Verifying Properties of Bit-vector Multiplication Using Cutting Planes Reasoning

Vincent Liew[1], Paul Beame[2], Jo Devriendt[3], Jan Elffers[4], Jakob Nordström[5]
[1,2]*Allen School of Computer Science & Engineering*
University of Washington
Seattle, WA, USA
[1]vliew@cs.washington.edu, [2]beame@cs.washington.edu
[3,4]*Department of Computer Science*
Lund University & University of Copenhagen
Lund, Sweden and Copenhagen, Denmark
[3]jo.devriendt@cs.lth.se, [4]jan.elffers@cs.lth.se
[5]*Department of Computer Science*
University of Copenhagen & Lund University
Copenhagen, Denmark and Lund, Sweden
jn@di.ku.dk

***Abstract*—Systems mixing Boolean logic and arithmetic have been a long-standing challenge for verification tools such as SAT-based bit-vector solvers. Though SAT solvers can be highly efficient for Boolean reasoning, they scale poorly once multiplication is involved. Algebraic methods using Gröbner basis reduction have recently been used to efficiently verify multiplier circuits in isolation, but generally do not perform well on problems involving bit-level reasoning.**

We propose that pseudo-Boolean solvers equipped with cutting planes reasoning have the potential to combine the complementary strengths of the existing SAT and algebraic approaches while avoiding their weaknesses.

Theoretically, we show that there are optimal-length cutting planes proofs for a large class of bit-level properties of some well known multiplier circuits. This scaling is significantly better than the smallest proofs known for SAT and, in some instances, for algebraic methods. We also show that cutting planes reasoning can extract bit-level consequences of word-level equations in exponentially fewer steps than methods based on Gröbner bases.

Experimentally, we demonstrate that pseudo-Boolean solvers can verify the word-level equivalence of adder-based multiplier architectures, as well as commutativity of bit-vector multiplication, in times comparable to the best algebraic methods. We then go further than previous approaches and also verify these properties at the bit-level. Finally, we find examples of simple nonlinear bit-vector inequalities that are intractable for current bit-vector and SAT solvers but easy for pseudo-Boolean solvers.

***Index Terms*—Multiplier circuits, bit-vector arithmetic, verification, pseudo-Boolean solving, cutting planes, SAT solving, Gröbner bases**

I. Introduction

While there has been great progress in verification tools since the 1980s, current methods still cannot efficiently deal with problems that combine multiplication and Boolean operations. These problems are encapsulated in the theory of *bit-vector arithmetic*, which supports both common bit-level operations like shifting and word-level arithmetic operations like addition and multiplication of bit-vectors. Thus, bit-vector formulas can express the behavior of a program or arithmetic circuit in a natural, yet bit-precise, manner.

Though deciding bit-vector formulas is NEXPTIME-complete in general [31], current bit-vector solvers are fairly efficient on many problems arising in practice ([12], [18], [22], [27], [38], [41], [47]). However, for instances that involve multiplication these solvers must often rely on the *bit-blasting* approach [32], which determines the satisfiability of a bit-vector formula by converting it into an equisatisfiable CNF formula to be fed into a conflict-driven clause learning (CDCL) SAT solver ([4], [39], [42]).

While CDCL SAT solvers effectively handle bit-level operations, they tend to perform poorly when multiplication is involved, with running times scaling exponentially in the bit-width on such problems ([8], [14], [30]). CDCL solvers are based on *resolution* ([11], [46]), in the sense that a *resolution proof* can be extracted from the execution trace for an unsatisfiable formula [5]. Thus, weaknesses of this proof system impose hard limits on solver performance. Resolution is very poor at tasks like counting [24] and mod-2 reasoning [52], and though degree-2 multiplier identities were recently shown to have polynomial-size proofs [7], these proofs are quite large. To unlock the ability to solve even more complicated formulas that mix bit-level reasoning with multiplication, we need to fundamentally improve the back-end reasoning.

Two natural approaches for strengthening resolution-based reasoning are embodied by the proof systems *polynomial calculus* [16], which reasons with polynomials instead of clauses, and *cutting planes* [17], which operates on 0-1 linear inequalities. Both of these proof systems can efficiently simulate resolution, and can be exponentially stronger.

Computer algebra has recently emerged as a powerful tool for verifying isolated gate-level multiplier circuits ([10], [15], [36], [37], [44], [45], [50], [51], [54]). A major advantage of

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_27

Gröbner basis methods, which perform algebraic reasoning that is captured by the polynomial calculus proof system, is that they operate with polynomials instead of disjunctive clauses. This makes it possible to encode the *correctness* of a multiplier with input bit-vectors $\mathbf{x}, \mathbf{y}$ and output bit-vector $(\mathbf{xy})$ through the word-level *specification equation*:

$$\Big(\sum_{i=0}^{n-1} 2^i x_i\Big)\Big(\sum_{i=0}^{n-1} 2^i y_i\Big) - \Big(\sum_{i=0}^{2n-1} 2^i (xy)_i\Big) = 0.$$

Unfortunately, for the non-algebraic parts of circuits, Gröbner basis methods are typically orders of magnitude slower than SAT solvers and scale poorly on general reasoning. We provide an explanation for this by showing, drawing on [25], that Gröbner basis methods require an exponential number of steps to derive bit-level consequences of word-level properties. Hence, these methods are unlikely to supplant the role of SAT solvers for bit-vector arithmetic.

We propose instead that *conflict-driven pseudo-Boolean solvers* [13] that take advantage of the *cutting planes* method for 0-1 linear inequalities [17] have the potential to achieve the "best of both worlds", combining the strengths of Gröbner basis methods for polynomials with the efficiency of CDCL SAT solvers for Boolean reasoning. Cutting planes reasoning can easily express word-level properties and does not suffer the same obstacles as polynomial calculus, since only a linear number of steps are needed to derive *all* of the individual bit-equalities from a word-level equality.

An essential aspect of this approach in improving on SAT-based methods is that one can express the correctness of 1-bit adders, basic building blocks of arithmetic circuits, directly via pairs of inequalities, instead of using sets of clauses, and one can similarly directly express word-level properties of circuit outputs. Together, these yield a higher-level fully precise form of "bit-blasting".

The main theoretical contribution of this paper is the construction of optimal, $O(n^2)$-length cutting planes proofs for a large class of n-bit ring identities, including commutativity and distributivity. We emphasize that these identities can be proven not only at the word level, but also for individual bits.

While $O\big(n^2\big)$-length polynomial calculus proofs are known for some of these properties at the word level [29], this algebraic method cannot efficiently extract the bit-equalities. As a consequence, for example, the best known polynomial calculus proof for the bit-level property "the middle bit of xy equals the middle bit of yx" is still the $O\big(n^5 \log n\big)$-length resolution proof given by [7], which is much larger than our $O\big(n^2\big)$-length cutting planes proof.

These ring identities appeared previously as testbed instances representing the gap between word-level and bit-level methods of reasoning. For example, it was observed in 2016 that proving the commutativity of a multiplier circuit is already intractable for SAT solving at 16 bits [9]. While bit-vector solvers try to overcome this shortcoming of SAT by implementing word-level preprocessing and inprocessing, the verification of larger systems containing multiplication and bit-logic (that appear for instance, in cryptography) remains a key weakness. The ability to verify these ring identities at the bit level, rather than through preprocessing, is a good test for the potential of any method for verifying these more complex systems.

Experimentally, we are able to use pseudo-Boolean solvers to verify the word-level equivalence of several different multiplier circuits of up to 256 bits in similar times to those of the best algebraic methods. We find that these solvers can be particularly efficient at extracting all of the bit-level equalities from a word-level equality, which neither CDCL solvers nor Gröbner basis reduction can do efficiently.

