Reducing Bit-Vector Polynomials to SAT using Gröbner Bases*

Thomas Seed[1], Andy King[1], and Neil Evans[2]

[1] University of Kent, Canterbury, CT2 7NF, UK
[2] AWE Aldermaston, Reading, RG7 4PR, UK

Abstract. We address the satisfiability of systems of polynomial equations over bit-vectors. Instead of conventional bit-blasting, we exploit word-level inference to translate these systems into non-linear pseudo-boolean constraints. We derive the pseudo-booleans by simulating bit assignments through the addition of (linear) polynomials and applying a strong form of propagation by computing Gröbner bases. By handling bit assignments symbolically, the number of Gröbner basis calculations, along with the number of assignments, is reduced. The final Gröbner basis yields expressions for the bit-vectors in terms of the symbolic bits, together with non-linear pseudo-boolean constraints on the symbolic variables, modulo a power of two. The pseudo-booleans can be solved by translation into classical linear pseudo-boolean constraints (without a modulo) or by encoding them as propositional formulae, for which a novel translation process is described.

Keywords: Gröbner Bases · Bit-Vectors · Modulo Arithmetic · SMT

1 Introduction

Some of the most influential algorithms in algebraic computation, such as Buchberger's algorithm [7] and Collin's Cylindrical Algebraic Decomposition algorithm [8], were invented long before the advent of SMT. SMT itself has evolved from its origins in SAT into a largely independently branch of symbolic computation. Yet the potential of cross-fertilising one branch with the other has been repeatedly observed [1, 6, 10], and a new class of SMT solvers is beginning to emerge that apply both algebraic and satisfiability techniques in tandem [15, 16, 23]. The problem, however, is that algebraic algorithms do not readily fit into the standard SMT architecture [22] because they are not normally incremental or backtrackable, and rarely support learning [1].

For application to software verification, the SMT background theory of bit-vectors is of central interest. Solvers for bit-vectors conventionally translate bit-vector constraints into propositional formulae by replacing constraints with propositional circuits that realise them, a technique evocatively called bit-blasting.

* Supported by an AWE grant on the verification of AVR microcontroller code.

However, particularly for constraints involving multiplication, the resulting formulae can be prohibitively large. Moreover, bit-blasting foregoes the advantages afforded by reasoning at the level of bit-vectors [3, 14].

In this paper we present a new architecture for solving systems of polynomial equalities over bit-vectors. Rather than converting to SAT and bit-blasting, the method sets bits in order of least significance through the addition of certain polynomials to the system. Computing a Gröbner basis [5] for the resulting system effects a kind of high-level propagation, which we have called bit-sequence propagation, in which the values of other bits can be automatically inferred. Furthermore, we show how the procedure can be carried out with symbolic truth values without giving up bit-sequence propagation, thus unifying Gröbner basis calculations that would otherwise be separate.

Once all bits are assigned truth values (symbolic or otherwise), the resulting Gröbner basis prescribes an assignment to the bit-vectors which is a function of the symbolic truth values. The remaining polynomials in the basis relate the symbolic truth values and correspond to non-linear pseudo-boolean constraints modulo a power of two. These constraints can be solved either by translation into classical linear pseudo-boolean constraints (without a modulo) or else by encoding them as propositional formulae, for which a novel translation process is described. Either way, the algebraic Gröbner basis computation is encapsulated in the phase that emits the pseudo-boolean constraints, hence the Gröbner basis engine [5] does not need to be backtrackable, incremental or support learning. The approach can be extended naturally to handle polynomial disequalities since the bit-vectors in the disequalities can be reformulated in terms of the symbolic variables of the equalities, and the disequalities forced to hold as well by virtue of a Tseytin transform [24]. Overall, the architecture provides a principled method for compiling high-level polynomials to low-level pseudo-boolean constraints.

In summary, this paper makes the following contributions:

- We specialise a Gröbner basis algorithm for integers modulo 2^ω [5], using the concept of rank [21], introducing an algorithmic modification to ease the computational burden of computing Gröbner bases;
- We introduce bit-sequence propagation, in which an individual bit is set to a truth value 0 or 1 by adding a suitable (linear) polynomial to the system and then the effect on other bits is inferred by computing a Gröbner basis;
- We show how bit assignments can be handled symbolically in order to unify distinct Gröbner basis computations, eventually yielding a residue system of non-linear pseudo-boolean constraints;
- We show how the resulting pseudo-boolean systems can be solved by employing a novel rewrite procedure for converting non-linear modulo pseudo-booleans to propositional formulae.

The paper is structured as follows: Section 2 illustrates bit-sequence propagation through a concrete example. The supporting concepts of Gröbner bases for modulo integers and pseudo-boolean encoding are detailed in Section 3. Experimental results are given in Section 4 and Section 5 surveys related work.

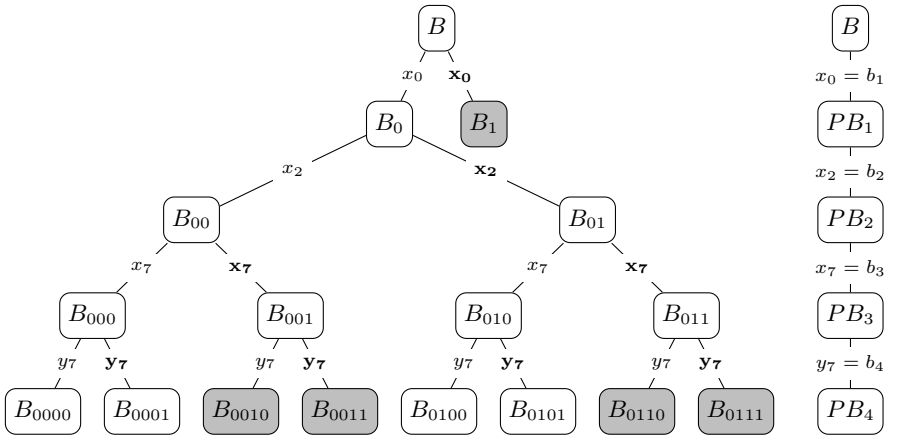

Fig. 1. Bit-assignments and word-level propagation: 0/1 bits and symbolic bits

2 Bit-sequence Propagation in a Nutshell

Classically, that is for polynomials over algebraically closed fields, unsatisfiability can be decided by Hilbert's Nullstellensatz [9]. This equates unsatisfiability with the existence of a non-zero constant polynomial in a Gröbner basis for the polynomials. The concept of Gröbner basis is inextricably linked with that of an ideal [9]. The ideal for a given system (set) of polynomials is the least set closed under the addition of polynomials drawn from the set and multiplication of an arbitrary polynomial with a polynomial from the set; an ideal shares the same zeros as the system from which it is derived, but is not finite. A Gröbner basis is merely a finite representation of an ideal, convenient because, among other things, it enables satisfiability to be detected, at least over a field.

