{k+1} - 1) = (x \ll (k+1)) - x$</i>				
	strength_16_7	0.029s	0.017s	1.7×
	strength_16_15	0.041s	0.014s	2.9×
	strength_16_31	0.115s	0.014s	8.2×
	strength_32_31	0.358s	0.022s	16×
	strength_32_63	0.801s	0.025s	32×
	strength_chain_16	0.930s	0.039s	24×

Two observations. First, Booth is *worse* on pure constant-commutativity ($x \cdot c = c \cdot x$), often by 5–20×

 and timing out on most 32-bit instances: its sign correction and pre-computed doublings add overhead the solver cannot simplify when both sides are computed with `bvmul`. Second, Booth excels on strength-reduction benchmarks where one side uses `bvmul` and the other is a hand-coded shift-add expression: the compressed representation matches the target, so the equivalence check becomes a few consistency constraints rather than a full circuit-to-circuit comparison. On `strength_chain_16` (three reductions at once) Booth is 24× faster than shift-add (125 KB DRAT proof vs. 3.2 MB). Booth’s benefit thus depends on whether the benchmark structure matches its decomposition—the optimal encoding depends on problem structure, not a single hierarchy (Table 5).

6.2. 4-Bit Block Multiplication

We divide both operands into 4-bit sections⁴ and compute $4 \times 4 \rightarrow 8$ -bit sub-products using shift-add (small, propagation-friendly), then accumulate block products via a Dadda tree. The 4-bit-boundary intermediate variables are eliminated efficiently by BVE: on `mul_ineq_12` block4 has the highest BVE elimination rate of any encoding (4,197 of 4,206 variables, 100% counting eliminated and fixed) and is the fastest encoding (13,617 vs. 15,662 conflicts for shift-add), suggesting the balanced tree aids BVE on problems mixing arithmetic and comparison.

6.3. No Single Encoding Dominates

The encoding comparison in this section and the cross-solver table in Section 7 together show that no multiplier encoding wins on every pattern. Table 5 summarises which encoding is fastest on each

⁴This block-decomposition encoding was proposed by one of the authors in unpublished prior work; it is implemented in CBMC’s `src/solvers/flattening/bv_utils.cpp` (block4 multiplier), which serves as the reproducibility reference.

problem class and, briefly, which structural feature drives the win:

Table 5

Best encoding per benchmark class.

Problem class	Best encoding	Key metric
Commutativity (2 identical circuits)	COMBA-CS	congruence closure
Constant multiplication: strength-reduction form	Booth	compressed PP matches target
Constant multiplication: pure commutativity	shift-add	Booth overhead dominates
Circuit equivalence (vs. shift-add)	shift-add	structural matching
Inequality constraints	block4	BVE elimination

Each row’s mechanism, explained above, is traceable to the DRAT analysis: COMBA-CS’s popcount variables align with the congruence pairs CADI_{CAL} finds between identical circuits (commutativity); Booth’s compressed grid matches a hand-coded shift-add target (strength reduction) but is dead weight when both sides use bvmul (pure constant commutativity, where shift-add’s smaller formula wins); block4’s 4-bit boundaries give BVE the most to eliminate (inequalities, 100% on mul_ineq_12); and when the target is itself a shift-add circuit, shift-add yields the tightest match.

Encoding selection must therefore respond to problem structure rather than a static ranking—the motivation for the adaptive heuristic (Section 5) and the cross-solver comparison (Section 7).

7. Evaluation

We ran all experiments on an Intel Xeon Platinum 8124M (3.0 GHz, 36 cores) with 68 GB RAM, Ubuntu 24.04, GCC 13.3.0 and solvers CADI_{CAL} 3.0.0, MINISAT 2.2.1, MERGESAT (docker-solvers branch, MiniSat-compatible API), and CRYPTOMINISAT 5.11.21, reporting the median of 3 runs with a 120 s timeout and a 6 GB memory limit and using ripple-carry adders unless otherwise noted. Two benchmark pools underlie the sections that follow: 36 custom QF_BV benchmarks (plus eight DSP kernels) for the core ablations (Sections 7.1–7.4), and a 66-benchmark stratified sample drawn from the SMT-COMP 2024 QF_BV track (Section 7.6). The 36-benchmark core suite spans algebraic identities (commutativity, overflow, strength reduction), industrial hashing and DSP kernels (MurmurHash3, Q15 DSP, keyed hash, division-by-constant), matrix/MAC products (trace, dot-product commutativity), division and floating-point (roundtrip, FP add/mul), and factoring.

7.1. Layer Ablation

To understand what the encoding contributes on top of CBMC’s word-level simplification, we ran a three-layer ablation on representative benchmarks, each configuration enabling one further layer: shift-add only → +simplify → +COMBA-CS (the algebraic pre-solver is disabled throughout, so this isolates the bit-blasting pipeline).

Table 6

Layer ablation, one representative per regime (CADI_{CAL}, seconds; T/O = 120 s; algebraic pre-solver disabled).