We also show that pseudo-Boolean solvers can be used to efficiently verify a number of bit-vector inequalities combining multiplication with bit-wise operations. In contrast, these inequalities are much harder or intractable for the top bit-vector solvers Boolector ([12], [43]), Z3 [18], Yices2 [19] and CVC4 [2]. Our examples demonstrate some of the potential of pseudo-Boolean solvers for reasoning with nonlinear, bit-precise systems that are out of reach of current methods. These bit-vector inequalities are inspired by the combinations of arithmetic and bit-wise operations that naturally arise in embedded systems or high-performance computation, where "bit hacks" can be used to implement methods such as absolute value or "reverse the bits in a byte" (see [1] and [26]) and more complicated mixtures of arithmetic and bit-wise operations are used in cryptographic and hashing computations.

II. Notation and preliminaries

We write the i-th entry of a bit-vector $\mathbf{x}$ as a Boolean variable x_i. We typically refer to circuits by the output bit-vectors that they produce — for example we use $\mathbf{C}$ to refer to both a circuit and its output bit-vector, depending on the context. Often we write this output bit-vector in terms of the inputs, so that a multiplier circuit denoted by $\mathbf{xy}$ is understood to take input bit-vectors $\mathbf{x}, \mathbf{y}$ and output a bit-vector labeled $\mathbf{xy}$. We label the internal variables of a circuit $\mathbf{C}$ using the superscript C, for example: $t_{i,j}^{C}$.

Definition Given a set of polynomials Φ over a set of variables $\{x_1, x_2, \ldots, x_n\}$ and a field K, a *polynomial calculus* refutation of Φ is a sequence of polynomials ending with the polynomial 1 such that each line is either in Φ or is derived from the previous lines using the inference rules of linear combination and multiplication by a monomial m:

$$\frac{p \qquad q}{\alpha p + \beta q}\ (\alpha, \beta \in K), \qquad \frac{p}{m \cdot p}\ .$$

The polynomials $x^2 - x$ are also included as axioms for each variable x so that it only takes Boolean values. The polynomial p is interpreted to mean the equation $p = 0$.

Definition Given a set of 0-1 linear inequalities Φ over a set of variables $\{x_1, x_2, \ldots, x_n\}$, a *cutting planes* refutation of Φ is a sequence of 0-1 linear inequalities ending with the inequality $0 \geq 1$ such that each line is either in Φ or is derived from

the previous lines using the inference rules of positive linear combination

$$\frac{\sum_i a_i x_i \geq b \qquad \sum_i a'_i x_i \geq b'}{\sum_i (\alpha a_i + \beta b_i) x_i \geq \alpha b + \beta b'}$$

where $\alpha, \beta \geq 0$, and the *division rule*

$$\frac{\sum_i (c \cdot a_i) x_i \geq b}{\sum_i a_i x_i \geq \lceil \frac{b}{c} \rceil} .$$

The *literal axioms* $-x \geq -1$ and $x \geq 0$ are also included for each variable x. Throughout this paper we will use "=" as shorthand for the two equivalent "$\leq, \geq$" inequalities.

A. A polynomial calculus lower bound for bit-extraction

The bit-extraction lower bound discussed in the introduction follows directly from the following polynomial calculus lower bound for *subset-sum equations* due to Impagliazzo, Pudlak and Sgall.

Theorem II.1 ([25])**.** *Let $c_1, \ldots, c_n$ be nonzero real numbers such that no subset sums to the real number m. Then the equation $m - \sum_{i=1}^{n} c_i x_i = 0$ has no polynomial calculus refutation of degree $\lceil n/2 \rceil$ in the field of real numbers.*

Theorem II.2 ([25])**.** *Suppose that Φ is a set of polynomials of degree at most $\sqrt{n}$, where n is the number of variables appearing in Φ. Let d denote the minimum refutation degree of Φ, and M denote the minimum number of monomials in a refutation of Φ, and assume that $M \geq 3$. Then $M \geq \exp\left((d-1)^2/4n\right)$*

We combine Theorems II.1 and II.2 to demonstrate the weakness of polynomial calculus in extracting bit-level properties from word-level ones.

Corollary II.3. *For a fixed integer k, any polynomial calculus refutation of the system of two polynomials:*

$$f := \sum_{i=0}^{n-1} 2^i (s_i - s'_i)$$
$$g := s_k - s'_k - 1$$

contains at least $e^{n/4-1} \approx 2^{0.36n}$ monomials.

Proof. Define the polynomial $f' := \sum_{i \neq k} 2^i (s_i - s'_i) + 2^k$. Observe that Theorem II.1 gives us a degree lower bound of $n-1$ on refutations of the polynomial $\{f'\}$. Theorem II.2 translates this into a monomial size lower bound of $e^{n/4-1}$. The reduction below lifts this lower bound on $\{f'\}$ to the polynomials $\{f, g\}$.

We show that a length l polynomial calculus refutation of the polynomials $\{f, g\}$ may be converted into a length l refutation of the polynomial $\{f'\}$ without increasing the number of monomials in each line as follows: First notice that the polynomials f, f' are equivalent modulo the polynomial $g = s_k - s'_k - 1$. Given a PC refutation of $\{f, g\}$, we reduce each line by g (which effectively sets $s_k = 1$ and $s'_k = 0$), only reducing the number of monomials, to produce a refutation of $\{f'\}$. □

As a consequence of this corollary, polynomial calculus cannot derive $s_k = s'_k$ from the first equation using fewer than $e^{n/4-1}$ monomials. In comparison, cutting planes has small derivations that produce all of the bit-equalities.

Proposition II.4. *There is an $O(n)$-length cutting planes derivation of all n bit-equalities $s_i = s'_i$ from the equation $\sum_{i=0}^{n-1} 2^i s_i - \sum_{i=0}^{n-1} 2^i s'_i = 0$.*

Proof. We extract the individual bit-equalities in the low-to-high sequence $s_0 = s'_0, s_1 = s'_1, \ldots s_{n-1} = s'_{n-1}$. Recall that in cutting planes, the equation $\sum_{i=0}^{n-1} 2^i s_i - \sum_{i=0}^{n-1} 2^i s'_i = 0$ is represented by two inequalities. Take the inequality $\sum_{i=0}^{n-1} 2^i s_i - \sum_{i=0}^{n-1} 2^i s'_i \geq 0$, and use the literal axioms on s_0, s'_0 to get $\sum_{i=1}^{n-1} 2^i s_i - \sum_{i=1}^{n-1} 2^i s'_i \geq -1$. Divide this by 2 to get $\sum_{i=1}^{n-1} 2^{i-1} s_i - \sum_{i=1}^{n-1} 2^{i-1} s'_i \geq 0$. Finally, use linear combination to multiply this by 2 and add it to the equation $\sum_{i=0}^{n-1} 2^i s'_i - \sum_{i=0}^{n-1} 2^i s_i \geq 0$ to obtain the result $s'_i - s_i \geq 0$. A symmetric derivation gives $s_i - s'_i \geq 0$. □

B. Adder and multiplier circuit constructions

Definition A *ring identity* $L = R$ denotes a pair of ring expressions L, R that can be transformed into each other using commutativity, distributivity and associativity.

To prove that a given ring identity $L = R$ holds for some choice of circuit implementations for $+$ and $\times$, we use these implementations to build a circuit $\mathbf{L}$ representing the expression L and another circuit $\mathbf{R}$ for the expression R. The goal of our cutting planes proofs is to show that the resulting output bit-vectors $\mathbf{L}, \mathbf{R}$ are equal bit-by-bit, i.e., that $L_i = R_i$ holds for every i.

Circuits for addition and multiplication

The circuits that we will consider are built using *adders* that output, in binary, the sum of three input bits. A (1-bit) adder is encoded as follows:

Definition For an adder A with inputs a_0, a_1, a_2, the outputs c, d are determined by the equation $a_0 + a_1 + a_2 - 2c - d = 0$. We call c the *carry-bit* and d the *sum-bit*.