Unary bit-vectors constitute a field, but Nullstellensatz does not hold for bit-vectors with multiple bits. To see this, consider the polynomial equation $x^2 + 2 = 0$ where the arithmetic is 3-bit (modulo 8). Any solution x to this equation must be even. But, $0^2{+}2 = 2$, $2^2{+}2 = 6$, $4^2{+}2 = 2$ and $6^2{+}2 = 6$. Hence $x^2 + 2 = 0$ has no solutions, yet the Gröbner basis $\{x^2 + 2\}$ does not contain a non-zero constant polynomial. Moreover, even for a Gröbner basis of a satisfiable system, such as $\{x^2{+}4\}$, the solutions to the system cannot be immediately read off from the basis. The force of these observations is that Gröbner bases need to be augmented with search to test satisfiability and discover models. To illustrate this we consider a more complicated system:

$$B = \left\{ \begin{array}{rr} y^2 + 120x^2 + 123x + 48 = 0, & yx + 65x^2 + 50x + 32 = 0, \\ 2y + 63x^2 + 59x + 128 = 0, & x^3 + 135x^2 + 100x + 64 = 0, \\ 64x^2 + 192x = 0 & \end{array} \right\}$$

where $x, y \in \mathbb{Z}_{256}$. Henceforth we follow convention and omit $= 0$ from systems.

2.1 Solving using 0/1 truth values

Since $\mathbb{Z}_{256}$ is finite, this system can be solved by viewing the problem [20] as a finite domain constraint satisfaction problem. In this setting, each bit-vector is associated with a set of values that is progressively pruned using word-level constraint propagation rules. The search tree in the left-hand side of Fig 1 illustrates how pruning is achieved by setting and inferring bits in the order of least-significance, starting with the bits of x then those of y. On a left branch of the tree one bit, x_i or y_j, is set to 0; on a right branch the bit is set to 1 (indicated in bold). Each node is labelled with a Gröbner basis that encodes the impact of setting a bit on all other bits. Gröbner bases are indexed by their position in the tree. Grey nodes correspond to the solutions of B.

Computing B_0 Setting the least significant bit of x to 0 can be achieved by imposing $x = 2w$ for some otherwise unconstrained variable w. Hence, we add $2w - x$ to B and compute a Gröbner basis with respect to the lexicographical ordering on variables $w \succ y \succ x$, yielding:

$$\left\{\begin{array}{llll} wx + 86x + 96, & 2w + 255x, & y^2 + 219x + 48, & \\ yx + 134x + 96, & 2y + 231x + 64, & x^2 + 172x + 192, & 64x \end{array}\right\}$$

To eliminate dependence on w, polynomials involving w are removed, giving:

$$B_0 = \left\{\begin{array}{lll} y^2 + 219x + 48, & yx + 134x + 96, & 2y + 231x + 64, \\ x^2 + 172x + 192, & 64x & \end{array}\right\}$$

Note that B_0 contains $64x$ (representing $64x = 0$) which indicates that x is a multiple of 4. Thus bit 1 is also 0, although we did not actively impose it.

Now, observe the constraint $64x = 0$ implies $0 = 2^6(x-0)$ hence $x-0 = 2^2w'$ for some w'. To set the next bit to 0, put $w' = 2w$ which gives $x-0 = 8w$ yielding the polynomial $8w - x$. Otherwise, to set the next bit put $w' = 2w + 1$ giving the polynomial $8w - x + 4$.

Computing B_{00} Augmenting B_0 with $8w - x$, calculating a Gröbner basis, and then eliminating w gives:

$$B_{00} = \left\{\begin{array}{lll} y^2 + 219x + 48, & yx + 128, & 2y + 231x + 64, \\ x^2, & 2x + 160 & \end{array}\right\}$$

Since B_{00} includes $2x+160$ (representing $2x+160 = 0$) it follows that only bit 7 is undetermined. To constrain it, observe $0 = 2(x - 48)$ thus $x - 48 = 2^7w'$ for some w'. Putting $w' = 2w$ gives $x-48 = 256w = 0$ hence the polynomial $x-48$. Conversely, putting $w' = 2w+1$ gives $x - 48 = 256w + 128 = 128$ thus $x - 176$.

Computing B_{000} and B_{001} Adding $x - 48$ and $x - 176$ to B_{00}, computing a Gröbner basis, and eliminating w (a vacuous step), respectively yields:

$$B_{000} = \left\{y^2 + 64, \quad 2y + 144, \quad x + 208\right\} \; B_{001} = \left\{y^2 + 192, \quad 2y + 16, \quad x + 80\right\}$$

Both systems contain a single constraint on x which uniquely determines its value, hence we move attention to y. Both B_{000} and B_{001} contain equations with leading terms $2y$ and thus only bit 7 of y must be constrained. Following the same procedure as before, we obtain:

$$B_{0000} = \begin{Bmatrix} y + 200, \\ x + 208, \\ 128 \end{Bmatrix} \; B_{0001} = \begin{Bmatrix} y + 72, \\ x + 208, \\ 128 \end{Bmatrix} \; B_{0010} = \begin{Bmatrix} y + 136, \\ x + 80 \end{Bmatrix} \; B_{0011} = \begin{Bmatrix} y + 8, \\ x + 80 \end{Bmatrix}$$

These Gröbner bases all completely constrain x and y, hence are leaf nodes. Note that B_{0000} and B_{0001} contain the non-zero, constant polynomial 128, indicating unsatisfiability. Hence, only B_{0010} and B_{0011} actually yield solutions (highlighted in grey), namely $x \mapsto 176, y \mapsto 120$ and $x \mapsto 176, y \mapsto 248$ respectively.

Computing B_* The general principle is that if $2^k(x - \ell)$ is in the basis and ω is the bit width, then the linear polynomial $2^{\omega-k+1}w - x + \ell$ is added for some fresh w to set the next undermined bit to 0. Conversely, to set the next bit to 1, the polynomial $2^{\omega-k+1}w - x + 2^{\omega-k} + \ell$ is added. We name this tactic bit-sequence propagation. Using this tactic to flesh out the rest of the tree gives the following satisfiable bases (also marked in grey in the figure):

$$B_1 = \{y + 183, x + 91\} \quad B_{0110} = \{y + 158, x + 92\} \quad B_{0111} = \{y + 30, x + 92\}$$

yielding $x \mapsto 165, y \mapsto 73$, $x \mapsto 164, y \mapsto 98$ and $x \mapsto 164, y \mapsto 226$ respectively.

2.2 Solving using symbolic truth values

To reduce the total number of Gröbner basis calculations, we observe that it is sufficient to work with symbolic bits. The right-hand side of Figure 1 illustrates how this reduces the number of bases calculated to 4, albeit at the cost of carrying symbolic bits in the basis. Bit-sequence propagation generalises via the single rule: if $2^k(x - \ell)$ is in the basis and ω is the bit width, then the polynomials $2^{\omega-k+1}w - x + 2^{\omega-k}b + \ell$ and $b^2 - b$ are added to the basis. This sets the next undermined bit to the symbolic value b; the polynomial $b^2 - b$ merely asserts that each symbolic b can only be 0 or 1. This construction gives:

$$PB_1 = \begin{Bmatrix} y^2 + 219x + 216b_1 + 48, & yx + 6x + 181b_1 + 96, & yb_1 + 183b_1, \\ 2y + 103x + 203b_1 + 64, & x^2 + 44x + 139b_1 + 192, & xb_1 + 91b_1, \\ 64x + 192b_1, & b_1^2 + 255b_1 & \end{Bmatrix}$$

$$\vdots$$

$$PB_4 = \begin{Bmatrix} y + 128b_4 + 192b_3 + 214b_2 + 153b_1 + 200, & x + 12b_2 + 255b_1 + 80, \\ b_4^2 + 255b_4, & 128b_4b_1 + 128b_1, & b_3^2 + 255b_3, & 64b_3b_1, \\ 128b_3 + 128b_2 + 128, & b_2^2 + 255b_2, & 2b_2b_1 + 254b_1, & b_1^2 + 255b_1 \end{Bmatrix}$$

The final PB_4 expresses x and y as combinations of b_4, b_3, b_2 and b_1:

$$y \equiv_{256} -128b_4 - 192b_3 - 214b_2 - 153b_1 - 200 \qquad x \equiv_{256} -12b_2 - 255b_1 - 80$$

Observe that the remaining polynomials are non-linear pseudo-boolean constraints over b_4, b_3, b_2 and b_1 modulo 256. The polynomials $b_i^2 + 255b_i$, which assert that each b_i is binary, are subsequently ignored.