Benchmark	shift only	+simplify	+COMBA-CS
comm_16 (simplify-solved)	T/O	0.005	0.005
bf16_mul_comm_v2 (encoding-driven)	22.1	22.1	10.8
mul_ineq_12 (neither helps)	0.52	0.52	0.52

The ablation (Table 6) reveals three regimes: commutativity, associativity and distributivity are canonicalised by CBMC’s word-level simplifier to ≈5 ms, so the encoding has no measurable effect; bf16_mul_comm_v2 is beyond the word-level simplifier’s reach (floating-point semantics) and benefits from COMBA-CS for a 2× speedup; mul_ineq_12 and strength_chain_16 are insensitive to both layers on

this configuration, although we showed earlier that Booth dramatically improves `strength_chain_16` when selected explicitly. The encoding matters for the residual problems that word-level simplification cannot reduce.

7.2. Multiplier Encoding Comparison

Table 7

Four multiplier encodings across four solvers, ripple adder (seconds, T/O = 120 s). Best per row in bold. All rows re-measured on the pinned binary (cbmc @ 8a9cc557).

Solver	Benchmark	shift	Dadda	Std. Comba	COMBA-CS	Speedup
CADICAL	comm 9	2.19	0.59	0.34	0.16	14×
CADICAL	comm 11	60.8	3.38	1.74	0.95	64×
CADICAL	comm 13	T/O	T/O	8.21	3.21	∞
CADICAL	mat. trace	31.4	2.97	1.33	1.86	17×
MINISAT	comm 9	6.62	9.02	11.7	6.71	—
MINISAT	comm 11	T/O	T/O	T/O	T/O	—
MINISAT	mat. trace	T/O	60.4	5.43	17.1	∞
MERGESAT	comm 9	7.75	7.56	3.93	2.99	2.6×
MERGESAT	comm 11	T/O	23.2	31.1	13.9	∞
MERGESAT	comm 13	T/O	T/O	62.0	16.1	∞
MERGESAT	mat. trace	42.7	14.0	5.47	8.62	4.9×
CRYPTOMINISAT	comm 9	10.1	7.96	10.0	9.25	1.1×
CRYPTOMINISAT	comm 11	116	97.6	62.3	59.1	2.0×
CRYPTOMINISAT	mat. trace	T/O	32.8	28.6	80.8	∞

Table 7 reports wall-clock time (seconds, best per row in bold) for four multiplier encodings—shift-add, Dadda, standard Comba, and COMBA-CS—on each of four SAT solvers, so each row is a single-encoding sweep on a fixed solver and benchmark. The “Speedup” column shows COMBA-CS vs. shift-add (the previous default). A clear encoding hierarchy emerges for commutativity: shift-add < Dadda < standard Comba < COMBA-CS, consistent across all four solvers. For matrix trace, standard Comba is slightly faster than COMBA-CS on CADICAL (1.34 s vs. 1.93 s) because the carry-save second pass in COMBA-CS creates slightly different variable structure than standard Comba’s immediate carry propagation.

Context: algebraic pre-solver disabled. The numbers in Table 7 are measured with the algebraic pre-solver disabled, which is the relevant comparison for evaluating the encoding itself. When the pre-solver is enabled—the current CBMC default—all polynomial commutativity benchmarks (comm_9, comm_11, comm_13) are solved algebraically in ~5 ms regardless of encoding, masking the encoding’s effect. Table 7 therefore isolates the encoding’s contribution; in the deployed pipeline, encoding choices matter for benchmarks the pre-solver cannot handle: bitwise operations, inequalities, floating-point, and structural circuit mismatches.

7.3. Cross-Solver Analysis

The four solvers differ in their response to COMBA-CS for identifiable reasons: CADICAL benefits most (64×) because it combines the structural improvement with inprocessing BVE on COMBA-CS’s popcount variables and congruence closure on the gates between the two multiplier circuits; MERGESAT benefits from the structural improvement alone (2.6–∞×), as a MINISAT fork without inprocessing; MINISAT’s SatELite preprocessor simplifies the compact shift-add formula on small problems (comm BW=9: 6.62 vs. 6.71 s) but on larger formulas the structural benefit dominates (matrix trace: 5.43 s where shift-add times out); MINISAT times out on comm BW=11 for *all* encodings, exceeding its capacity regardless of encoding; CRYPTOMINISAT is consistently the slowest on multiplication. Its native XOR handling

provides no benefit (confirmed with and without XOR constraints), and although it does perform inprocessing variable elimination (via its `OccSimplifier`), it lacks the congruence-closure step that CADiCAL applies to the gates between two identical multiplier circuits—the mechanism the per-solver breakdown above identifies as decisive on commutativity-style benchmarks. The CaDiCaL-vs-others gap we observe on multiplication-heavy benchmarks is consistent with Dutertre’s broader empirical evaluation of sixteen SAT solvers on 15,447 bit-blasted QF_BV CNFs from SMT-LIB, which also found CADiCAL clearly superior [17]; our contribution is to identify BVE and congruence closure as the specific features responsible on multiplication. Importantly, the encoding benefit is not specific to CADiCAL: the COMBA-CS structural redesign helps even on a solver *without* inprocessing—MERGESAT, a MINISAT fork lacking BVE, still gains $2.6\text{--}\infty\times$ from COMBA-CS alone. CADiCAL’s BVE and congruence closure *amplify* this benefit (up to $64\times$) rather than create it, so the encoding finding transfers across solvers while the largest absolute speedups remain CADiCAL-specific.