In our circuits, each variable belongs to a column, i. The variables in column i have a *weight* of 2^i. Each adder is also assigned to a column. An adder A belonging to the i-th column takes three input bits from column i and outputs a *sum-bit* into column i and a *carry-bit* into column $i+1$. The equation associated with A ensures that the weight of its outputs is equal to the weight of its inputs.

a) Ripple-Carry Adder: Figure 1 shows the design of a ripple-carry adder $\mathbf{x} + \mathbf{y}$, which takes in two bitvectors $\mathbf{x}, \mathbf{y}$ and outputs their sum in binary.

b) Multiplier circuits: Figure 2 shows the design of an array multiplier and our labeling of the internal circuit variables. The first phase of an array multiplier is a common part of many multiplier designs: the circuit computes a *tableau* of partial products $t_{i,j} = x_i \wedge y_j$ for each pair of input bits x_i and y_j. In the second phase, n ripple-carry adders are arranged

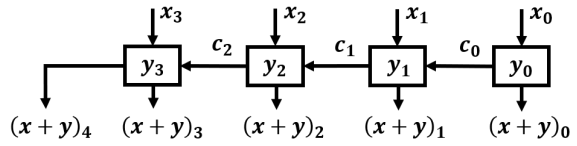

Fig. 1. A 4-bit ripple-carry adder adding $\mathbf{x}, \mathbf{y}$. Each box represents a full adder with incoming arrows and the labels in the boxes representing inputs and outgoing arrows representing outputs.

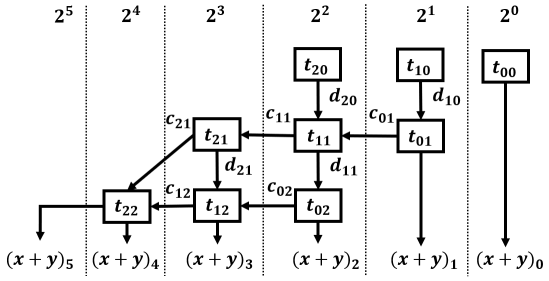

Fig. 2. 3-bit array multiplier.

in a grid-like fashion in order to sum the n rows of the tableau. A closely related variant of the array multiplier is the diagonal multiplier, shown in Figure 3, which routes its carry bits to the next row instead of the same row.

Wallace-tree multipliers sum the tableau by arranging a network of adders in a tree-like structure. This log-depth structure reduces the number of rows in the tableau to 2, then uses an adder circuit to compute the final sum. In hardware implementations, this final stage adder is typically a *carry-lookahead adder*, so that the full multiplier has logarithmic depth. However, carry-lookahead adders use non full-adder components, which will lie outside the scope of this paper. The Wallace-tree multipliers in this paper will use ripple-carry adders for this final stage, so that the multiplier contains only full adder components.

III. Array multiplier commutativity in $O(n^2)$ steps

In this section, we give $O(n^2)$-length derivations for the word-level equivalence of the output bit-vectors $\mathbf{xy}$ and $\mathbf{yx}$ for both polynomial calculus and cutting planes. For polynomial calculus, this proof was, in essence, previously written down in [44].

Swapping the order of inputs $\mathbf{x}, \mathbf{y}$ to a multiplier has the effect of reversing the order of tableau values in each column. In particular we have the equalities $t^{xy}_{i,j} = t^{yx}_{j,i}$ between tableau variables. The next lemma shows that from these bit-level equalities we can derive the word-level equality of the output bit-vectors $\mathbf{xy}$ and $\mathbf{yx}$ using only $O(n^2)$ linear combination steps. As both polynomial calculus and cutting planes can carry out such steps (recall that cutting planes represents "=" using two inequalities), they can both perform this proof.

Lemma III.1. *Suppose that we have two n-bit array multipliers $\mathbf{xy}$ and $\mathbf{yx}$ implementing the two sides of the commutativity relation $xy = yx$. Further, suppose that we are given the n^2 equalities between the tableau variables $t^{xy}_{i,j} = t^{yx}_{j,i}$. Then there is a derivation in degree 1 and length $3n^2 + 1$ of the equation $\sum_{i=0}^{n-1} 2^i (xy)_i - \sum_{i=0}^{n-1} 2^i (yx)_i = 0$ that only uses linear combinations.*

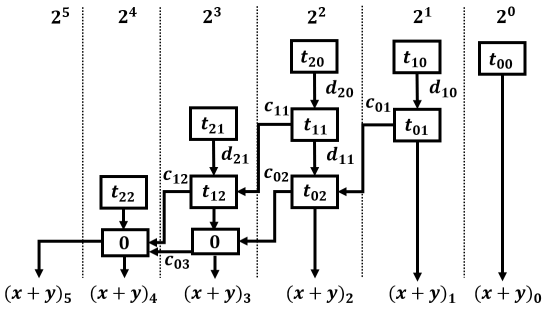

Fig. 3. 3-bit diagonal multiplier.

Proof. We first derive two "conservation of weight" equations for the circuits $\mathbf{xy}$ and $\mathbf{yx}$ that state that the total weight of a multiplier's output bits is the same as the total weight of its tableau bits. We obtain these by adding up the adder constraints, weighting them so that the internal circuit variables cancel. For an adder in column i corresponding to a constraint $a_0 + a_1 + a_2 - 2c - d = 0$, this weighting simply scales the constraint up by a factor of 2^i. Once all the n^2 adder equations for an array multiplier $\mathbf{xy}$ have been summed together, we will arrive at an equation stating that the weight of tableau variables $t^{xy}_{i,j}$ is the same as the weight of the output variables $\mathbf{xy}$. After repeating the same steps for the multiplier $\mathbf{yx}$, we arrive at the two equations

$$\Big(\sum_{i,j=0}^{n-1} 2^{i+j} t^{xy}_{i,j}\Big) - \Big(\sum_{i=0}^{2n-1} 2^i (xy)_i\Big) = 0$$
$$\Big(\sum_{i,j=0}^{n-1} 2^{i+j} t^{yx}_{j,i}\Big) - \Big(\sum_{i=0}^{2n-1} 2^i (yx)_i\Big) = 0$$

having used $2n^2$ linear combination steps. We then use a total of n^2 further derivation steps to replace $t^{yx}_{j,i}$ by $t^{xy}_{i,j}$ for each pair $i, j \in [0, n-1]$ in the latter equation. Finally, we subtract the two equations to finish the derivation. □

Theorem III.2. *There is a polynomial calculus derivation in length $4n^2 + 1$, and also a cutting planes derivation in length $14n^2 + 2$, of the equation $\sum_{j=0}^{n-1} 2^i (xy)_i - \sum_{j=0}^{n-1} 2^i (yx)_i = 0$ from the array multiplier circuits $\mathbf{xy}$ and $\mathbf{yx}$.*

Proof. Given the previous lemma, to complete our derivation we need to obtain the tableau equalities $t^{xy}_{i,j} = t^{yx}_{j,i}$. In polynomial calculus, we get each equality with one subtraction step with the equations $t^{xy}_{i,j} = x_i y_j$ and $t^{yx}_{j,i} = y_j x_i$. So deriving these equalities takes an additional n^2 polynomial calculus steps.

In cutting planes, it takes 3 linear inequalities (clauses) to represent a constraint $t^{xy}_{i,j} = x_i y_j$. From these, we can derive that $t^{xy}_{i,j} = t^{yx}_{j,i}$ in eight steps. Hence, deriving the tableau

equalities takes $8n^2$ cutting planes steps. Afterwards, it takes two cutting planes steps to carry out each of the $3n^2+1$ linear combination steps of Lemma III.1. □

In cutting planes we can use proposition II.4 to prove bit-level equality from the equation $\sum_{i=0}^{n-1} 2^i(xy)_i - \sum_{i=0}^{n-1} 2^i(yx)_i = 0$, which gives the following corollary.

Corollary III.3. *There is a length-$O(n^2)$ cutting planes derivation yielding all of the $2n$ equalities $(xy)_i = (yx)_i$ from the array multiplier circuits* $\mathbf{xy}$ *and* $\mathbf{yx}$.