2.3 Solving using SAT

These pseudo-booleans can be simplified by observing that when all coefficients in the constraint are divisible by a power of 2 then the modulo can be lowered:

$$\begin{aligned} 128b_4b_1 + 128b_1 \equiv_{256} 0 &\iff b_4b_1 + b_1 \equiv_2 0 \\ 64b_3b_1 \equiv_{256} 0 &\iff b_3b_1 \equiv_4 0 \\ 128b_3 + 128b_2 + 128 \equiv_{256} 0 &\iff b_3 + b_2 + 1 \equiv_2 0 \\ 2b_2b_1 + 254b_1 \equiv_{256} 0 &\iff b_2b_1 + 127b_1 \equiv_{128} 0 \end{aligned}$$

Since the reduced versions of the first and third constaints are modulo 2 they can be mapped immediately to the propositional formulae:

$$\begin{aligned} b_4b_1 + b_1 \equiv_2 0 &\iff (b_4 \wedge b_1) \oplus b_1 \\ b_3 + b_2 + 1 \equiv_2 0 &\iff \neg(b_3 \oplus b_2) \end{aligned}$$

where the negation is introduced because of the constant 1. The second and fourth constraints cannot be handled so directly because the modulus is not 2. However, for the second, we can use the fact that the left-hand side is a single term to infer either b_3 or b_1 must be 0, yielding the formula $\neg b_3 \vee \neg b_1$. Finally, for the fourth constraint, we do a case split on b_2. Setting $b_2 = 0$ simplifies the constraint to $127b_1 \equiv_{128} 0$, from which $b_1 = 0$ is inferred. Conversely, setting $b_2 = 1$ simplifies the constraint to $128b_1 \equiv_{128} 0$ which is vacuous. Overall, we derive the formula $(\neg b_2 \wedge \neg b_1) \vee b_2$ for the fourth constraint. There are 5 truth assignments for the formula assembled from the above 4 sub-formulae, yielding 5 assignments to x and y that concur with those given previously.

The reasoning exemplified here has been distilled into a series of rules, presented in Section 3.8, for encoding non–linear modulo pseudo-booleans into SAT. An alternative approach finds the values for b_4, b_3, b_2 and b_1 using a cutting-plane pseudo-boolean solver [19] alongside a modulo elimination transformation [13, Section 3]. Regardless of the particular method employed to solve this system, observe that search has been isolated in the SAT/pseudo-boolean solver; the Gröbner bases are calculated in an entirely deterministic fashion.

3 Theoretical underpinnings

In its classical form, Buchberger's algorithm for computing Gröbner bases is only applicable to polynomials over fields [18], such as rationals and complex numbers, where every non-zero element has a multiplicative inverse. However, integers modulo 2^ω only have this property for $\omega = 1$, as even numbers do not have multiplicative inverses for $\omega > 1$. Nevertheless, a variant of Buchberger's algorithm has been reported that is applicable to modulo integers with respect to arbitrary moduli [5]. This section specialises this variant to integers modulo 2^ω, exploting the concept of rank [21] to efficiently determine divisibility over modulo integers. An algorithmic refinement is also reported that reduces the number of calculated S-polynomials, the key to improving efficiency. The section concludes with a formalisation of the rules for encoding non-linear modulo pseudo-booleans into propositional formulae, first introduced in Section 2.3.

3.1 Modulo Integers

Let $\mathbb{N}$ (resp., $\mathbb{N}_+$), denote the non-negative (resp., positive) integers, $\omega \in \mathbb{N}_+$ be the bit-width, $m = 2^\omega$ and $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ denote the integers modulo m. The rank of $x \in \mathbb{Z}_m$ [21] is defined: $\mathsf{rank}_\omega(x) = \max\{j \in \mathbb{N} \mid 2^j \text{ divides } x\}$ if $x > 0$ and ω otherwise. Rank can be computed by counting the number of trailing zeros in an integer's binary representation [26].

Example 1. In $\mathbb{Z}_{256}$ when $\omega = 8$, $\mathsf{rank}_8(0) = 8$, $\mathsf{rank}_8(15) = 0$ and $\mathsf{rank}_8(56) = 3$.

If $x \in \mathbb{Z}_m$, $x \neq 0$ then $x = 2^{\mathsf{rank}_\omega(x)} d$ where $d = x/2^{\mathsf{rank}_\omega(x)}$ is odd. This is referred to as the rank decomposition of x. If $x \in \mathbb{Z}_m$ then x is said to be invertible if there exists $x^{-1} \in \mathbb{Z}_m$, necessarily unique, such that $xx^{-1} = 1$. This occurs iff x is odd, in which case x^{-1} can be found as a stationary point of the sequence $y_1 = 1$, $y_{n+1} = y_n(2 - xy_n)$ [21]. For $x_1, x_2 \in \mathbb{Z}_m$, x_1 is said to be divisible by x_2 if $x_1 = yx_2$ for some $y \in \mathbb{Z}_m$. This occurs iff $\mathsf{rank}_\omega(x_1) \geq \mathsf{rank}_\omega(x_2)$, in which case, letting $x_i = 2^{k_i} d_i$ be the rank decomposition of x_i, it follows $y = 2^{k_1 - k_2} d_1 {d_2}^{-1}$.

3.2 Polynomials and Ideals

A monomial is an expression $\boldsymbol{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ where $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ is a vector of variables and $\boldsymbol{\alpha} = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathbb{N}^n$. A term is an expression $c\boldsymbol{x}^{\boldsymbol{\alpha}}$ where $c \in \mathbb{Z}_m$. A polynomial is either 0 or an expression $t_1 + \cdots + t_s$ where each t_i is a term. In this expression, we assume all t_i have non-zero coefficients and distinct monomials, since terms with 0 coefficients can be removed and terms with the same monomial can be collected by summing their coefficients. The set of all polynomials over $\mathbb{Z}_m$ is denoted $\mathbb{Z}_m[\boldsymbol{x}]$. For $p \in \mathbb{Z}_m[\boldsymbol{x}]$ and $\boldsymbol{a} \in \mathbb{Z}_m^n$, $[\![p]\!](\boldsymbol{a})$ denotes the result of substituting a_i for each x_i in p and evaluating the result.

