

Analysing All Polynomial Equations in \mathbb{Z}_{2^w}

Helmut Seidl, Andrea Flexeder and Michael Petter

Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany,
{seidl, flexeder, petter}@cs.tum.edu,
WWW home page: <http://www2.cs.tum.edu/~{seidl, flexeder, petter}>

Abstract. In this paper, we present methods for checking and *inferring all* valid polynomial relations in \mathbb{Z}_{2^w} . In contrast to the infinite field \mathbb{Q} , \mathbb{Z}_{2^w} is finite and hence allows for finitely many polynomial functions only. In this paper we show, that checking the validity of a polynomial invariant over \mathbb{Z}_{2^w} is, though decidable, only *PSPACE*-complete. Apart from the impracticable algorithm for the theoretical upper bound, we present a feasible algorithm for verifying polynomial invariants over \mathbb{Z}_{2^w} which runs in polynomial time if the number of program variables is bounded by a constant. In this case, we also obtain a polynomial-time algorithm for inferring all polynomial relations. In general, our approach provides us with a feasible algorithm to infer all polynomial invariants up to a low degree.

1 Introduction

In reasoning about termination of programs, the crucial aspect is the knowledge about program invariants. Therefore, it is not surprising that the field of checking and finding of program invariants has been quite active, recently.

Many analyses interpret the values of variables regarding the field \mathbb{Q} . Modern computer architectures, on the other hand, provide arithmetic operations modulo suitable powers of 2. It is well-known that there are equalities valid modulo 2^w , which do not hold in general. The polynomial $2^{31}x(x+1)$, for example, constantly evaluates to 0 modulo 2^{32} but may show non-zero values over \mathbb{Q} . Accordingly, an analysis based on \mathbb{Q} will systematically miss a whole class of potential program invariants.

```
1  int b = ?;  
2  int c = 1 << 31, y = 0, x = 0;  
3  while (y-b != 0) {  
4      x = c*x*x + (c+1)*x + 1;  
5      y = x*x + y;  
6  }
```

Fig. 1. Computing the square power sum on 32bit machines

Example 1. As an example, consider the program from figure 1. This program repeatedly increases the value of program variable x in line 4 by 1 – if arithmetic is modulo 2^{32} . Therefore, the program `powersum()` computes a square sum. Thus, at program line 6 the polynomial invariant $2 \cdot x^3 + 3 \cdot x^2 + x - 6 \cdot y = 0$ holds modulo 2^{32} — but not over the field \mathbb{Q} .

An exact analysis of the example program should take into account the structure of polynomials over the domain $\mathbb{Z}_{2^{32}}$: The right hand side in the assignment in line 4 of the example can be rewritten as $2^{31}\mathbf{x}(\mathbf{x} + 1) + \mathbf{x} + 1$ where the first summand $2^{31}\mathbf{x}(\mathbf{x} + 1)$ is equivalent to the zero polynomial over $\mathbb{Z}_{2^{32}}$. Such polynomials are called *vanishing*. Singmaster [17] investigates the special structure of univariate vanishing polynomials over \mathbb{Z}_m and provides necessary and sufficient conditions for a polynomial to vanish over \mathbb{Z}_m . Hungerbühler and Specker extend this result to multivariate polynomials and introduce a *canonical form* for polynomials in quotient rings [3]. Shekhar et.al. present an algorithm to compute this canonical representation over the quotient ring \mathbb{Z}_{2^w} [16]. A minimal Gröbner base characterising all vanishing polynomials in arbitrary quotient rings is given by Wienand in [18]. In contrast to the infinite field \mathbb{Q} , the ring \mathbb{Z}_{2^w} is finite. Therefore, there are just finitely many distinct k -ary polynomial functions. In fact, it will turn out that we can restrict ourselves to polynomials in k variables up to a total degree $1.5(w + k)$. Due to this upper bound on the total degrees of the polynomials of interest, the problem of checking or inferring of polynomials over \mathbb{Z}_{2^w} becomes an analysis problem over finite domains only and therefore trivially is computable. Hence, the key issue is to provide tight upper complexity bounds as well as algorithms which also show decent behaviour on practical examples.

In this paper, we first consider the problem of checking whether a given polynomial relation is valid at a given program point. While being decidable over \mathbb{Q} , we show that this problem becomes *PSPACE*-complete over \mathbb{Z}_{2^w} . Furthermore, we present a practical algorithm for this problem which is based on effective precise weakest precondition computation. In case that the number of variables is bounded by a (small) constant, this algorithm even runs in polynomial time.