For other ring identities such as distributivity, we no longer have straightforward equalities between the tableau variables on either side of the identity. For distributivity, the natural generalization of these tableau variable equalities contains nonlinear terms. Before we give our cutting planes proofs, we introduce the (k,d)-cutting planes proof system in the next section as a convenient way to work with nonlinear terms within cutting planes.

IV. (k,d)-Cutting planes proofs

Our cutting planes multiplier proofs will be written in a more convenient format that allows for a limited number of nonlinear terms in each inequality. Although cutting planes proofs only allow the use of linear inequalities, we will also be able to efficiently represent a large class of *nonlinear* Boolean inequalities using sets of linear inequalities.

Definition We say that a polynomial inequality ϕ on the Boolean variables X is (k,d)-nonlinear if is written in the form

$$\ell(X) + \sum_{i=1}^{k} \ell_i m_i \geq b$$

where $\ell(X)$ is an integer linear form (i.e., $\ell(X) = \sum_i c_i x_i$), each $\ell \in \{\ell_1, \ldots, \ell_k\}$ is a non-negative integer linear form (i.e., $\ell = \sum_i c_i x_i$ and each $c_i \geq 0$), each m_i is a degree at most $d-1$ monomial with coefficient $+1$ or -1, containing only variables disjoint from ℓ_i, and lastly, b is an integer.

We emphasize that this proof system distinguishes between inequalities ϕ and ϕ' that are semantically equivalent, but are syntactically different due to different factorizations. For example, the inequality $(x_1+x_2)y_1 \geq b$ is *not* considered to be the same as the inequality $x_1y_1+x_2y_1 \geq b$. The first inequality is $(1,2)$-nonlinear while the second is $(2,2)$-nonlinear. In simulating (k,d)-nonlinear inequalities by ordinary linear ones, these two inequalities will be represented by two different (though semantically equivalent) sets of linear inequalities.

Definition Let $\mathcal{CP}^{+(k,d)}$ denote the (k,d)*-cutting planes* proof system. Each line is a (k,d)-nonlinear inequality on a set of Boolean variables $\{x_i\}$. Its rules are as follows. The *literal axioms* are the same as in $\mathcal{CP}$: for each variable x_i we have $x_i \geq 0$ and $-x_i \geq -1$. The division rule and linear combination rule from $\mathcal{CP}$ generalize as one would expect. Writing $\ell(X) = \sum_i (c \cdot a_i) x_i$:

$$\frac{\sum_i (c \cdot a_i) x_i + \sum_i (c \cdot \ell_i) m_i \geq b}{\sum_i a_i x_i + \sum_i \ell_i m_i \geq \lceil \frac{b}{c} \rceil}$$

And for any $\alpha, \beta \in \mathbb{N}$, as long as the result is (k,d)-nonlinear:

$$\frac{\sum_i \ell_i m_i + \ell(X) \geq b, \quad \sum_j \ell'_j m'_j + \ell'(X) \geq b'}{\sum_i \alpha \ell_i m_i + \alpha \ell(X) + \sum_j \beta \ell'_j m'_j + \beta \ell'(X) \geq \alpha b + \beta b'}$$

The *factoring rule* is that if the (k,d)-nonlinear inequality ϕ contains two terms $\ell m, \ell' m$ with the same monomial m, then we can factor these into term $(\ell + \ell')m$. Syntactically:

$$\frac{\ell(X) + \sum_i \ell_i m_i + \ell m + \ell' m \geq b}{\ell(X) + \sum_i \ell_i m_i + (\ell + \ell') m \geq b}.$$

The *distributing rule* is the reverse of the factoring rule, except we can only distribute "one at a time". For example, rewriting $(10y_1 + 8y_2)x_1x_2x_3 \to 10y_1x_1x_2x_3 + 8y_2x_1x_2x_3$ would require 8 applications of the distributing rule below. For a non-negative linear form $\ell = \sum_i c_i x_i$, where each $c_i \geq 0$, define $\max(\ell) = \sum_i c_i$. Because of a technical detail related to the simulation size, we require that the two inequalities $\max(\ell) \geq \ell \geq 0$ have been derived from the literal axioms before making this inference:

$$\frac{\ell(X) + \sum_i \ell_i m_i + (\ell + y_r) m_p \geq b \qquad \max(\ell) \geq \ell \geq 0}{\ell(X) + \sum_i \ell_i m_i + \ell m_p + y_r m_p \geq b}.$$

The *multiplication rule* permits the multiplication of an inequality ϕ by a variable z, provided that the resulting inequality ϕz is (k,d)-nonlinear. Decomposing $\ell(X) = \ell(X)^+ - \ell(X)^-$ into a sum of positive terms $\ell(X)^+$ and negative terms $\ell(X)^-$:

$$\frac{\ell(X)^+ - \ell(X)^- + \sum_i \ell_i m_i \geq b}{\ell(X)^+ \, z - \ell(X)^- \, z + \sum_i \ell_i m_i z - bz \geq 0}.$$

Theorem IV.1. *Fix a pair of positive integers $k \geq 1$ and $d \geq 2$. The cutting planes proof system $\mathcal{CP}$ p-simulates the $\mathcal{CP}^{+(k,d)}$ proof system. In particular, a $\mathcal{CP}^{+(k,d)}$ proof of s lines can be simulated by a cutting planes proof of at most $(k+4)d^k s$ lines.*

The idea of the proof of Theorem IV.1 is to find a tight set of at most d linear upper bounds for each degree d nonlinear term. To simulate an inequality with k nonlinear terms of degree at most d, we use the set of at most d^k linear inequalities obtained by plugging in every combination of upper bounds for each nonlinear term. The full details of this simulation can be found in [34].

V. Optimal cutting planes multiplier proofs

In the previous proof of commutativity, we were able to give cutting planes proofs without including nonlinear terms. However, when giving proofs for distributivity and other larger identities, nonlinear terms are difficult to avoid. This is where the (k,d)-cutting planes format is convenient for expressing $O(n^2)$ length cutting planes proofs of distributivity. We generalize these proofs for distributivity to obtain $O(n^2)$ length proofs for a large class of degree two ring identities.

In the first half of these proofs, we sum up the adder-constraints in each ripple-carry adder circuit $\mathbf{x}+\mathbf{y}$ to derive the "conservation of weight" equation $\sum_i 2^i(x_i + y_i) = \sum_i 2^i(x+y)_i$, and also in each multiplication circuit $\mathbf{xy}$ to derive the "conservation of weight" equation $\sum_{i,j} 2^{i+j} t_{i,j}^{xy} = \sum_i 2^i(xy)_i$.

This section focuses on the second half of the proof, where the goal is to show that both sides hold equal weight in their multiplier tableau variables. The idea is to derive an equation $\rho(i,j)$ relating the (i,j)-th tableau entry of each multiplier. Fixing j and summing these equations along i gives an equation $\rho(j)$ relating the j-th rows of each multiplier. Finally, adding together the equations $\rho(j)$ yields the desired equation for the full multiplier tableaus.

A. Distributivity

Theorem V.1. *There is a length $O(n^2)$ $\mathcal{CP}$ proof that the circuits $(\mathbf{x}+\mathbf{y})\mathbf{z}$ and $\mathbf{xz}+\mathbf{yz}$ for length n bit-vectors $\mathbf{x},\mathbf{y},\mathbf{z}$ have equal outputs.*

Proof. We will give a length $O(n^2)$ proof in $\mathcal{CP}^{+(5,2)}$. By Theorem IV.1, this implies that there is an equivalent cutting planes proof that is only a constant factor larger. We begin with the following lemma, which gives a small derivation that the weight of the j-th row of the multiplier $(\mathbf{x}+\mathbf{y})\mathbf{z}$ is the same as the combined weight of the j-th rows of multipliers $\mathbf{xz}$ and $\mathbf{yz}$.