An ideal is a set $I \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ such that $\Sigma_{i=1}^s u_i p_i \in I$ for all $s \in \mathbb{N}$, $p_i \in I$ and $u_i \in \mathbb{Z}_m[\boldsymbol{x}]$. If $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ then $\langle P \rangle = \{\sum_{i=1}^s u_i p_i \mid s \geq 0, p_i \in P, u_i \in \mathbb{Z}_m[\boldsymbol{x}]\}$ is the ideal generated by P; if $I = \langle P \rangle$ then P is said to be a basis for I. The solution (zero) set of $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ is defined: $\gamma(P) = \{\boldsymbol{a} \in \mathbb{Z}_m^n \mid \forall p \in P.\ [\![p]\!](\boldsymbol{a}) = 0\}$. Note that $\gamma(\langle P \rangle) = \gamma(P)$, hence if P_1, P_2 are both bases for the same ideal I, then $\gamma(P_1) = \gamma(I) = \gamma(P_2)$. Given a set $P_1 \subseteq \mathbb{Z}_m[\boldsymbol{x}]$, it is thus desirable to find a basis P_2 for $\langle P_1 \rangle$ which exposes information about the zeros of P_1. The concept of Gröbner basis makes this idea concrete.

3.3 Leading Terms

Let $\prec$ denote the lexicographical ordering over monomials defined by $\boldsymbol{x}^{\boldsymbol{\alpha}} \prec \boldsymbol{x}^{\boldsymbol{\beta}}$ if $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ and $\alpha_i < \beta_i$ at the first index i where $\alpha_i \neq \beta_i$. Note that this ordering depends on the order of the variables in $\boldsymbol{x}$. If $p \in \mathbb{Z}_m[\boldsymbol{x}]$ then either $p = 0$ or else $p = c\boldsymbol{x}^{\boldsymbol{\alpha}} + q$ for some polynomial q, where all terms $d\boldsymbol{x}^{\boldsymbol{\beta}}$ appearing in q satisfy $\boldsymbol{x}^{\boldsymbol{\beta}} \prec \boldsymbol{x}^{\boldsymbol{\alpha}}$. In the latter case, the leading term, coefficient and monomial of p are defined $\mathsf{lt}(p) = c\boldsymbol{x}^{\alpha}$, $\mathsf{lc}(p) = c$ and $\mathsf{lm}(p) = \boldsymbol{x}^{\boldsymbol{\alpha}}$ respectively.

Example 2. Let $p_1 = y^2 + 3yx \in \mathbb{Z}_{256}[y, x]$ and $p_2 = 3xy + y^2 \in \mathbb{Z}_{256}[x, y]$. Note that p_1 and p_2 consist of the same terms, yet $\mathsf{lt}(p_1) = y^2$, $\mathsf{lc}(p_1) = 1$ and $\mathsf{lm}(p_1) = y^2$ and $\mathsf{lt}(p_2) = 3xy$, $\mathsf{lc}(p_2) = 3$ and $\mathsf{lm}(p_2) = xy$.

3.4 Reduction

Leading terms give a rewrite procedure over polynomials. First, note that if $t_1 = c_1\boldsymbol{x}^{\boldsymbol{\alpha_1}}$, $t_2 = c_2\boldsymbol{x}^{\boldsymbol{\alpha_2}}$ are terms then there exists a term t such that $t_1 = tt_2$ iff $\boldsymbol{\alpha_1} \geq \boldsymbol{\alpha_2}$ component-wise and c_1 is divisible by c_2; in this case, $t = d\boldsymbol{x}^{\boldsymbol{\beta}}$ where $c_1 = dc_2$ and $\boldsymbol{\beta} = \boldsymbol{\alpha_1} - \boldsymbol{\alpha_2}$. With divisibility in place, reducibility can be defined:

Definition 1. *Let $p, q, r \in \mathbb{Z}_m[\boldsymbol{x}]$, $p, q \neq 0$. Then, p is said to be reducible by q to r, denoted $p \to_q r$, if $\mathsf{lt}(p) = c\boldsymbol{x}^{\boldsymbol{\alpha}}\,\mathsf{lt}(q)$ and $p = c\boldsymbol{x}^{\boldsymbol{\alpha}}q + r$ for some term $c\boldsymbol{x}^{\boldsymbol{\alpha}}$.*

Reducibility lifts to sets $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ by $\to_P = \bigcup_{p\in P} \to_p$ and $\to_P^+$ (resp. $\to_P^*$) is the transitive (resp. transitive, reflexive) closure of $\to_P$. If $p \to_P^+ r$ for some r then p is said to be reducible by P, otherwise irreducible by P, denoted $p \not\to_P$.

Example 3. Let $p = yx^2 + 2yx + 5y + x$ and $P = \{p_1, p_2\} \subseteq \mathbb{Z}_{256}[y, x]$ where $p_1 = yx+3y$ and $p_2 = 4y+x$. Then, $\mathsf{lt}(p) = yx^2 = x\,\mathsf{lt}(p_1)$ and $p = xp_1+r_1$ where $r_1 = 255yx + 5y + x$. Similarly, $\mathsf{lt}(r_1) = 255yx = 255\,\mathsf{lt}(p_1)$ and $r_1 = 255p_1 + r_2$ where $r_2 = 8y + x$. Finally, $\mathsf{lt}(r_2) = 8y = 2\,\mathsf{lt}(p_2)$ and $r_2 = 2(4y + x) + r_3$ where $r_3 = 255x$. Thus, $p \to_{p_1} r_1 \to_{p_1} r_2 \to_{p_2} r_3 = 255x$ hence $p \to_P^+ 255x$.

Note if $p \to_P^+ r$ then r has a strictly smaller leading term than p. Moreover, if $p \in \mathbb{Z}_m[\boldsymbol{x}]$, $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ and $p \to_P^* 0$ then $p \in \langle P\rangle$. The converse of this does not hold in general, as the following example shows:

Example 4. If $p = x$ and $P = \{p_1, p_2\} \subseteq \mathbb{Z}_{256}[y, x]$ where $p_1 = 2yx^2 + 2x^2 + 6yx + x$ and $p_2 = 4y + 4$ then p is irreducible by P, yet $p = (154yx + 206y + 154x + 1)p_1 + (179yx^3 + 50yx^2 + 75yx + 179x^3 + 25x^2)p_2 \in \langle P\rangle$.

3.5 Gröbner Bases

Definition 2. *Let $I \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ be an ideal. A set $P \subseteq I$ is a Gröbner basis for I iff, for all $p \in I$, if $p \neq 0$ then p is reducible by some $q \in P$.*

If $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ is a Gröbner basis for the ideal $I \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ and $p \in I$ then $p \to_P^* 0$. Hence, Gröbner bases allow ideal membership to be tested by reduction. In order to compute Gröbner bases, an auxiliary definition is required:

Definition 3. *The S-polynoimal of $p_1, p_2 \in \mathbb{Z}_m[\boldsymbol{x}]$ is defined:*

$$\mathsf{spol}(p_1, p_2) = d_2 2^{k-k_1}\boldsymbol{x}^{\boldsymbol{\alpha}-\boldsymbol{\alpha_1}}p_1 - d_1 2^{k-k_2}\boldsymbol{x}^{\boldsymbol{\alpha}-\boldsymbol{\alpha_2}}p_2$$

where, if $p_i = 0$ then $k_i = \omega$, $d_i = 1$ and $\boldsymbol{\alpha_i} = \boldsymbol{0}$, else $2^{k_i}d_i$ is the rank decomposition of $\mathsf{lc}(p_i)$ and $\boldsymbol{x}^{\boldsymbol{\alpha_i}} = \mathsf{lm}(p_i)$, $k = \max(k_1, k_2)$ and $\boldsymbol{\alpha} = \max(\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2})$.