Secondly, we consider the problem of inferring all polynomial relations which are valid at a given program point. This problem, though not known to be computable in \mathbb{Q} , turns out to be computable in exponential time over \mathbb{Z}_{2^w} . Again, we present an algorithm for inferring all polynomial invariants of a given shape, whose runtime turns out to be polynomial given that the number of variables is bounded by a constant. Both algorithms have been implemented, and we report on preliminary experiments.

Related Work

The pioneer in the area of finding polynomial relations was Karr [4] who inferred the validity of polynomial relations of degree at most 1 (i.e., affine relations) over programs using affine assignments and tests only. An algorithm for checking validity of polynomial relations over programs using polynomial assignments is provided by Müller-Olm and Seidl [7] and was extended later to deal with disequality guards as well [9]. Their approach is based on effective weakest precondition computations where conjunctions of polynomial relations are described by *polynomial ideals*. Termination of a fixpoint computation in \mathbb{Q} thus is guaranteed by Hilbert's base theorem. In [9], the authors also observe that their method for checking the validity of polynomial relations can be used to construct an algorithm for inferring all polynomial invariants up to a fixed degree. In [13,14] Rodriguez-Carbonell et al. pick up the idea of describing invariants by polynomial ideals and propose a *forward* propagating analysis, based on a constraint system over these ideals. As infinite *descending* chains of polynomial ideals cannot be avoided in \mathbb{Q} when merging execution paths [8, Example 1], they provide special

cases or widening techniques to infer polynomial identities. Sankaranarayanan et al. also investigate polynomial invariants[15]. They propose to use *polynomial templates* to capture the effect of assignments in their analysis. These templates describe parametric polynomial properties. When determining the generic parameters via Gröbner bases, certain inductive invariants can be inferred. In contrast to the former approaches, Colon [2] provides an *interprocedural forward analysis* for polynomial programs. This analysis is based on ideals of polynomial *transition invariants*. In order to deal with infinite descending chains, Colon abstracts ideals with *pseudo-ideals*, which essentially are vector spaces of polynomials up to a given degree. The application of weakest precondition computations to interprocedural analysis of polynomial relations over \mathbb{Q} is discussed in [8]. An exact (even interprocedural) analysis of affine relations for programs using affine assignments over the domain \mathbb{Z}_m is provided in [10,11].

This paper is organised as follows. In Section 2 we specify the concrete semantics for the program class that is inspected by our analysis by means of control flow graphs. Section 3 gives a detailed description of the characteristics of polynomials in \mathbb{Z}_{2^w} . In Section 4, we first provide the complexity class for the general case of verifying polynomial invariants in \mathbb{Z}_{2^w} . We then specify our abstraction for the concrete semantics with the help of polynomial ideals. In Section 5 we present our specific concrete representation of polynomial ideals in \mathbb{Z}_{2^w} . We show how they contribute in the case of constantly many program variables to a better runtime complexity than the theoretical worst case in Section 4. We then illustrate in Section 6, how to extend this procedure to infer valid invariants up to a fixed degree and thus for inferring all valid relations. Section 7 finally summarises our results.

2 Fixpoint semantics

In this section we introduce the programs to be analysed together with the concrete semantics our polynomial analysis is based on. Basically, we emanate from the same concrete semantics as in [9].

The vector of variables $\mathbf{x} = (x_1, \dots, x_k)$ from the set of program variables $\mathbf{X} = \{x_1, \dots, x_k\}$ can take values in the ring \mathbb{Z}_{2^w} . A program state, which assigns values to variables, can be modelled by a k -dimensional vector $x = (x_1, \dots, x_k) \in \mathbb{Z}_{2^w}^k$, where x_i is the value assigned to variable x_i .

We assume that the basic statements in the considered program class are either *polynomial assignments* of the form $\mathbf{x}_j := p$ or *non-deterministic assignments* of the form $\mathbf{x}_j := ?$ where $\mathbf{x}_j \in \mathbf{X}$ or *polynomial disequality guards* of the form $p \neq 0$ where $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$. Recalling, that finding polynomial invariants in presence of equality guards turns out to be undecidable, we keep to non-deterministic branching instead. Non-deterministic assignments $\mathbf{x}_j := ?$ represent a safe abstraction of statements our analysis cannot handle precisely, e.g. non-polynomial expressions or user input.