Lemma V.2. *For each $j \in [0, n-1]$ there is a length $O(n)$ derivation in $\mathcal{CP}^{+(5,2)}$ of the equality $\rho(j)$, defined as: $\sum_{i=0}^{n} 2^{i+j} \cdot t_{i,j}^{(x+y)z} = \sum_{i=0}^{n-1} 2^{i+j} \cdot (t_{i,j}^{xz} + t_{i,j}^{yz})$. from the circuits $(\mathbf{x}+\mathbf{y})\mathbf{z}$ and $\mathbf{xz}+\mathbf{yz}$.*

Proof. Fix $j \in [0, n-1]$. We give a constant length derivation for each cell-wise constraint $\rho(i,j)$, defined for $i \in [1, n-1]$ as

$$t_{i,j}^{(x+y)z} = t_{i,j}^{xz} + t_{i,j}^{yz} + c_{i-1}^{x+y} z_j - 2c_i^{x+y} z_j$$

and defined for $i = 0$ and $i = n$ the same way, absent the non-existing variables c_{-1}^{x+y}, c_n^{x+y}, $t_{n,j}^{xz}$ and $t_{n,j}^{yz}$. Adding up the constraints $\rho(i,j)$ will yield $\rho(j)$.

Start with the equation $x_i + y_i + c_{i-1}^{x+y} - 2c_i^{x+y} - (x+y)_i = 0$, given by the i-th adder in the ripple-carry adder $(\mathbf{x}+\mathbf{y})$. Multiplying this equation by z_j, we obtain the $(5,2)$-nonlinear equation $x_i z_j + y_i z_j + c_{i-1}^{x+y} z_j - 2c_i^{x+y} z_j - (x+y)_i z_j = 0$. Substituting in the tableau variables $t_{i,j}^{(x+y)z}, t_{i,j}^{xz}, t_{i,j}^{yz}$ gives us $\rho(i,j)$.

To derive $\rho(j)$ we add together the constraints $\rho(i,j)$ so that the carry terms telescope: We start with $\rho(n,j)$. Use linear combination to derive the equation $2\rho(n,j) + \rho(n-1,j)$:

$$2t_{n,j}^{(x+y)z} + t_{n-1,j}^{(x+y)z} = t_{n-1,j}^{xz} + t_{n-1,j}^{yz} + c_{n-1}^{x+y} z_j.$$

Repeating this step for $\rho(n-2,j), \ldots, \rho(0,j)$ gives $\rho(j)$. □

The rest of the proof combines equations $\rho(j)$ given by Lemma V.2 with the conservation of weight equations. We first observe that combining the conservation of weight equations gives us, in a constant number of steps, the two equations

$$\sum_{i=0}^{2n} 2^i \cdot (xz+yz)_i = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} 2^{i+j} \cdot (t_{i,j}^{xz} + t_{i,j}^{yz}) \quad (1)$$

$$\sum_{i=0}^{2n} 2^i \cdot ((x+y)z)_i = \sum_{j=0}^{n-1} \sum_{i=0}^{n} 2^{i+j} \cdot t_{i,j}^{(x+y)z}. \quad (2)$$

Sum all of the equalities $\rho(j)$ to derive the equation ρ, stating that both sides have equal weight in their tableau variables: $\sum_{i,j} 2^{i+j} \cdot (t_{i,j}^{xz} + t_{i,j}^{yz}) = \sum_{i,j} 2^{i+j} \cdot t_{i,j}^{(x+y)z}$. Combine this with equations 1 and 2 to obtain the final result: $\sum_i 2^i \cdot (xz+yz)_i = \sum_i 2^i \cdot ((x+y)z)_i$. □

Notice that we only used the structure of the multipliers $(\mathbf{x}+\mathbf{y})\mathbf{z}$, $\mathbf{xz}$ and $\mathbf{yz}$ to derive the conservation of weight equations relating the sum of tableau variables to the output of the multiplier. The above proof is thereby compatible with any integer multiplier for which we can efficiently derive these conservation of weight equations. For example, we obtain $O(n^2)$ length proofs for Wallace tree multipliers using a final stage ripple-carry adder. In comparison, the best prior proof known for, say, checking that the middle pair of bits of an array multiplier and a Wallace tree multiplier are equal, was the quasi-polynomial size $n^{O(\log n)}$ resolution proof given in [6].

Reversing the order of multiplier inputs only has the effect of permuting the order of tableau variables, so the above proof also immediately generalizes to identities like $z(x+y) = zx + xz$ that mix distributivity and commutativity.

B. 2-Colorable identities

In this section we state some theorems that we can obtain by generalizing the ideas behind the proofs for the identity $(x+y)z = xz + yz$ to provide $O(n^2)$ length cutting planes proofs for larger instances of distributivity. The proofs of these theorems may be found in [34].

Theorem V.3. *Let $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_s}$ and $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_{s'}}$ be length n bit-vectors. Define the circuit $\mathbf{L}$ as $(\mathbf{x_1} + \mathbf{x_2} + \ldots + \mathbf{x_s})(\mathbf{y_1} + \mathbf{y_2} + \ldots + \mathbf{y_{s'}})$. Also define the circuit $\mathbf{R}$ as $\left(\sum_{\alpha,\beta} \mathbf{x}_\alpha \mathbf{y}_\beta\right)$, representing the fully expanded version of $\mathbf{L}$. There is a length $O(n^2)$ cutting planes proof that circuits L and R have equal outputs.*

Theorem V.3 gives us $O(n^2)$ cutting planes proofs for fixed ring identities that can be written as sum of independent bit-vector distributing or factoring steps. However, there exist identities such as $x(y+z) + wz = xy + (x+w)z$ which cannot be decomposed into a sum of independent distributing and factoring components. Nevertheless, we can still give an $O(n^2)$ length proof of this identity. We define the notion of a *2-colorable* degree two identity to identify the general class of ring identities for which our technique can derive $O(n^2)$ length proofs.

Definition Let $L = R$ be a degree two ring identity. A *2-coloring* for $L = R$ is an assignment of either the color red or

blue to each bit-vector, with multiplicity (so a bit-vector may appear twice with different colors), such that: (1) each bit-vector in a sub-expression $(\mathbf{x_1} + \mathbf{x_2} + \ldots + \mathbf{x_r})$ has the same color as the bit-vector representing the sub-expression, (2) two sub-expressions that are multiplied together have opposite colors, and (3) the colored version of $L = R$, where a blue input bit-vector colored blue $\mathbf{x_i}$ is distinguished from its red counterpart $\mathbf{x_i}$, is still a valid ring identity.

For example, $(\mathbf{x} + \mathbf{y})\mathbf{z} = \mathbf{xz} + \mathbf{zy}$ has the 2-coloring $(\mathbf{x} + \mathbf{y})\mathbf{z} = \mathbf{xz} + \mathbf{zy}$. The more general form of distributivity in Theorem V.3 clearly always has an 2-coloring. Lastly, the identity $\mathbf{x}(\mathbf{y} + \mathbf{z}) + \mathbf{wz} = \mathbf{xy} + \mathbf{z}(\mathbf{x} + \mathbf{w})$ has the 2-coloring $\mathbf{x}(\mathbf{y} + \mathbf{z}) + \mathbf{wz} = \mathbf{xy} + \mathbf{z}(\mathbf{x} + \mathbf{w})$. An example of an identity without a 2-coloring is $\mathbf{x}(\mathbf{y} + \mathbf{z}) + \mathbf{w}(\mathbf{x} + \mathbf{y}) = \mathbf{y}(\mathbf{x} + \mathbf{w}) + \mathbf{x}(\mathbf{z} + \mathbf{w})$.

Theorem V.4. *Let $L = R$ be a 2-colorable degree two ring identity on length n bit-vectors $\mathbf{x_1}, \ldots, \mathbf{x_s}$. There is a length $O(n^2)$ cutting planes proof that the circuits $\mathbf{L}$ and $\mathbf{R}$ have equivalent outputs.*

VI. Experiments

The goal of our experiments was to evaluate the potential of using cutting planes solvers to reason with mixtures of multiplication and bit-level logic. Such problems are a key weakness of using a SAT-based approach to "bit-blasting". We found several types of problems where pseudo-Boolean solvers performed well out-of-the-box. These include checking the word-level equivalence, commutativity, or correctness of different multipliers, extracting bit-equalities from word-level equalities, and verifying nonlinear bit-vector inequalities.