Robustness. The comparison is robust to run-to-run noise (coefficient of variation $<1\%$ over five runs) and reproduces when solver effort is measured by DRAT proof size rather than wall-clock: COMBA-CS compresses commutativity proofs by up to $90\times$ (comm_11: 142 MB $\rightarrow$ 1.6 MB), Booth compresses strength-reduction proofs by $17\text{--}26\times$, block4 achieves the smallest proof on the commutativity benchmarks (comm_11: 1,124 KB), and sortnet produces the largest proofs on the inequality and strength-reduction benchmarks. Wallace and Dadda diverge by up to $10\times$ despite both being tree-based reductions. The full proof-size table across eight encodings (Table 8) shows that the per-row ranking reproduces the solver-time ranking, so the timing differences are not an artefact of CDCL heuristics; every claim in the paper could be restated in terms of proof sizes with the same conclusions.

Table 8

DRAT proof sizes in KB across encodings (algebraic layers disabled, CADiCAL, 300 s timeout). Best (smallest) per row in bold.

Benchmark	shift-add	Dadda	Wallace	Comba	COMBA-CS	Booth	block4	sortnet
comm_11	145,073	8,475	88,305	5,799	1,617	93,252	1,124	22,651
comm_13	T/O	38,226	T/O	13,169	3,183	T/O	2,586	74,123
mul_ineq_12	1,406	1,544	1,590	1,727	1,406	3,500	1,814	7,856
overflow_detect_16	1,462	779	1,017	3,860	1,462	524	1,209	287
strength_16_31	275	234	185	401	224	16	129	790
strength_chain_16	3,275	4,651	2,487	3,438	2,826	125	1,846	7,455

7.4. Scaling

The BW=16 separation in Figure 2 is telling: COMBA-CS solves in 5.4 s, block4 in 10.8 s, standard Comba in 33.7 s, and everything else times out at 120 s. This $2\text{--}5\times$ gap between the top-three encodings remains stable up to BW=16 (beyond which all bit-blasting encodings eventually time out in this range of wall-clock). The ordering $\text{shift-add} < \text{Dadda} < \text{Comba} < \text{COMBA-CS}$ predicted by carry-propagation depth (Section 3) holds at every bit-width tested.

7.5. Industrial Benchmarks

COMBA-CS gives large speedups on benchmarks with multiple symbolic $\times$ symbolic multiplications (matrix trace, MAC; Table 9). The keyed-hash regression (1.3 s $\rightarrow$ 6.3 s) is constant multiplication routed through dadda-cs instead of shift-add by the adaptive fallback, but stays fast in absolute terms.

More broadly, UNSAT problems show consistent encoding sensitivity (up to $64\times$) with stable ranking across solvers. SAT problems (factoring) show encoding sensitivity at large bit-widths but with unpredictable ranking—different encodings make different solutions easy to find.

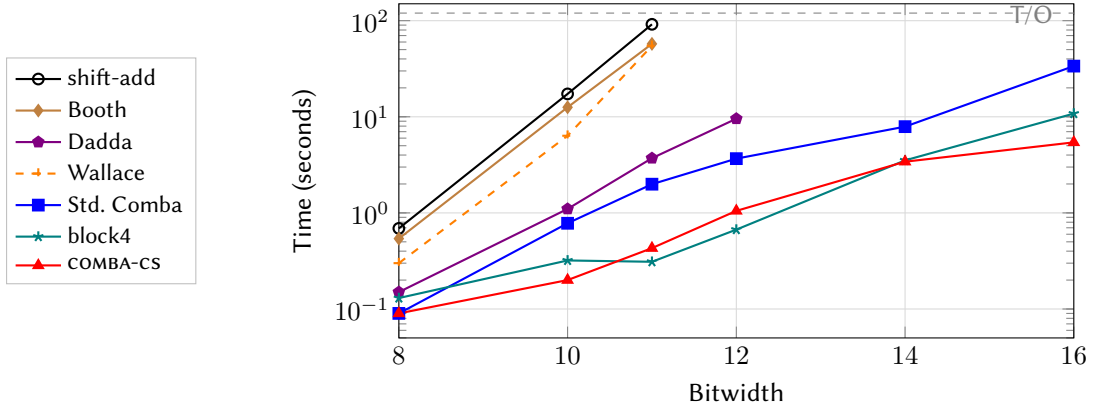


Figure 2: Commutativity scaling: seven encodings across BW=8–16 on CADiCAL (CBMC’s `smt2_solver`, algebraic layers disabled). `shift-add`, `Booth`, and `Wallace` time out at $BW \geq 12$; `Dadda` at $BW \geq 14$; only standard `Comba`, `block4`, and `COMBA-CS` solve $BW=16$ within 120 s. `COMBA-CS`’s advantage grows with bit-width.

Table 9

Industrial benchmarks: `shift-add` vs. `COMBA-CS`, ripple adder (CADiCAL).