Example 5. Let $p_1 = 2yx^2 + 2x^2 + 6yx + x$ and $p_2 = 4y + 4$ in $\mathbb{Z}_{256}[y, x]$. Then, $\mathsf{spol}(p_1, p_2) = 2(2yx^2 + 2x^2 + 6yx + x) - x^2(4y + 4) = 12yx + 2x$ and $\mathsf{spol}(p_1, 0) = 128(2yx^2 + 2x^2 + 6yx + x) - yx^2(0) = 128x$.

The following theorem, adapted from [5], provides an effective criterion for detecting that a finite basis is a Gröbner basis:

Theorem 1. *Let $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ and suppose for all $p_1, p_2 \in P$, $\mathsf{spol}(p_1, p_2) \to_P^* 0$ and $\mathsf{spol}(p_1, 0) \to_P^* 0$. Then, P is a Gröbner basis for $\langle P\rangle$.*

Note that if $|P| = \ell$ then this criterion takes $\binom{\ell}{2} + \ell$ reductions to verify.

```
function buchberger(F)
begin
    G := F; S := {{g1, g2} | g1 ∈ G, g2 ∈ G ∪ {0}, g1 ≠ g2}
    while (S ≠ ∅)
        {g1, g2} := element(S)
        S := S − {g1, g2}
        r := reduce(spol(g1, g2), G)
        if (r ≠ 0)
            G := G ∪ {r}
            S := S ∪ {{r, g} | g ∈ G ∪ {0}}
        end if
    end while
    return G
end
```

Fig. 2. Buchberger's algorithm over integers modulo 2^ω

3.6 Buchberger's Algorithm

Theorem 1 also yields a strategy for converting a finite basis $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ for an ideal $I \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ to a Gröbner basis for I. The strategy works by reducing each S-polynomial of the basis in turn; if some S-polynomial does not reduce to 0 then its reduced form is added to the basis and the procedure continues. Eventually, all S-polynomials of basis elements reduce to 0, at which point the algorithm returns. The algorithm determined by this strategy is called Buchberger's algorithm. Fig. 2 presents a version of Buchberger's algorithm, adapted from [5], which utilises a set S to record the set of S-polynomials that have yet to be reduced. It requires an auxiliary function reduce which realises reduction. More specifically, if $p \in \mathbb{Z}_m[\boldsymbol{x}]$ and $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ is finite then $p \to_P^* \mathsf{reduce}(p, P)$ and $\mathsf{reduce}(p, P) \not\to_P$.

Example 6. The table gives a trace of Buchberger's algorithm on $P = \{p_1, p_2\} \subseteq \mathbb{Z}_{256}[x, y]$ where $p_1 = 2yx^2 + 2x^2 + 6yx + x$ and $p_2 = 4y + 4$. Step k displays the values of G and S, and the next reduction, after k iterations of the main loop.

step	G	S	*reduction*
0	$\{p_1, p_2\}$	$\{\{p_2, 0\}, \{p_1, p_2\}, \{p_1, 0\}\}$	$0 \to_G 0$
1	$\{p_1, p_2\}$	$\{\{p_1, p_2\}, \{p_1, 0\}\}$	$12yx + 2x \to_G 246x = p_3$
2	$\{p_1, p_2, p_3\}$	$\{\{p_3, 0\}, \{p_2, p_3\}, \{p_1, p_3\}, \{p_1, 0\}\}$	$0 \to_G 0$
3	$\{p_1, p_2, p_3\}$	$\{\{p_2, p_3\}, \{p_1, p_3\}, \{p_1, 0\}\}$	$236x \to_G 0$
4	$\{p_1, p_2, p_3\}$	$\{\{p_1, p_3\}, \{p_1, p_0\}\}$	$226yx + 246x^2 + 123x \to_G 123x = p_4$
5	$\{p_1, p_2, p_3, p_4\}$	$\{\{p_4, 0\}, \{p_3, p_4\}, \{p_2, p_4\}, \{p_1, p_4\}, \{p_1, 0\}\}$	$0 \to_G 0$
⋮	⋮	⋮	⋮
9	$\{p_1, p_2, p_3, p_4\}$	$\{\{p_1, 0\}\}$	$128x \to_G 0$
10	$\{p_1, p_2, p_3, p_4\}$	$\emptyset$	–

```
function modifiedBuchberger(F)
begin
    G := ∅; A := F; S := ∅
    while (A ≠ ∅ ∨ S ≠ ∅)
        if (A ≠ ∅)
            p := element(A); A := A − {p}
        else
            {f1, f2} := element(S); S := S − {f1, f2}; p := spol(f1, f2)
        end if
        r := reduce(p, G)
        if (r ≠ 0)
            H := {g ∈ G | g ↛r}; G := H ∪ {r}; A := A ∪ (G − H)
            S := {{g1, g2} ∈ S | g1, g2 ∈ H ∪ {0}}) ∪ {(p, h) | h ∈ H ∪ {0}}
        end if
    end while
    return G
end
```

Fig. 3. Modified Buchberger's algorithm over integers modulo 2^ω

3.7 Modified Buchberger's Algorithm

Note that, for the Gröbner basis computed in Example 6, only p_2 and p_4 are necessary to ensure the Gröbner property, since any polynomial reducible by p_1 or p_3 must be reducible by p_4. This observation is formalised in the result:

Theorem 2. *Let $P \subseteq \mathbb{Z}_m[\boldsymbol{x}]$ and $P' = \{p \in P \mid p \not\to_{P-\{p\}}\}$. Suppose for all $p \in P - P'$ that $p \to^*_{P'} 0$, and for all $p_1, p_2 \in P'$, $\mathsf{spol}(p_1, p_2) \to^*_{P'} 0$ and $\mathsf{spol}(p_1, 0) \to^*_{P'} 0$. Then, P' is a Gröbner basis for $\langle P \rangle$.*

Proof. By Theorem 1, P' is a Gröbner basis for $\langle P' \rangle$. However, since $P' \subseteq P$ and $p \to^*_{P'} 0$ for all $p \in P - P'$ it follows that $\langle P \rangle = \langle P' \rangle$. The result follows.

If $|P| = \ell_1$ and $|P'| = \ell_2$ then this criterion takes $\binom{\ell_2}{2} + \ell_2 + (\ell_1 - \ell_2) = \binom{\ell_2}{2} + \ell_1$ reductions to verify, as opposed to $\binom{\ell_1}{2} + \ell_1$ reductions via the original criterion. Moreover, as for the original criterion, it yields an algorithm for converting a finite basis into a Gröbner basis. The algorithm operates as the original, except whenever a basis element becomes reducible by a newly added element, it is removed from, and then reduced by, the basis. If it reduces to 0 then it is discarded; otherwise, its reduced form is added to the basis. All S-polynomials derived from it are then discarded. Fig. 3 presents a modification to Buchberger's algorithm that implements this idea using a third set A to store elements that are removed from the basis. Elements of A are reduced in preference to elements of S, and the algorithm terminates when both A and S are empty. To handle the fact that some elements of the input basis could be reducible by other elements of the input basis, the set G is initialised to $\emptyset$ and the set A is initialised to P. Hence, the first iterations of the loop effectively add the input polynomials to the basis.