In our experiments, we used an Intel Core i7-6700K CPU at 4.00GHz with a memory limit of 8GB. The wall-clock time limit was set to 1200 seconds. We list experiment times in seconds (wall-clock time) and write TO if the time limit of 1200 seconds was exceeded. Our benchmarks are available at [35].

We used two pseudo-Boolean solvers, each equipped with a different form of cutting planes reasoning. The first, Sat4j-CP [33], employs *saturation* in its conflict analysis. The second solver, RoundingSat [21], [48], instead uses division; we used the new multi-precision version of the solver for which we could also log and separately verify the derivations it used.

Our experiments focused on integer multipliers with n-bit inputs and $2n$ bits of output. We report results on three different circuits to represent multiplication: array, diagonal, and Wallace-tree multipliers with final stage ripple-carry adder. As noted in the introduction, we directly represent the adder constraints as two inequalities instead of as a set of clauses and define a "spec-equation multiplier" without a circuit by simply using the specification equation

$$\sum_{i,j=0}^{n-1} 2^{i+j} t_{i,j}^{xy} - \sum_{i=0}^{2n-1} 2^i (xy)_i = 0$$

TABLE I

Time to prove equivalences between multipliers using Sat4j and RoundingSat. We give the time to prove equivalence the word-level, the time to extract the individual bits of the word-level equivalence, and the sum of these gives the total time to prove bit-level equivalence. We compare performance to the algebraic approach of [28]

Instance	n	Sat4j-CP Word-level	RoundingSat Extract	RoundingSat Bit-level
array $x \cdot y = y \cdot x$	32	6	1	7
	64	8	6	14
	128	25	41	66
	256	171	158	329
diagonal $x \cdot y = y \cdot x$	32	7	1	8
	64	7	6	13
	128	25	41	66
	256	172	158	330
array spec-eqn	32	6	1	7
	64	18	6	24
	128	135	41	176
	256	TO	N/A	TO
diagonal spec-eqn	32	4	1	5
	64	18	6	24
	128	129	41	170
	256	TO	N/A	TO
diagonal $\equiv$ array	32	2	1	3
	64	5	6	11
	128	16	41	57
	256	102	158	260
Instance	n	Gröbner [28] Word-level	Extract	Bit-level
gate-array $x \cdot y = y \cdot x$	32	1	N/A	N/A
	64	3	N/A	N/A
	128	27	N/A	N/A
	256	273	N/A	N/A
gate-array spec-eqn	32	1	N/A	N/A
	64	2	N/A	N/A
	128	14	N/A	N/A
	256	136	N/A	N/A

Pseudo-Boolean benchmarks with array, diagonal, or spec-eqn used our generator. Gate-level array multipliers were generated by Boolector [43].

TABLE II

Time to prove equivalences with Wallace tree multipliers using Sat4j and RoundingSat.

Instance	n	Sat4j-CP Word-level	RoundingSat Extract	RoundingSat Bit-level
Wallace $x \cdot y = y \cdot x$	16	1	1	2
	32	5	1	6
	48	TO	N/A	TO
	64	TO	N/A	TO
Wallace $\geq$ spec-eqn	16	1	N/A	N/A
	32	5	N/A	N/A
	48	65	N/A	N/A
	64	360	N/A	N/A
array $\equiv$ Wallace	16	1	1	2
	32	2	1	3
	48	45	3	48
	64	41	6	47

along with the partial product constraints $t_{i,j}^{xy} = x_i y_j$. In these two ways, the pseudo-Boolean format allows us to "bit-blast" multiplication, along with other word-level functions, to a higher-level description than CNF while maintaining full bit-precision.

Our first set of experiments, presented in Table I, uses

the pseudo-Boolean solvers Sat4j-CP and RoundingSat to verify the word-level and bit-level equivalence of different multiplier circuits. More precisely, we use Sat4j-CP to prove an equation of the form $\sum_i 2^i(s_i - s'_i) = 0$ stating that the total weight of the outputs $\mathbf{s}, \mathbf{s}'$ is the same for the two multipliers. Then we have RoundingSat deduce, from this equation, each equality $s_i = s'_i$ individually in order to prove equivalence at the bit-level. Performance on bit-extraction scaled particularly well with the right choice of pseudo-Boolean solver, as shown in Table III, which also includes a comparison with the theoretical lower bound we showed for algebraic methods. Using these two steps, we can efficiently check the commutativity of array, diagonal, and Wallace-tree multipliers, as well as several equivalences between array, diagonal, and spec-equation multipliers. We can also check some of these properties of Wallace-tree multipliers for up to 32 or 64 bits.

An important step for showing word-level equivalence was to do some basic pre-processing to find equivalent partial products ($t_{i,j}^{xy}$ variables). Adding these equivalences was key to obtaining efficient solve times in Sat4j-CP. In contrast, we found that adding these equivalences did not help SAT-based solvers. We note that most bit-vector solvers, and many SAT solvers, already perform similar pre-processing to find equivalent variables; current pseudo-Boolean solvers based on cutting planes do not yet have such pre-processing.

To provide some context for these results, we compared the performance of our pseudo-Boolean approach to the algebraic approach of [28], which is currently the fastest method for verifying these properties. We replicated their verification of the commutativity and correctness of a simple gate-level array multiplier "btor", generated by Boolector, by using their tool, AMulet, in our environment to obtain the solve times at the bottom of Table I. We note that AMulet, is also capable of similarly fast solve times for more complicated gate-level multipliers such as Booth-encoded Wallace-tree multipliers. We direct interested readers to [28] for further experiments using the algebraic approach to verify commutativity, correctness, and equivalence of these other gate-level multiplier architectures.

Current pseudo-Boolean solvers have limited reasoning capabilities for these lower level multipliers. In particular, these solvers degenerate to SAT-based reasoning when given a CNF input. Our focus is not so much on verifying a large spectrum of multiplier circuits as on bit-vector solving, where we are free to choose the most efficient way to represent bit-vector multiplication.

We see that for simple array and diagonal multipliers, our approach (on adder-level multipliers) achieves comparable times to the algebraic approach (on gate-level multipliers) for proving commutativity and word-level equivalence. Furthermore, we are able to efficiently extract each of the individual bit-level equalities that a word-level equality implies.

For Wallace-tree multipliers with a final stage ripple-carry adder (wt-rca), we could check its equivalence with an array for 64 bits within 1 minute. We could also check commutativity for 32 bits in 5 seconds. However, we hit time-out on larger instances of 48 or 64 bits. We were also unable to completely verify the equivalence of a wt-rca and spec equation multiplier for 32-bit instances, though we could show that the the output of the wt-rca is at least as large as the output of the spec equation in 5 seconds. We see that Sat4j-CP has a harder time with these more complicated multiplier architectures.

TABLE III

Time in seconds to prove the equality $s_0 = s'_0$ from the equation $\sum_{i=0}^{n-1} 2^i(s_i - s'_i) = 0$ for the cutting planes solvers RoundingSat (RS) and Sat4j-CP, compared to the SAT-based solvers Sat4j-Res and NaPS [49]. We also compare with the polynomial calculus lower bound given by Corollary II.3.

n	RS	Sat4j-CP	Sat4j-Res	NaPS	#monomials
12	.001	7	.4	.1	7
16	.001	TO	3	2	20
20	.001		81	39	54
24	.001		TO	208	148
28	.002			Error	403
32	.002				1096
64	.009				3×10^6
128	.04				2×10^{13}
256	.2				2×10^{27}
512	.4				1×10^{55}

Our other experiments, presented in Table IV, use the solver RoundingSat to verify some nonlinear bit-vector inequalities involving untruncated multiplication and the operations "|" for bit-wise OR, "&" for bit-wise AND. We use these bit-wise operations to apply the *bit masks* "$\mid k$" and "$\&k$", where k is set to the constant alternating bit-string $(10)^{(n/2)}$. (This value was an arbitrary choice that contains a mix of 1s and 0s; we observed similar performance across all solvers with other values of k.) The inequalities listed follow from thinking of "|" and "&" as, respectively, computing the bit-wise maximum and minimum of their inputs.