Benchmark	shift-add	COMBA-CS	Speedup
MurmurHash3 determinism	9.9s	7.1s	1.4×
$\text{tr}(AB) = \text{tr}(BA)$	31.3s	1.9s	17×
MAC dot product comm	14.7s	4.0s	3.7×
keyed hash determinism	1.3s	6.3s	0.2×
div-by-const optimisation	9.0s	9.0s	1.0×

7.6. Community Benchmarks (Custom Suite and SMT-COMP)

On a 44-benchmark custom QF_BV suite run through CBMC’s `smt2_solver` (which lacks word-level simplification and the algebraic pre-solver, isolating the encoding), the adaptive `COMBA-CS` heuristic (Section 5) introduces no regression except one inherent circuit-equivalence case (`hw_equiv_12`, $0.15\times$: a `bvmul` compared against a hand-built `shift-add` circuit, which no encoding heuristic can match without knowing the target’s structure). To further test external validity we took a stratified sample of 66 multiplication-containing benchmarks from the SMT-COMP 2024 QF_BV track, drawn across 22 submitter categories.⁵ We ran each benchmark with CADiCAL and a 60 s timeout in two configurations that differ only in the multiplier encoding: *shift-add-nosimp* (`shift-add` encoding, simplification disabled), the raw bit-blasting baseline; and *combacs-nosimp* (`COMBA-CS` encoding, simplification disabled), which isolates the encoding’s contribution on top of the same baseline. Both use CBMC’s `smt2_solver`, so the only difference between them is the multiplier encoding. (We restrict this experiment to pure bit-blasting; the word-level algebraic pre-solver is orthogonal to the encoding question studied here and is outside its scope.)

Table 10

SMT-COMP QF_BV sample (66 benchmarks, CADiCAL, 60 s T/O).

Configuration	Solved	PAR-2 (s/benchmark)
<code>shift-add-nosimp (smt2_solver)</code>	34/66	62.5
<code>COMBA-CS-nosimp (smt2_solver)</code>	34/66	62.6

The encoding-isolation comparison is decisive (Table 10): on this broad community sample `COMBA-CS` and `shift-add` solve the same 34 benchmarks with near-identical PAR-2 (62.5 vs. 62.6 s), and on the 34

⁵Sampling procedure: all benchmarks containing `bvmul` with file size < 100 KB were enumerated per submitter category; we took 5 benchmarks from each category (fewer if the category had fewer than 5), yielding 66 benchmarks.

both-solved the per-benchmark ratio is between $0.85\times$ and $1.33\times$.

The headline takeaway is that the COMBA-CS encoding is *nearly neutral on a broad QF_BV community sample*: it is not a silver bullet, nor is it a regression. The per-benchmark speedups we report elsewhere (COMBA-CS up to $64\times$ on commutativity, Booth up to $32\times$ on strength reduction, block4 up to $129\times$ smaller proofs on comm_11) are pattern-specific: they manifest when the benchmark matches the encoding’s target. On general QF_BV benchmarks most multiplications are small ($BW \leq 32$) and the bottleneck is rarely the multiplier. Our improvements are most visible on the bv-term-small-rw family, brummayerbiere3, and calypto; they are invisible on ecrw (512-bit rewrites) and brummayerbiere4 (unconstrained), which are uniformly hard for both configurations.

Threats to validity. Bit-blasting scales poorly with bit-width (exponential in the worst case) but is relatively insensitive to the number of variables; our benchmarks exercise bit-widths 8–32, so results may differ at 64+ bits. Some QF_BV benchmarks are circuit-equivalence problems whose difficulty is highly encoding-sensitive—a new encoding that improves general QF_BV performance may still fail on these specific instances, making aggregate QF_BV evaluation potentially misleading for encoding research. Finally, the 66-benchmark SMT-COMP sample is small relative to the full QF_BV suite ($\sim 16,000$ benchmarks containing bvmul); stratifying by submitter category controls for benchmark-family bias but not for difficulty distribution within a category, and a full-suite evaluation is future work.

8. Negative Results

A significant portion of this investigation produced negative results. We report them to save future effort and because the failure modes illuminate SAT solver behaviour. A natural first question is whether asymptotically superior algorithms—Karatsuba [18] ($O(n^{1.585})$), Toom-Cook [19] ($O(n^{1.465})$), or Schönhage-Strassen [20] ($O(n \log n \log \log n)$)—produce better SAT encodings. CBMC includes implementations of all three. They are designed for *computational* efficiency (fewer arithmetic operations), not *verification* efficiency (fewer SAT conflicts). Karatsuba’s combination step requires additions with carry chains—exactly the structure Section 3 identifies as the hardness source.

Table 11
Subquadratic encodings vs. std. Comba (CADICAL, seconds).

Benchmark	Std. Comba	Karatsuba	Toom-Cook	Schönhage
comm BW=9	0.34	3.01	2.69	89.8
comm BW=13	8.14	9.00	22.4	T/O
prime BW=28	3.42	118	T/O	T/O
aws_mul	1.54	3.84	T/O	T/O

Karatsuba (Table 11) is competitive on commutativity at BW=13 but $35\times$ slower on primality (relative to standard Comba). Schönhage-Strassen produces 6,647 variables for BW=7 (vs. 317 for Comba) and times out everywhere beyond comm BW=9—designed for thousands of bits, it is far too expensive at practical bit-widths. On the re-measured pinned binary standard Comba is fastest on all four problems, including aws_mul (1.5 s)—its constant-multiplication handling has improved—so the subquadratic algorithms are now uniformly slower than Comba.