Example 7. The following table summarises a trace of the modified Buchberger algorithm on the same input as Example 6.

step	G	A	S	*reduction*
0	$\emptyset$	$\{p_1, p_2\}$	$\emptyset$	$p_1 \rightarrow_G p_1$
1	$\{p_1\}$	$\{p_2\}$	$\{\{p_1, 0\}\}$	$p_2 \rightarrow_G p_2$
2	$\{p_1, p_2\}$	$\emptyset$	$\{\{p_2, 0\}, \{p_1, p_2\}, \{p_1, 0\}\}$	$0 \rightarrow_G 0$
3	$\{p_1, p_2\}$	$\emptyset$	$\{\{p_1, p_2\}, \{p_1, 0\}\}$	$12yx + 2x \rightarrow_G 246x = p_3$
4	$\{p_2, p_3\}$	$\{p_1\}$	$\{\{p_3, 0\}, \{p_2, p_3\}\}$	$p_1 \rightarrow_G x = p_4$
5	$\{p_2, p_4\}$	$\{p_3\}$	$\{\{p_4, 0\}, \{p_2, p_4\}\}$	$p_3 \rightarrow_G 0$
6	$\{p_2, p_4\}$	$\emptyset$	$\{\{p_4, 0\}, \{p_2, p_4\}\}$	$0 \rightarrow_G 0$
7	$\{p_2, p_4\}$	$\emptyset$	$\{\{p_2, p_4\}\}$	$4x \rightarrow_G 0$
8	$\{p_2, p_4\}$	$\emptyset$	$\emptyset$	$-$

As noted above, the first two steps of the trace simply add the two input polynomials to the basis, which had already been performed in the original trace. Removing these steps yields a trace length of 6 compared to 10 in the original example. Moreover, by construction, neither p_2 nor p_4 is reducible by the other.

3.8 Encoding pseudo-boolean constraints

Figure 4 presents rules for translating a polynomial in the form $\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d$ to a propositional formula such that $\boldsymbol{c} \in \mathbb{Z}_m^\ell$, $d \in \mathbb{Z}_m$ and $\boldsymbol{X} \in \wp(\cup \boldsymbol{x})^\ell$, where $\boldsymbol{x}$, recall, is the vector of variables and ℓ is the arity of the vectors $\boldsymbol{c}$ and $\boldsymbol{X}$. This form of constraint, although restrictive, is sufficient to express the pseudo-booleans which arise in the final Gröbner basis, as illustrated below:

Example 8. Returning to PB_4 of Section 2 the polynomials $128b_4b_1 + 128b_1$ and $128b_3 + 128b_2 + 128$ can be written as $\langle 128, 128\rangle \cdot \langle \{b_1, b_4\}, \{b_1\}\rangle \equiv_{256} 0$ and $\langle 128, 128\rangle \cdot \langle \{b_3\}, \{b_2\}\rangle \equiv_{256} 128$ since $128 = -128 \pmod{256}$.

The rules of Figure 4 collectively reduce the problem of encoding a constraint to that of encoding one or more strictly simpler constraints. For brevity, we limit the commentary to the more subtle rules. The false rule handles constraints which are unsatisfiable because the coefficients $\boldsymbol{c}$ are all even and d is odd. The scale rule reduces the encoding problem to that for an equi-satisfiable constraint obtained by dividing the modulo, coefficients and constant by a power of 2. The set rule handles constraints where d is odd and there is a unique term c_iX_i for which c_i is odd. In this circumstance all the variables of X_i must be 1 for the constraint to be satisfiable. Conversely, clear deals with constraints for which d is even and there exists a unique c_iX_i for which c_i is odd since then one variable of X_i must be 0 for satisfiability. When none of above are applicable, split is applied to reduce to encoding problem to that of two strictly smaller constraints.

4 Experimental results

Our aim is to apply high-level algebraic reasoning to systematically reduce polynomials to compact systems of low-level constraints. Our experimental work thus

$$\text{true}\frac{}{\epsilon \cdot \epsilon \equiv_{2^r} 0 \;\rightarrow\; \mathsf{true}} \qquad \text{false}\frac{\forall c_i \in \boldsymbol{c}.\ \mathsf{rank}(c_i) > 0 \quad \mathsf{rank}(d) = 0}{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d \;\rightarrow\; \mathsf{false}}$$

$$\text{xor}\frac{}{\mathbf{1} \cdot \boldsymbol{X} \equiv_2 1 \;\rightarrow\; \bigoplus_{X \in \boldsymbol{X}} (\bigwedge X)} \qquad \text{iff}\frac{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_2 1 \;\rightarrow\; f}{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_2 0 \;\rightarrow\; \neg f}$$

$$\text{scale}\frac{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d \;\rightarrow\; f}{(2^s \boldsymbol{c}) \cdot \boldsymbol{X} \equiv_{2^{r+s}} (2^s d) \;\rightarrow\; f} \qquad \text{set}\frac{\mathsf{rank}(d) = 0 \quad \exists! c_i \in \boldsymbol{c}.\ \mathsf{rank}(c_i) = 0 \quad (\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d)[x \mapsto 1 \mid x \in X_i] \;\rightarrow\; f}{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d \;\rightarrow\; (\bigwedge X_i) \wedge f}$$

$$\text{clear}\frac{\mathsf{rank}(d) > 0 \quad \exists! c_i \in \boldsymbol{c}.\ \mathsf{rank}(c_i) = 0 \quad \forall x \in X_i.\ (\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d)[x \mapsto 0] \;\rightarrow\; f_x}{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d \;\rightarrow\; \bigvee_{x \in X_i} (\neg x \wedge f_x)}$$

$$\text{split}\frac{x \in \bigcup \boldsymbol{X} \quad (\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d)[x \mapsto 0] \;\rightarrow\; f_0 \quad (\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d)[x \to 1] \;\rightarrow\; f_1}{\boldsymbol{c} \cdot \boldsymbol{X} \equiv_{2^r} d \;\rightarrow\; (\neg x \wedge f_0) \vee (x \wedge f_1)}$$

Fig. 4. Reduction rules for pseudo-boolean polynomials modulo 2^r

assesses how the complexity of the low-level constraints relate to that of the input polynomials. Although we provide timings for our Buchberger algorithm, which as far as we know is state-of-the-art, this is not our main concern. Indeed, fast algorithms for calculating Gröbner bases over fields have emerged in the last two decades [11, 12], and similar performance gains seem achievable for modulo arithmetic. In light of this, our Buchberger algorithm is implemented in Scala 2.13.0 (compiled to JVM) using BigInt for complete generality. Experiments were performed on a 2.7GHz Intel i5 Macbook with 16Gbyte of SDRAM.