Let Lab denote the set of basic statements and polynomial disequality guards. A *polynomial program* is given by a non-deterministic *control flow graph*, consisting of:

- *program points* N ,
- a set of edges $E \subseteq N \times N$,
- a mapping $A : E \rightarrow \text{Lab}$ from edges to statements or polynomial disequality guards
- a special *start point* $\text{st} \in N$.

The program executions reaching a given program point are characterised by a constraint system, for which our analysis provides a precise abstract interpretation. A program execution r (also called *run*) is a finite sequence $r \equiv r_1; \dots; r_m$ where each r_i is a basic statement or disequality guard. \mathbf{R} denotes the set of runs, which can be characterised as the smallest solution of a system of subset constraints on run sets \mathbf{R} , reaching the target program point t .

$$[\mathbf{R1}] \mathbf{R}(t) \supseteq \{\varepsilon\}$$

$$[\mathbf{R2}] \mathbf{R}(u) \supseteq f_e(\mathbf{R}(v)), \text{ if } e = (u, v) \in E$$

Constraint $[\mathbf{R1}]$ expresses, that the set of runs reaching program point t when starting from t contains the empty run, denoted by “ ε ”. By $[\mathbf{R2}]$, a run starting from u is obtained by considering an outgoing edge $e = (u, v)$ and concatenating a run corresponding to e with a run starting from v , where $f_e(R) = \{r; t \mid r \in \mathbf{R}(e) \wedge t \in R\}$. If edge e is annotated by $A(e) \equiv p \neq 0$ or $A(e) \equiv \mathbf{x}_j := p$, it gives rise to a single execution: $\mathbf{R}(e) = \{A(e)\}$. The effect of an edge e annotated by $\mathbf{x}_j := ?$ is captured by collecting *all constant* assignments:

$$\mathbf{R}(e) = \{\mathbf{x}_j := c \mid c \in \mathbb{Z}_{2^w}\}$$

Each run induces a *partial* transformation of the underlying program state $x \in \mathbb{Z}_{2^w}^k$. In the case of a disequality guard $p \neq 0$ this results in a partial identity function:

$$\text{dom}(\llbracket p \neq 0 \rrbracket) = \{x \in \mathbb{Z}_{2^w}^k \mid p(x) \neq 0\}$$

A polynomial assignment $\mathbf{x}_j := p$ causes the transformation with $\text{dom}(\llbracket \mathbf{x}_j := p \rrbracket) = \mathbb{Z}_{2^w}^k$ and

$$\llbracket \mathbf{x}_j := p \rrbracket x = (x_1, \dots, x_{j-1}, p(x), x_{j+1}, \dots, x_k)$$

Extending these definitions to runs, we obtain: $\llbracket \varepsilon \rrbracket = \text{Id}$, where Id is the identity function and $\llbracket r; r_{rest} \rrbracket = \llbracket r_{rest} \rrbracket \circ \llbracket r \rrbracket$ where “ \circ ” denotes composition of partial functions. The partial transformation $f = \llbracket r \rrbracket$ induced by a run r can always be represented by polynomials $q_0, \dots, q_k \in \mathbb{Z}_{2^w}[\mathbf{X}]$ such that $\text{dom}(f) = \{x \in \mathbb{Z}_{2^w}^k \mid q_0(x) \neq 0\}$ and $f(x) = (q_1(x), \dots, q_k(x))$ for every $x \in \text{dom}(f)$. For the identity transformation induced by the empty path ε the polynomials $1, \mathbf{x}_1, \dots, \mathbf{x}_k$ would hold. Transformations induced by polynomial assignments or guards can thus be represented in this manner and are closed under composition, similarly to [9].

3 The ring of polynomials in \mathbb{Z}_{2^w}

In order to develop an analysis inferring all polynomial relations modulo $m = 2^w$, we fix a bit width $w \geq 2$ of our numbers in the following. For $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, let $\mathbb{Z}_{2^w}[\mathbf{X}]$ denote the ring of all polynomials with coefficients in \mathbb{Z}_{2^w} . Each polynomial p can be written as a sum of terms, i.e., have the form $p = \sum_{\delta} c_{\delta} \cdot \mathbf{x}_1^{\delta_1} \dots \mathbf{x}_k^{\delta_k}$, with its degree $\delta \in \mathbb{N}^k$, $c \in \mathbb{Z}_{2^w}$ and $\mathbf{x} \in \mathbf{X}$. We call $\mathbf{x}_1^{\delta_1} \dots \mathbf{x}_k^{\delta_k}$ a monomial of total degree $\sum \delta_i$ and c_{δ} its coefficient. We agree on the terms for each polynomial to be sorted lexicographically on the string of degrees in the variables from \mathbf{X} . Then each polynomial p has a head term (respectively head coefficient or head monomial), that leads the trailing terms.