We compare RoundingSat's performance on these inequalities against the bit-vector solvers Boolector, Yices2, Z3 and CVC4. Our inputs to these bit-vector solvers used the word-level format SMT-LIB2 [3] to allow for full use of word-level reasoning and other non-SAT capabilities. We found that these bit-vector solvers (with the exception of Boolector) generally exceeded the time limit at 20 bits. On the other hand, when we "bit-blasted" multiplication using the spec-equation, RoundingSat outperformed all of the bit-vector solvers, with the exception of last inequality $(x \mid k)(y+1) \geq ky+x$, where Boolector won out by a few bits.

VII. Conclusions & Directions

In this paper, we have described a new approach to deciding nonlinear bit-vector formulas: include 1-bit adders among the set of essential building blocks along with the usual Boolean operations and express properties using pseudo-Boolean formulas rather than CNF formulas during "bit-blasting". We have shown, both experimentally and in principle, how pseudo-Boolean solvers based on cutting planes reasoning, when given these new bit-blasted formulas, can achieve levels of performance comparable to, or better than, the best alterna-

TABLE IV

Time to prove bit-vector inequalities containing both multiplication and bit-level operations. We compare RoundingSat (RS), Boolector 3.2.0 (Btor), Z3 4.8.7, Yices 2.6.2 and CVC4.

Inequality	n	RS	Btor	Z3	Yices2	CVC4
$(x \mid k)z \geq kz$	16	17	14	21	31	44
	20	11	136	TO	TO	TO
	24	16	TO			
	28	501				
	32	TO				
$kz \geq (x\&k)z$	16	.06	10	15	172	31
	20	.5	117	1154	TO	TO
	24	.7	TO	TO		
	28	.6				
	32	.6				
$(x \mid k)z \geq (x\&k)z$	16	.2	14	22	31	44
	20	7	TO	TO	TO	TO
	24	2				
	28	629				
	32	TO				
$(x \mid z)(z \mid k) \geq kx$	16	.008	19	43	114	50
	20	.05	351	TO	TO	TO
	24	.1	TO			
	28	.2				
	32	.2				
$kx \geq (x\&z)(z\&k)$	16	.04	10	32	100	48
	20	.07	243	TO	TO	TO
	24	.1	TO			
	28	23				
	32	7				
$(x \mid k)(y+1)$ $\geq ky + x$	16	.4	25	29	38	118
	20	TO	342	TO	TO	TO
	24		TO			

Bit-vector k is the value $(10)^{(n/2)}$. & is bit-wise AND, | is bit-wise OR.

tive methods on a number of natural multiplier verification examples.

In particular, we have given $O\big(n^2\big)$-length cutting planes proofs for a broad class of properties of multipliers, matching the optimal efficiency of the best Gröbner basis algorithms for these properties at the word level, while also being able to extract bit-level properties. Importantly, Gröbner basis algorithms are not known to be able to extract such bit-level properties efficiently: We have shown that such methods require exponential time to extract bit-level consequences from word-level properties.

An interesting open question is whether polynomial size cutting planes proofs can be found for degree three identities such as associativity. Although word-level associativity has an $O(n^2)$ length proof in polynomial calculus, this cannot be used to show the individual bit-level equalities.

We also have shown experimentally that for several of these properties on inputs of up to 256 bits — namely, commutativity, correctness, and equivalence — pseudo-Boolean solvers can achieve performance comparable to that of the best algebraic solvers at the word-level, and, in contrast to algebraic methods, also solve these problems at the bit-level.

Finally, we have experimentally verified a number of crafted bit-vector inequalities, each involving a mixture of multiplication and bit-wise operations and have shown that our pseudo-Boolean approach can achieve much better verification performance than several of the best current bit-vector solvers.

The idea of using pseudo-Boolean solving for verifying nonlinear bit-vector formulas appears not to have been explored previously. One possible explanation for this is that when pseudo-Boolean solvers are run purely on CNF inputs, their reasoning collapses to that of CDCL SAT solvers, only much less efficient ones because of the more involved data structures and algorithms required in the pseudo-Boolean case. Our use of 1-bit adders as fundamental structures is critical to achieving the performance that we obtain.

Conflict-driven pseudo-Boolean solvers are still at a relatively early stage of development, especially compared to the 25+ years of concerted effort directed at optimizing Gröbner basis algorithms and CDCL solvers. In particular, there is quite some variation in the different forms of conflict analysis methods used, and some of these methods have been shown to be quite weak. In fact, many solvers, such as NaPS [49] and Open-WBO [40], do not use any cutting planes reasoning and instead reduce the problem to SAT. Other shortcomings in the cutting planes reasoning used in current solvers are discussed in ([20], [23], [53]). In our experiments, different conflict analysis methods worked best on different problems. For example, we found that the saturation-based solver Sat4j-CP worked much better than RoundingSat for checking word-level equalities. On the other hand, the division-based solver RoundingSat significantly outperformed Sat4j-CP when tasked with extracting bit-equalities, and also for checking bit-vector inequalities. This is in contrast with CDCL solvers where the best ideas for conflict analysis have largely converged on a single method that is used by all of the currently best solvers.

We view this work as providing a "call to arms" for pseudo-Boolean solver development, focusing especially on features that will be useful in verification of these kinds of bit-vector problems. In particular, though our experiments validate the pseudo-Boolean approach in principle, none of the solvers we used allowed us to verify the properties for which we provided more complex cutting planes proofs in Section V. Thus, there is substantial scope for developing new methods and heuristics for pseudo-Boolean solving that can carry out much more of this cutting planes reasoning in practice.

ACKNOWLEDGMENTS

Paul Beame and Vincent Liew's research was supported in part by NSF-SHF grant CCF-1714593. Jo Devriendt, Jan Elffers, and Jakob Nordström were funded by the Swedish Research Council (VR) grant 2016-00782, and Jakob Nordström was also supported by the Independent Research Fund Denmark (DFF) grant 9040-00389B. Some development work on RoundingSat was done using computational resources provided by the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N) at Umeå University.

References

[1] S. Anderson, "Bit twiddling hacks," https://graphics.stanford.edu/~seander/bithacks.html, accessed: 2020-05-15.

[2] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovi'c, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, Jul. 2011, pp. 171–177, snowbird, Utah. [Online]. Available: http://www.cs.stanford.edu/~barrett/pubs/BCD+11.pdf

[3] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB standard: Version 2.5," Department of Computer Science, The University of Iowa, Tech. Rep., 2015.

[4] R. J. Bayardo Jr. and R. Schrag, "Using CSP look-back techniques to solve real-world SAT instances," in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, Jul. 1997, pp. 203–208.

[5] P. Beame, H. A. Kautz, and A. Sabharwal, "Towards understanding and harnessing the potential of clause learning," *J. Artif. Intell. Res. (JAIR)*, vol. 22, pp. 319–351, 2004.

[6] P. Beame and V. Liew, "Towards verifying nonlinear integer arithmetic," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 238–258.

[7] ——, "Toward verifying nonlinear integer arithmetic," *J. ACM*, vol. 66, no. 3, pp. 22:1–30, June 2019. [Online]. Available: https://doi.org/10.1145/3319396

[8] A. Biere, "Collection of Combinational Arithmetic Miters Submitted to the SAT Competition 2016," in *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2016-1. University of Helsinki, 2016, pp. 65–66.

[9] ——, "Weaknesses of CDCL solvers," in *Fields Institute Workshop on Theoretical Foundations of SAT Solving*, August 2016, http://www.fields.utoronto.ca/talks/weaknesses-cdcl-solvers.

[10] A. Biere, M. Kauers, and D. Ritirc, "Challenges in verifying arithmetic circuits using computer algebra," in *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '17)*, 2017.