Datasets To exercise the symbolic method, polynomial systems were generated for different numbers of bit-vectors n and different bit-widths ω. For each $\omega \in \{2, 4, 8, 16, 32, 64\}$ and $n \in \{2, 3, 4\}$, 100 polynomial systems were constructed by randomly generating points in $\mathbb{Z}_{2^\omega}^n$ and deriving a system with these points as their zeros. First, each point was described as a system of n (linear) polynomials. Second, a single system was then formed with n points as its zeros through the introduction of $n-1$ fresh variables [2]. Third, the fresh variables were eliminated by calculating a Gröbner base to derive a basis constituting a single datapoint.

Symbolic variable and pseudo-boolean count Figure 5 presents box and whisker diagrams that summarise the numbers of symbolic variables and pseudo-booleans appearing in the derived pseudo-boolean systems. For each box and whisker, the lower and upper limits of the box indicate the first and third quartiles, the central line the median. The inter-quartile range (IQR) is the distance from the top to the bottom of the box. By convention the whiskers extend to 1.5 times the IQR above and below the median value; any point falling outside of this range is considered to be an outlier and is plotted as an individual point. Figure 5 was derived from datapoints generated from 6 random points. Similar trends

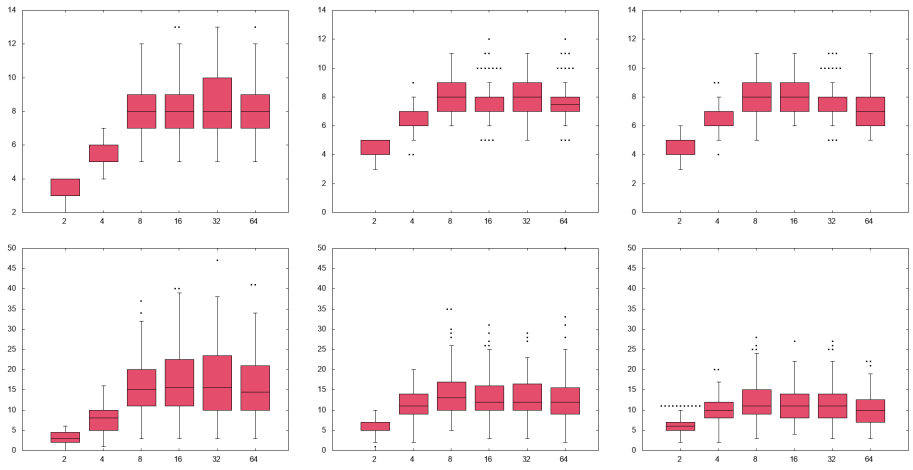

Fig. 5. Top row: number of symbolic variables (y) against ω (x) for $n = 2$, 3 and 4; Bottom row: number of pseudo-booleans (y) against ω (x) for $n = 2$, 3 and 4

are observed with fewer points and appear to be displayed for more points, but variable elimination impedes dataset generation and large-scale evaluation.

For both the number of symbolic variables and the number of pseudo-booleans, the medians level off after an initial increase and then appear to be relatively independent of ω. This surprising result suggests that algebraic methods have a role in reducing the complexity of polynomials for bit-vectors, which is sensitive to ω for bit-blasting. This implication is that the number of Gröbner base calculations also stablises with ω since this tallies with the number of symbolic variables. We also observe that the number of symbolic bits employed is typically only a fraction of the total number of bits occurring in the bit-vectors, hence setting a single symbolic bit is often sufficient to infer values for many other bits.

Pseudo-boolean versus multiplication count The upper row of figure 6 presents a fine-grained analysis of the number of pseudo-booleans, comparing this count to the number of bit-vector multiplications in the datasets. Multiplications are counted as follows: a monomial x^3yz, say, in a polynomial system contributes $2 + 1 + 1 = 4$ to its multiplication count, irrespective of whether it occurs singly or multiply in the system. The term $42x^3yz$ also contributes 4 to the count, so simple multiplications with constants are ignored. Addition is also not counted, the rationale being to compare the number of pseudo-booleans against the number of bit-vector multiplications, assuming that different occurrences of a monomial are detected and factored together. The x-axis of the histograms of Figure 6 divides the different ratios into bins, the first column giving the number of datasets for which the ratio falls within $[0, 0.25)$. As n increases the ratios bunch more tightly around the bin $[0.5, 0.75)$ and, more significantly, the

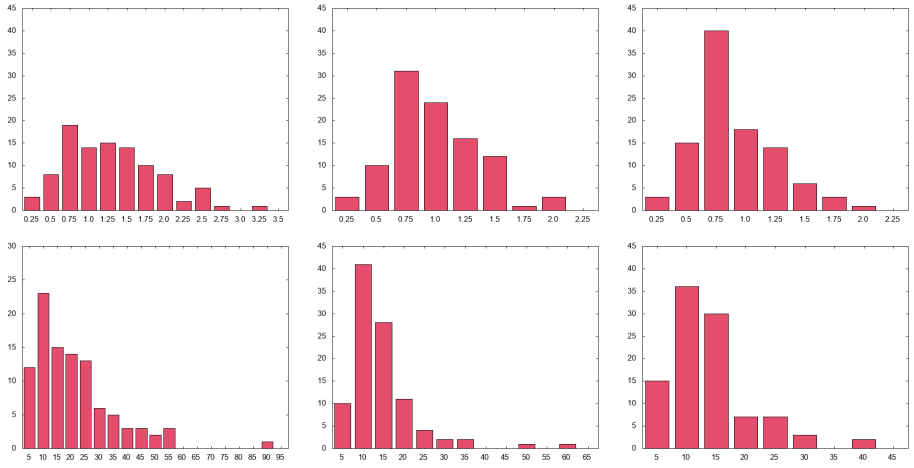

Fig. 6. Histograms for the ratio of the number of pseudo-booleans (top row)/logical connectives (bottom row) to the multiplication count for $n = 2$, 3, 4 and $\omega = 32$

number of pseudo-booleans rarely exceed twice the multiplication count, at least for $\omega = 32$.

Logical connectives versus multiplication count The lower row of Figure 6 examines the complexity of the resulting pseudo-boolean systems from another perspective: the number of logical connectives required to encode them. The pseudo-boolean systems were translated to propositional formulae using the reduction rules of Figure 4 and their complexity measured by counting the number of logical connectives used within them. The histograms present a frequency analysis of ratios of the number logical connectives to the multiplication count. Remarkably, histograms show that typically no more than 25 logical connectives are required per multiplication for $\omega = 32$.

Timing Although the number of symbolic variables is a proxy for complexity, it ignores that Gröbner basis computations increase in cost with the number of symbolic variables. Fig. 7 is intended to add clarity, plotting the time in seconds to calculate the pseudo-booleans against ω. As expected, the median runtimes increase with ω for any given n, though not alarmingly so for an implementation based on Buchberger rather than a modern, fast engine such as F5 [12]. It should be emphasised that the Gröbner basis computations are the dominating overhead: the resulting SAT instances are almost trivial for our datapoints. By way of an initial comparison, our approach is 23-fold slower on average than CVC4 [4] on our 64 bit problems though, no doubt, deploying F5 rather than Buchberger would significantly reduce this gap, as would recoding in C++.