Recall that the coefficient ring \mathbb{Z}_{2^w} is not a field. More precisely, only all odd elements are invertible while every even element is a *zero divisor*. Thus, e.g., $2 \cdot 2^{w-1} \equiv$

0 in \mathbb{Z}_{2^w} . Useful facts about this ring can be found in [10] or basic text books on commutative ring theory, as [5].

Similarly to the case of fields [10], the set of polynomials $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$ which evaluate to 0 for a given subset $X \subseteq \mathbb{Z}_{2^w}^k$, is closed under addition and multiplication with arbitrary polynomials. A non-empty subset I of a ring with this property is also called an *ideal*. Thus, our program analysis maintains for every program point an ideal of polynomials.

Recall that the ring \mathbb{Z}_{2^w} is a *principal ideal* ring meaning that every ideal $I \subseteq \mathbb{Z}_{2^w}$ can be represented as the set $I = \{z \cdot a \mid z \in \mathbb{Z}_{2^w}\}$ of all multiples of a single ring element a . Thus by Hilbert's basis theorem, every ideal $I \subseteq \mathbb{Z}_{2^w}[\mathbf{X}]$ can be represented as the set of all linear combinations of a finite set $G = \{g_1, \dots, g_n\} \subseteq \mathbb{Z}_{2^w}[\mathbf{X}]$, i.e., $I = \{p_1g_1 + \dots + p_n g_n \mid p_i \in \mathbb{Z}_{2^w}[\mathbf{X}]\}$. In this case, we also refer to G as the set of generators of I and denote this by $I = \langle G \rangle$.

Assume $p, p' \in G$ are polynomials which share the same monomial t in their headterm, i.e., are of the form: $p = a \cdot 2^e \cdot s \cdot t + p_{rest}$ and $p' = a' \cdot 2^{e'} \cdot t + p'_{rest}$ with $e \geq e'$, some monomial s and odd $a, a' \in \mathbb{Z}_{2^w}$. In this case, we say that p is *reducible* by p' . More generally, we call p *reducible* by a set R of polynomials if p is reducible w.r.t. some $p' \in R$. If p is reducible by the polynomial p' , p can be reduced to the polynomial $q = a' \cdot p - a \cdot 2^{e-e'} \cdot s \cdot p'$. If $q = 0$, p is a multiple of p' and thus redundant in every set G of generators containing p' , i.e., $\langle G \setminus \{p\} \rangle = \langle G \rangle$. If $q \neq 0$, we can replace the polynomial p in G with the (simpler) polynomial q , i.e., the set G generates the same ideal as the set $G' = (G \setminus \{p\}) \cup \{q\}$.

Starting from a set G of generators, we can successively apply reduction to eventually arrive at a *reduced* set \bar{G} generating the same ideal as G . Here, we call the set \bar{G} reduced iff no polynomial $p \in \bar{G}$ is reducible w.r.t. $\bar{G} \setminus \{p\}$.

Lemma 1. *Assume that $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$ and $G \subseteq \mathbb{Z}_{2^w}[\mathbf{X}]$ is a finite reduced set of polynomials. Then a reduced set $\bar{G} \subseteq \mathbb{Z}_{2^w}[\mathbf{X}]$ can be constructed with $\langle \{p\} \cup G \rangle = \langle \bar{G} \rangle$. The algorithm runs in time $\mathcal{O}(k \cdot r^2)$ if r is the number of different exponents of monomials occurring during reduction, and k the number of program variables.*

Proof. A single reduction of a polynomial by a set of reduced polynomials can be carried out by as many subtractions (each of cost k) as a polynomial has monomials. This number is bounded by the number of different exponents r occurring during the reduction. Adding p to G is carried out by reducing potentially $|G| \leq r$ many polynomials. \square

Here, the total degree of a monomial $\mathbf{x}_1^{r_1} \dots \mathbf{x}_k^{r_k}$ is $d = r_1 + \dots + r_k$, and the total degree of a polynomial is the total degree of its head monomial. Thus, the number r of possibly occurring different exponents of monomials is bounded by:

Proposition 1. *The number of different exponents of monomials is given by*

$$r \leq \binom{d+k}{k} \leq \min((d+1)^k, (k+1)^d)$$

Note that the upper complexity bound is a crude worst-case estimation only. The practical run-time might be much smaller if the occurring polynomials are short, i.e., contain only few monomials.