[11] A. Blake, "Canonical expressions in Boolean algebra," Ph.D. dissertation, University of Chicago, 1937.

[12] R. Brummayer and A. Biere, "Boolector: An efficient SMT solver for bit-vectors and arrays," in *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, ser. TACAS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 174–177. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00768-2-16

[13] D. Chai and A. Kuehlmann, "A fast pseudo-Boolean constraint solver," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 305–317, March 2005.

[14] S. Chakraborty, A. Gupta, and R. Jain, "Matching multiplications in bit-vector formulas," in *Verification, Model Checking, and Abstract Interpretation*, A. Bouajjani and D. Monniaux, Eds. Cham: Springer International Publishing, 2017, pp. 131–150.

[15] M. Ciesielski, T. Su, A. Yasin, and C. Yu, "Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[16] M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the Groebner basis algorithm to find proofs of unsatisfiability," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 174–183. [Online]. Available: http://doi.acm.org/10.1145/237814.237860

[17] W. Cook, C. Coullard, and G. Turán, "On the complexity of cutting-plane proofs," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 25 – 38, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X87900394

[18] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[19] B. Dutertre, "Yices 2.2," in *Computer-Aided Verification (CAV' 14)*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, July 2014, pp. 737–744.

[20] J. Elffers, J. Giráldez-Cru, J. Nordström, and M. Vinyals, "Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers," in *Theory and Applications of Satisfiability Testing – SAT 2018*, O. Beyersdorff and C. M. Wintersteiger, Eds. Cham: Springer International Publishing, 2018, pp. 75–93.

[21] J. Elffers and J. Nordström, "Divide and conquer: Towards faster pseudo-Boolean solving," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 1291–1299. [Online]. Available: https://doi.org/10.24963/ijcai.2018/180

[22] A. Franzén, A. Cimatti, A. Nadel, R. Sebastiani, and J. Shalev, "Applying SMT in symbolic execution of microcode," in *Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '10. Austin, TX: FMCAD Inc, 2010, pp. 121–128. [Online]. Available: http://dl.acm.org.offcampus.lib.washington.edu/citation.cfm?id=1998496.1998520

[23] S. Gocht, J. Nordström, and A. Yehudayoff, "On division versus saturation in pseudo-Boolean solving," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 1711–1718. [Online]. Available: https://doi.org/10.24963/ijcai.2019/237

[24] A. Haken, "The intractability of resolution," *Theoretical Computer Science*, vol. 39, pp. 297 – 308, 1985, third Conference on Foundations of Software Technology and Theoretical Computer Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397585901446

[25] R. Impagliazzo, P. Pudlák, and J. Sgall, "Lower bounds for the polynomial calculus and the Gröbner basis algorithm," *Computational Complexity*, vol. 8, no. 2, pp. 127–144, 1999.

[26] H. S. W. Jr., *Hacker's Delight, Second Edition*. Pearson Education, 2013. [Online]. Available: http://www.hackersdelight.org/

[27] M. Katelman and J. Meseguer, "vlogsl: A strategy language for simulation-based verification of hardware," in *Hardware and Software: Verification and Testing*, S. Barner, I. Harris, D. Kroening, and O. Raz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–145.

[28] D. Kaufmann, "Formal verification of multiplier circuits using computer algebra," Ph.D. dissertation, Johannes Kepler University Linz, 2020.

[29] D. Kaufmann, A. Biere, and M. Kauers, "SAT, computer algebra, multipliers," in *Vampire 2018 and Vampire 2019. The 5th and 6th Vampire Workshops*, ser. EPiC Series in Computing, L. Kovács and A. Voronkov, Eds., vol. 71. EasyChair, 2020, pp. 1–18.

[30] D. Kaufmann, M. Kauers, A. Biere, and D. Cok, "Arithmetic verification problems submitted to the SAT Race 2019," in *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, M. Heule, M. Järvisalo, and M. Suda, Eds., vol. B-2019-1. University of Helsinki, 2019, p. 49.

[31] G. Kovásznai, A. Fröhlich, and A. Biere, "Complexity of fixed-size bit-vector logics," *Theory of Computing Systems*, vol. 59, no. 2, pp. 323–376, Aug 2016. [Online]. Available: https://doi.org/10.1007/s00224-015-9653-1

[32] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.

[33] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *JSAT*, vol. 7, pp. 59–6, 01 2010.

[34] V. Liew, "A path paved by proof complexity towards verifying nonlinear integer arithmetic," Ph.D. dissertation, University of Washington, 2020.

[35] ——, "Ring Benchmarks," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3987457

[36] A. Mahzoon, D. Große, and R. Drechsler, "Polycleaner: Clean your polynomials before backward rewriting to verify million-gate multipliers," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

[37] ——, "Revsca: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, June 2019, pp. 1–6.

[38] F. Marić and P. Janičić, "Urbiva: Uniform reduction to bit-vector arithmetic," in *Automated Reasoning*, J. Giesl and R. Hähnle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 346–352.

[39] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, May 1999, preliminary version in *ICCAD '96*.

[40] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver,," in *Theory and Applications of Satisfiability Testing – SAT 2014*, C. Sinz and U. Egly, Eds. Cham: Springer International Publishing, 2014, pp. 438–445.

[41] R. Michel, A. Hubaux, V. Ganesh, and P. Heymans, "An SMT-based approach to automated configuration," in *SMT 2012. 10th International Workshop on Satisfiability Modulo Theories*, ser. EPiC Series in Computing, P. Fontaine and A. Goel, Eds., vol. 20. EasyChair, 2013, pp. 109–119. [Online]. Available: https://easychair.org/publications/paper/bKGs

[42] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, Jun. 2001, pp. 530–535.

[43] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0 system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 53–58, 2014 (published 2015).

[44] D. Ritirc, A. Biere, and M. Kauers, "Column-wise verification of multipliers using computer algebra," in *Formal Methods in Computer-Aided Design, FMCAD 2017, Vienna, Austria, October 02-06, 2017.*, D. Stewart and G. Weissenbacher, Eds. IEEE, 2017, pp. 23–30.

[45] ——, "Improving and extending the algebraic approach for verifying gate-level multiplies," in *Design, Automation and Test in Europe (DATE'18)*, 2018.

[46] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, vol. 12, no. 1, pp. 23–41, Jan. 1965.

[47] A. Romano and D. Engler, "Expression reduction from programs in a symbolic binary executor," in *Model Checking Software*, E. Bartocci and C. R. Ramakrishnan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 301–319.

[48] "RoundingSat," https://gitlab.com/miao_research/roundingsat.

[49] M. Sakai and H. Nabeshima, "Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers," *IEICE Transactions on Information and Systems*, vol. E98.D, no. 6, pp. 1121–1127, 2015.

[50] A. A. R. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal verification of integer multipliers by combining Gröbner basis with logic reduction," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016*, Dresden, Germany, March 2016, pp. 1048–1053. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=7459464

[51] M. Temel, A. Slobodová, and W. A. Hunt, "Automated and scalable verification of integer multipliers," in *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12224. Springer, 2020, pp. 485–507. [Online]. Available: https://doi.org/10.1007/978-3-030-53288-8_23

[52] A. Urquhart, "Hard examples for resolution," *J. ACM*, vol. 34, no. 1, p. 209–219, Jan. 1987. [Online]. Available: https://doi.org/10.1145/7531.8928

[53] M. Vinyals, J. Elffers, J. Giráldez-Cru, S. Gocht, and J. Nordström, "In between resolution and cutting planes: A study of proof systems for pseudo-Boolean SAT solving," in *Theory and Applications of Satisfiability Testing – SAT 2018*, O. Beyersdorff and C. M. Wintersteiger, Eds. Cham: Springer International Publishing, 2018, pp. 292–310.

[54] C. Yu, M. Ciesielski, and A. Mishchenko, "Fast algebraic rewriting based on and-inverter graphs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1907–1911, Sep. 2018.