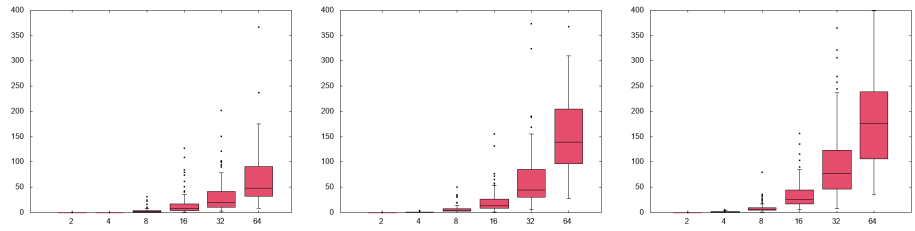

Fig. 7. Timings for Buchberger in seconds (y) against ω (x) for $n = 2, 3, 4$

5 Related Work

Momentum may be growing [1, 6, 10] for combining algebraic and SMT techniques but work at this intersection has mainly focused on CAD [16, 23]. Gröbner bases have been used [3], however, for interpolating non-linear constraints over bit-vectors by use of symbolic conversion predicates. These predicates are used to lazily convert between bit-vectors and rationals, over which Gröbner bases are computed. A closer integration of Gröbner bases and bit-vectors is offered by modifying Buchberger's algorithm [7] to handle modulo arithmetic [5], work which is developed in this paper. This approach has found application in verifying the equivalence of multiplication circuits [17], using signed and unsigned machine arithmetic. Further afield, but also motivated by the desire to bypass bit-blasting, efficient portfolio bit-vector solvers have been developed [25], combining learning with word-level propagators [20] that iteratively restrict the values that can be assigned to a bit-vector. In contrast to our work, the propagators are designed to run in constant time and make use of low-level bit-twiddling operations [26].

6 Concluding Discussion

This paper argues for translating polynomial equalities over bit-vectors into pseudo-boolean constraints, the central idea being to use Gröbner bases to expose the consequences of setting an individual bit on the bit-vectors over which a polynomial system is defined. The resulting technique, named bit-sequence propagation, typically infers the values of many bits from setting a single bit, even in the context of symbolic bit assignments. The symbolic bits enable the Gröbner bases to be calculated in a deterministic fashion, with search encapsulated within the pseudo-boolean solver, whether one is employed directly or a reduction to SAT is used. Furthermore, the technique extends to systems of mixed polynomial equalities and disequalities. Disequalities can be handled by expressing each disequality in terms of symbolic variables by rewriting the disequalities using the final basis derived for the equalities. Each disequality can then be converted to propositional formulae, as with an equality, and then a standard transform [24] used to ensure that the negation of each of these formulae holds.

References

1. Ábráham, E.: Building Bridges between Symbolic Computation and Satisfiability Checking. In: International Symposium on Symbolic and Algebraic Computation. pp. 1–6. ACM Press (2015)
2. Adams, W., Loustaunau, P.: An Introduction to Gröbner Bases. American Mathematical Society (1994)
3. Backeman, P., Rümmer, P., Zeljic, A.: Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. In: Formal Methods in Computer Aided Design. pp. 1–10. IEEE (2018)
4. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Computer-Aided Verification. Lecture Notes in Computer Science, vol. 6806, pp. 171–177 (2011)
5. Brickenstein, M., Dreyer, A., Greuel, G., Wedler, M., Wienand, O.: New Developments in the Theory of Gröbner Bases and Applications to Formal Verification. Journal of Pure and Applied Algebra **213**, 1612–1635 (2009)
6. Brown, C.: Bridging Two Communities to Solve Real Problems. In: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 11–14. IEEE Press (2016)
7. Buchberger, B.: Bruno Buchberger's PhD thesis 1965: An Algorithm for Finding the Basis Elements of the Residue Class ring of a Zero Dimensional Polynomial Ideal. Journal of Symbolic Computation **41**, 475–511 (2006)
8. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In: Automata Theory and Formal Languages. pp. 134–183. Springer (1975)
9. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties and Algorithms. Undergraduate Texts in Mathematics, Springer (1992)
10. Davenport, J.H., England, M., Griggio, A., Sturm, T., Tinelli, C.: Symbolic Computation and Satisfiability Checking. Journal of Symbolic Computation (2019), https://doi.org/10.1016/j.jsc.2019.07.017
11. Faugére, J.: A New Efficient Algorithm for Computing Gröbner Bases (F4). Journal of Pure and Applied Algebra **139**(1-3), 61–88 (1999)
12. Faugére, J.: A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5). In: International Symposium on Symbolic and Algebraic Computation. pp. 75–83 (2002)
13. Fekete, Y., Codish, M.: Simplifying Pseudo-Boolean Constraints in Residual Number Systems. In: SAT. Lecture Notes in Computer Science, vol. 8561, pp. 351–366. Springer (2014)
14. Griggio, A.: Effective Word-Level Interpolation for Software Verification. In: Formal Methods in Computer-Aided Design. pp. 28–36. IEEE (2011)
15. Horácek, J., Burchard, J., Becker, B., Kreuzer, M.: Integrating Algebraic and SAT Solvers. In: Mathematical Aspects of Computer and Information Sciences. vol. 10693, pp. 147–162. Springer (2017)
16. Jovanović, D., de Moura, L.: Solving Non-linear Arithmetic. In: International Joint Conference on Automated Reasoning. vol. 7364, pp. 339–354. Springer (2012)
17. Kaufmann, D., Biere, A., Kauers, M.: Verifying Large Multipliers by Combining SAT and Computer Algebra. In: Formal Methods in Computer-Aided Design. pp. 28–36 (2019)
18. Lang, S.: Graduate Texts in Mathematics: Algebra. Springer (2002)

19. Manquinhoand, V.M., Marques-Silva, J.: Using Cutting Planes in Pseudo-Boolean Optimization. Journal on Satisability, Boolean Modeling and Computation **2**, 199–208 (2006)
20. Michel, L., Van Hentenryck, P.: Constraint Satisfaction over Bit-vectors. In: Constraint Programming. Lecture Notes in Computer Science, vol. 7514, pp. 527–543. Springer (2012)
21. Müller-Olm, M., Seidl, H.: Analysis of Modular Arithmetic. In: European Symposium on Programming. Lecture Notes in Computer Science, vol. 3444, pp. 46–60. Springer (2005)
22. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of the ACM **53**(6), 937–977 (2006)
23. T. Viehmann and G. Kremer and E. Ábráham: Comparing Different Projection Operators in the Cylindrical Algebraic Decomposition for SMT Solving. In: International Workshop on Satisfiability Checking and Symbolic Computation (2017), http://www.sc-square.org/CSA/workshop2-papers/RP2-FinalVersion.pdf
24. Tseytin, G.S.: On the Complexity of Derivation in Propositional Calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic. pp. 115–125. Steklov Mathematical Institute (1970), translated from Russian: Zapiski Nauchnykh Seminarov LOMI 8 1968
25. Wang, W., Søndergaard, H., Stuckey, P.: Wombit: A Portfolio Bit-Vector Solver Using Word-Level Propagation. Journal of Automated Reasoning **63**(3), 723–762 (2019)
26. Warren, H.: Hacker's Delight. Addison-Wesley (2012)