8.1. Supplying the Polynomial Proof

Section 3 establishes that commutativity has a polynomial-size refutation CDCL does not find. We tested whether *supplying* that guidance closes the gap, on the array-multiplier commutativity CNF (CADICAL, at $n=9$ where the bare instance needs 822k conflicts; Table 12). Asserting the $2N$ output bit-equalities trivialises the instance, but the proof bypasses the multiplier entirely—it is unchanged when the two products are replaced by free variables—so it measures nothing about proof search.

Guidance that keeps the circuit in the proof does not help: the n^2 true partial-product correspondences do not reduce the conflict count (972,767 vs. 822,218 bare), and the restricted Beame–Liew strip—a few hundred variables, on which a polynomial proof provably exists and which our generator builds and `drat-trim` validates—still costs $\sim 2.7 \times 10^5$ conflicts. The heuristic route fares no better: renumbering into strip order and forcing the phase yields at most a $1.4\times$ speedup and stays exponential (6×10^5 conflicts at $n=9$, rising to 2.5×10^6 at $n=10$)—consistent with Vinyals’ result [21] that relevance-based decision heuristics cannot polynomially simulate resolution. The gap is thus robust to both halves of Beame and Liew’s open problem [1]; closing it appears to require either a stronger proof system at the solver level [22, 23] or, as we pursue, a change to the encoding structure.

Table 12

Guidance injection on array-multiplier commutativity ($n=9$, CADICAL; bare = 822,218 conflicts). Only the multiplier-bypassing assertion trivialises the instance.

Guidance supplied	Conflicts	Circuit in proof?
None (bare)	822,218	—
Output bit-equalities	0	no (bypass)
Partial-product corresp. (n^2)	972,767	yes
Restricted BL strip (column 9)	272,390	yes
Strip-order variables + phase	606,940	yes

8.2. Approaches That Hurt Performance

Several approaches that add structure to the encoding *reintroduce* carry chains and thereby hurt performance. Radix-4/8 multipliers are $2\text{--}15\times$ slower: pre-computing $3x$, $5x$, $7x$ adds carry chains that dominate at practical bit-widths. Full-adder tree popcount is $3\times$ slower than pop0 because it creates longer carry chains ($\log_2 n$ vs. $2\text{--}4$ bits). Non-propagation-complete carry merge—encoding carry with 4 clauses instead of 6—causes a $25\times$ regression [8]. Caching AND gates between multiplications is 89% slower because shared partial-product variables prevent independent BVE.

Sorting networks. We compute n^2 partial product bits, sort each column with a bitwise sorting network (compare-and-swap = $(x \vee y, x \wedge y)$), and add auxiliary constraints enforcing the sorted property ($out[i] \Rightarrow out[i-1]$). Despite achieving the highest fixed-variable count (2,964 vs. 47 for shift-add on `strength_chain_16`), the encoding produces 13,283 variables and 40,366 clauses ($19\times$ and $14\times$ more than shift-add), and its 25.8M propagations overwhelm the structural benefit—auxiliary constraints that improve propagation locally can be net-negative when they dramatically increase formula size.

Greedy Addition and BVA. We tested Jia et al.’s Greedy Addition [24] (Huffman-like partial-product ordering) across the four solvers: CADICAL benefits $1.4\text{--}2.6\times$ because its inprocessing BVE works better with balanced trees, but MINISAT and MERGESAT regress up to $1.8\times$ because their preprocessing BVE prefers long sequential carry chains. Fazekas et al.’s Bounded Variable Addition [25], when enabled in CADICAL, times out on every multiplication benchmark in our suite.

8.3. Approaches With Limited or Chaotic Benefit

XOR-based techniques consistently disappoint. Offline Gaussian elimination, online incremental GE, and CRYPTOMINISAT’s built-in XOR handling gave at best $1.4\times$ on one benchmark with neutral-to-negative results elsewhere; CBMC’s XOR constraints are 3-variable gates, too short for GE to combine usefully. Solver tuning and abstraction refinement fail because the problem is structural, not parametric: 512 CADICAL option combinations ($16 \text{ options} \times 4 \text{ encodings} \times 8 \text{ benchmarks}$) changed no result by more than $\pm 3\%$. CBMC’s CEGAR loop (`--refine-arithmetic`) adds ≥ 2 solver calls of overhead on UNSAT benchmarks (commutativity BW=11: 4.0 s vs. 0.9 s direct COMBA-CS). All five parallel prefix

adders (Brent-Kung [BK], Kogge-Stone, Sklansky, Han-Carlson, Ladner-Fischer) produce identical learned-clause quality (95% glue-1 [26]); BK wins solely by having the fewest variables.

8.4. BVE Interaction

Section 5 noted that inprocessing BVE is counterproductive for COMBA-CS at $BW \geq 11$; disabling it ($\text{elim}=0$) gives a further $1.1\text{--}2.3\times$ speedup, as BVE’s resolvents on the popcount variables are harder than the originals. Tracing clause creation and deletion in CADICAL shows why: the latest-learned clauses encode carry relationships between adjacent equality-check bits. For commutativity ($c = a \times b$, $d = b \times a$, assert $c = d$) the check builds XNOR gates $eq[i] = c[i] \Leftrightarrow d[i]$, and the late clauses take the form $(eq[i] \vee eq[i+1])$ (“if bit i differs, bit $i+1$ must agree”). Pre-providing these as encoding clauses gives an 18–52% speedup at $BW=10\text{--}13$ but a 59–94% *regression* at $BW=14\text{--}16$ —an instance of CDCL’s sensitivity to clause perturbation [27]: the hints change BVE’s elimination order, whose residual difficulty is non-monotonic. We enable them in CBMC only for 10–13-bit equality checks.

9. Related Work

Algebraic verification (brief). Early BDD-based multiplier verification [28] succeeded up to 16 bits; Bryant [7] showed the input-variable BDD is exponential, whereas Kumar et al. [29] verify multipliers to 1024 bits at $O(n^3)$ BDD size using partial-product variables—so, as with our SAT encodings, the blow-up is representation-dependent, not inherent. Gate-level algebraic verification via Gröbner bases or function extraction [30, 31, 32, 33] works over $\mathbb{Q}$, and Song et al. [34] use strong Gröbner bases over $\mathbb{Z}_{2^d}$ for the bit-vector equational theory. These techniques scale with bit-width but are limited to universal polynomial fragments, complementing bit-blasting; the Gröbner system can be exponentially stronger than resolution [35].

Encoding and SAT techniques. Dadda [11] and Wallace [12] proposed tree-based partial product reduction. Comba [13] proposed column-wise reduction. COMBA-CS extends Comba with carry-save separation, optimised for SAT solving rather than hardware implementation. Howe et al. [36] give compact bit-blasting encodings for univariate polynomial evaluation by fixing input bits one at a time; extended to multivariate products ($a \times b$) this yields structure close to the standard partial-product grid. Eén and Biere [6] introduced SatELite preprocessing with BVE. Biere et al. [10] extended this to inprocessing in CADICAL. We find that inprocessing BVE can be counterproductive for specific encoding structures (Section 8.4). Brain et al. [8] showed that PC encodings compose for adders but not multipliers. Brain [9] conjectured that a PC multiplier is exponential and explored Toom-Cook interpolation as an alternative. Jia et al. [24] improve bit-blasting for nonlinear integer arithmetic (QF_NIA) with Greedy Addition (a Huffman-like ordering that minimises auxiliary variables) and Multiplication Adaptation (starting with narrow bit-widths); we evaluate Greedy Addition across four solvers in Section 8. Fazekas et al. [25] extend incremental inprocessing to support Bounded Variable Addition (BVA), the dual of BVE. Encoding techniques for SAT have been studied extensively [37, 38].

Resolution upper bounds for ring identities. Beame and Liew [15, 1] refute an earlier conjecture that multiplier commutativity requires exponential-size resolution proofs: they give polynomial-size regular resolution proofs for any degree-2 ring identity (commutativity, distributivity) on array, diagonal, and Booth multipliers, and quasi-polynomial-size ($n^{O(\log n)}$) proofs for Wallace tree multipliers. Their construction splits the multiplier formula into narrow “critical strips” and uses a branching program (read-once BDD) that scans each strip in parallel; they note that current CDCL solvers do not find these short proofs and call it an open problem to add guiding information that does.

Escaping resolution hardness in CDCL. Whether a proof search can *find* the short proofs that exist is the question of *automatizability* in proof complexity; for CDCL it is constrained by the decision

heuristic. Vinyals [21] shows that the relevance-based heuristics used in practice (VSIDS, VMTF, CHB, LRB) cannot polynomially simulate resolution—there are formulas with polynomial-size resolution refutations that these heuristics require exponential time to solve. This is the theoretical counterpart of our empirical observation (Section 8.1) that reordering the solver’s decisions does not let it find the known short proof. Three complementary lines of work address the gap between existence of short proofs and CDCL’s ability to find them. (i) Encoding-level guidance—our approach—reshapes the initial formula so CDCL’s standard heuristics reach short proofs. (ii) Solver-level generalisation to stronger proof systems: Danner’s dissertation [22] develops CDXCL, a CDCL generalisation to XOR-OR-AND Normal Form (XNF) clauses that operates in the $\text{Res}(\oplus)$ proof system—equivalent to Itsykson and Sokolov’s SRES—and that provably escapes Resolution lower bounds on Tseitin formulas and related XOR-rich instances. Independently, Beame and Sun [23] develop $\text{CDCL}(\oplus)$, which both produces and polynomially simulates $\text{Res}(\oplus)$ proofs (given nondeterministic decisions and restarts), mirroring the classical CDCL/Resolution correspondence; their solver Xorcle outperforms Kissat and CRYPTO_{MINISAT} on XNF benchmarks. That two independent groups reach the same proof-system generalisation underlines it as the natural solver-level route around resolution hardness. (iii) Inprocessing-level reshaping: Pollitt et al. [39] revive Extended Resolution inside modern CDCL through CaDiCaL-ER (definition introduction during conflict analysis) and CaDiCaL-FX (bounded variable addition extended to XOR and ITE gate detection on learned clauses). They further give polynomial-size ER proofs for Tseitin formulas via XOR gate factorisation followed by bounded variable elimination. Our encoding-selection work is orthogonal to both: we shape the CNF that is fed to CDCL, whereas CDXCL changes the proof system and CaDiCaL-FX reshapes clauses during search.

Word-level solvers. Niemetz et al. [40] implement word-level normalisation in Bitwuzla that avoids bit-blasting for algebraic identities; cvc5 uses a similar rewriting infrastructure. These approaches are complementary to our encoding improvements: when word-level rewriting succeeds, bit-blasting is not needed; when it does not (bitwise operations, non-polynomial constraints), the encoding matters. Chukharev et al. [41] complement both by showing that circuit-structure-aware SAT partitioning solves multiplier-equivalence instances intractable for standard SAT and for algebraic methods.

10. Conclusion and Future Work

We presented a proof-guided design loop—analyse DRAT proofs for structural bottlenecks, then redesign the encoding to remove them—demonstrated in depth on COMBA-CS (up to $64\times$ on commutativity) and applied retrospectively to Booth radix-4 (up to $32\times$ on strength-reduction), 4-bit blocks, and sorting networks. A controlled experiment varying only the accumulator operator establishes carry propagation as the dominant driver of CDCL search difficulty: structural variations change running time by less than $5\times$ while carry presence changes it by orders of magnitude, localising the difficulty in the gap between Beame–Liew’s short proofs and CDCL’s ability to find them. No single encoding dominates: commutativity benefits from congruence closure (COMBA-CS), constant multiplication from fewer partial products (Booth), and circuit equivalence from structural matching (shift-add).

Open directions include automating the proof-guided loop, applying the DRAT-bottleneck redesign to other multiplier architectures (the Booth, 4-bit-block, and sorting-network encodings here were taken as given, not proof-guided-redesigned), and reconciling our encoding-level guidance with recent solver-level [22, 23] and inprocessing-level [39] work on finding the short proofs guaranteed by [1].

Declaration on Generative AI. During preparation, the authors used Kiro (an AI coding assistant by Amazon) to assist with implementation, experimentation, and drafting. All results were verified by the authors, and the final paper was reviewed and edited by the authors.

References

- [1] P. Beame, V. Liew, Toward verifying nonlinear integer arithmetic, *Journal of the ACM* 66 (2019) 22:1–22:30. doi:10.1145/3319396.
- [2] E. Clarke, D. Kroening, F. Lerda, A tool for checking ANSI-C programs, in: *TACAS*, 2004, pp. 168–176. doi:10.1007/978-3-540-24730-2_15.
- [3] D. Kroening, O. Strichman, *Decision Procedures: An Algorithmic Point of View*, Texts in Theoretical Computer Science. An EATCS Series, second ed., Springer, 2016. doi:10.1007/978-3-662-50497-0.
- [4] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, second ed., IOS Press, 2021, pp. 1267–1329. doi:10.3233/FAIA201017.
- [5] M. Järvisalo, M. J. H. Heule, A. Biere, Inprocessing rules, in: *IJCAR*, 2012, pp. 355–370. doi:10.1007/978-3-642-31365-3_28.
- [6] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: *SAT*, 2005, pp. 61–75. doi:10.1007/11499107_5.
- [7] R. E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, *IEEE Trans. Computers* 40 (1991) 205–213. doi:10.1109/12.73590.
- [8] M. Brain, L. Hadarean, D. Kroening, R. Martins, Automatic generation of propagation complete SAT encodings, in: *VMCAI*, 2016, pp. 536–556. doi:10.1007/978-3-662-49122-5_26.
- [9] M. Brain, Further steps down the wrong path: Improving the bit-blasting of multiplication, in: *SMT Workshop*, 2021, pp. 23–31. URL: <https://ceur-ws.org/Vol-2908/>.
- [10] A. Biere, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: *Proc. SAT Competition 2020*, 2020, pp. 51–53.
- [11] L. Dadda, Some schemes for parallel multipliers, *Alta Frequenza* 34 (1965) 349–356.
- [12] C. S. Wallace, A suggestion for a fast multiplier, *IEEE Trans. Electronic Computers* EC-13 (1964) 14–17. doi:10.1109/PGEC.1964.263830.
- [13] P. G. Comba, Exponentiation cryptosystems on the IBM PC, *IBM Systems Journal* 29 (1990) 526–538. doi:10.1147/sj.294.0526.
- [14] H. S. Warren, *Hacker’s Delight*, 2nd ed., Addison-Wesley, 2012.
- [15] P. Beame, V. Liew, Towards verifying nonlinear integer arithmetic, in: R. Majumdar, V. Kunčák (Eds.), *Computer Aided Verification (CAV)*, volume 10427 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 238–258. doi:10.1007/978-3-319-63390-9_13.
- [16] A. D. Booth, A signed binary multiplication technique, *The Quarterly Journal of Mechanics and Applied Mathematics* 4 (1951) 236–240. doi:10.1093/qjmam/4.2.236.
- [17] B. Dutertre, An empirical evaluation of SAT solvers on bit-vector problems, in: *11th International Workshop on Pragmatics of SAT (PoS)*, 2020.
- [18] A. Karatsuba, Y. Ofman, Multiplication of multidigit numbers on automata, *Soviet Physics Doklady* 7 (1962) 595–596.
- [19] A. L. Toom, The complexity of a scheme of functional elements realizing the multiplication of integers, *Soviet Mathematics Doklady* 3 (1963) 714–716.
- [20] A. Schönhage, V. Strassen, Schnelle Multiplikation großer Zahlen, *Computing* 7 (1971) 281–292. doi:10.1007/BF02242355.
- [21] M. Vinyals, Hard examples for common variable decision heuristics, in: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 1652–1659.
- [22] J. Danner, SAT Solving Using XOR-OR-AND Normal Forms and Cryptographic Fault Attacks, Ph.D. thesis, Universität Passau, Faculty of Computer Science and Mathematics, 2025. Rigorousum: 21 October 2025; reviewers: Prof. Martin Kreuzer (Passau), Prof. Armin Biere (Freiburg).
- [23] P. Beame, G. Sun, Extending CDCL to disjunctions of parity equations, *LIPIcs* 42 (2026). URL: <https://arxiv.org/abs/2605.15002>. arXiv:2605.15002.
- [24] F. Jia, R. Han, P. Huang, M. Liu, F. Ma, J. Zhang, Improving bit-blasting for nonlinear integer

- constraints, in: ISSTA, 2023, pp. 1230–1241. doi:10.1145/3597926.3598034.
- [25] K. Fazekas, F. Pollitt, M. Fleury, A. Biere, Incremental inprocessing rules beyond resolution, in: *Pragmatics of SAT*, 2025.
 - [26] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: *IJCAI*, 2009, pp. 399–404.
 - [27] J. Elffers, J. Nordström, Trade-offs between time and memory in a tighter model of CDCL SAT solvers, in: *SAT*, 2018, pp. 160–176. doi:10.1007/978-3-319-94144-8_10.
 - [28] J. R. Burch, Using BDDs to verify multipliers, in: *28th ACM/IEEE Design Automation Conference (DAC)*, 1991, pp. 408–412. doi:10.1145/127601.127702.
 - [29] J. Kumar, Y. Miyasaka, A. Srivastava, M. Fujita, Formal verification of integer multiplier circuits using binary decision diagrams, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* 42 (2023) 1365–1378. doi:10.1109/TCAD.2022.3198899.
 - [30] A. Biere, M. Kauers, D. Ritirc, Challenges in verifying arithmetic circuits using computer algebra, in: *SYNASC*, 2017, pp. 1–8. doi:10.1109/SYNASC.2017.00009.
 - [31] D. Kaufmann, Formal Verification of Multiplier Circuits using Computer Algebra, Ph.D. thesis, Johannes Kepler University Linz, 2020.
 - [32] D. Kaufmann, A. Biere, AMulet 2.0 for verifying multiplier circuits, in: *TACAS*, 2021, pp. 357–364. doi:10.1007/978-3-030-72013-1_19.
 - [33] C. Yu, W. Brown, D. Liu, A. Rossi, M. Ciesielski, Formal verification of arithmetic circuits by function extraction, *IEEE Trans. CAD* 35 (2016) 2131–2142. doi:10.1109/TCAD.2016.2547898.
 - [34] J. Song, H. Fu, C. Zhang, Equational bit-vector solving via strong Gröbner bases, *CoRR* abs/2402.16314 (2024). URL: <https://arxiv.org/abs/2402.16314>.
 - [35] M. Clegg, J. Edmonds, R. Impagliazzo, Using the Groebner basis algorithm to find proofs of unsatisfiability, in: *STOC*, 1996, pp. 174–183. doi:10.1145/237814.237860.
 - [36] J. M. Howe, M. Brain, A. Gàmez-Montolio, Evaluating binary polynomials using subpolynomials, in: *Proceedings of the 23rd International Workshop on Satisfiability Modulo Theories (SMT)*, volume 4008 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 3–15.
 - [37] G. S. Tseitin, On the complexity of derivation in propositional calculus, in: *Automation of Reasoning*, 1983, pp. 466–483. doi:10.1007/978-3-642-81955-1_28.
 - [38] D. A. Plaisted, S. Greenbaum, A structure-preserving clause form translation, *J. Symbolic Computation* 2 (1986) 293–304. doi:10.1016/S0747-7171(86)80028-1.
 - [39] F. Pollitt, Z. Battleman, M. Fleury, Y. Vizel, M. J. H. Heule, A. Biere, R. E. Bryant, Factoring learned clauses, in: *Proc. SAT*, 2026. To appear.
 - [40] A. Niemetz, M. Preiner, Bitwuzla, in: *CAV*, 2023, pp. 3–17. doi:10.1007/978-3-031-37703-7_1.
 - [41] K. Chukharev, I. Gribanova, D. Ivanov, S. Kochemazov, V. Kondratiev, A. Semenov, Effective partitioning method with predictable hardness for CircuitSAT, *IEEE Access* 13 (2025). doi:10.1109/ACCESS.2025.3527819.