Now assume that the ideal I is generated from a set G of generators and p is a polynomial. If p can be reduced (perhaps in several steps) to the 0 polynomial by means of the polynomials in G , then $p \in I$. The reverse, however, is only true for particularly saturated sets of generators such as *Gröbner bases* [1].

In the case of polynomials over a field, the constant zero polynomial is the only polynomial which evaluates to 0 for all vectors $x \in \mathbb{Z}_{2^w}^k$. This is no longer the case for the polynomial ring $\mathbb{Z}_{2^w}[\mathbf{X}]$. Let $I_v \subseteq \mathbb{Z}_{2^w}[\mathbf{X}]$ denote the ideal of all polynomials p with $p(x) = 0$ for all $x \in \mathbb{Z}_{2^w}^k$. The elements of I_v are also called *vanishing* polynomials. Only recently, a precise characterisation of the ideal I_v has been provided by Hungerbühler and Specker [3], which is recalled briefly, here. The first observation is that whenever 2^e divides $r! = r(r-1) \dots 1$, then 2^e also divides $(x+r-1) \dots (x+1) \cdot x$ for all x . Let $\nu_2(y)$ denote the maximal exponent e such that 2^e divides y . Thus, e.g., $\nu_2(1!) = 0$, $\nu_2(2!) = \nu_2(3!) = 1$ and $\nu_2(2^s!) = 2^s - 1$ for all $s \geq 1$. In particular, $\nu_2(r!) \geq w$ for $r \geq w + \log(w) + 1$. Since $1.5w + 1 \geq w + \log(w) + 1$, this implies that $\nu_2(r!) \geq \frac{2}{3}r - 1$.

Now consider the polynomial $p_r(\mathbf{x}_i) = \mathbf{x}_i \cdot (\mathbf{x}_i + 1) \dots (\mathbf{x}_i + r - 1)$. Then $2^{\nu_2(r!)}$ divides the value $p_r(z)$ for every z . Thus we obtain the following family $G(k, w)$ of vanishing polynomials:

$$2^a \cdot p_{r_1}(\mathbf{x}_1) \cdot \dots \cdot p_{r_k}(\mathbf{x}_k)$$

where $a \geq 0$, and $a + \nu_2(r_1!) + \dots + \nu_2(r_k!) \geq w$.

Example 2. Take \mathbb{Z}_{2^2} as domain. Then $p = \mathbf{x}^4 + 2\mathbf{x}^3 + 3\mathbf{x}^2 + 2\mathbf{x} = \mathbf{x}(\mathbf{x} + 1)(\mathbf{x} + 2)(\mathbf{x} + 3) = p_4(\mathbf{x})$. Since $\nu_2(4!) = 3 \geq 2$, $p(\mathbf{x})$ is a vanishing polynomial. \square

Note that Wienand [18] proves that the set $G(k, w)$ is not only contained in I_v but that I_v is in fact generated by $G(k, w)$.

Two polynomials $p, p' \in \mathbb{Z}_{2^w}[\mathbf{X}]$ are *semantically equivalent* if they define the same function $\mathbb{Z}_{2^w}^k \rightarrow \mathbb{Z}_{2^w}$, i.e., if $p - p' \in I_v$. We observe that for every polynomial in $\mathbb{Z}_{2^w}[\mathbf{X}]$, we can effectively find a semantically equivalent polynomial of small total degree. We have:

Lemma 2. *Every polynomial $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$ in k variables is semantically equivalent to a polynomial $p' \in \mathbb{Z}_{2^w}[\mathbf{X}]$ of degree less than $1.5(w + k)$.*

Proof. Let p' denote a polynomial of minimal total degree r and minimal number of monomials of total degree r which is semantically equivalent to p . Assume for a contradiction that $r \geq 1.5(w + k)$ and t is a monomial in p' of maximal total degree. Then t can be written as $t = a \cdot \mathbf{x}_1^{r_1} \dots \mathbf{x}_k^{r_k}$ where w.l.o.g. all $r_i \geq 1$. Then $\nu_2(r_j!) \geq \frac{2}{3}r_j - 1$ for all j . Consequently, the sum of these values is at least

$$\sum_j \left(\frac{2}{3}r_j - 1 \right) = \frac{2}{3} \sum_j (r_j - 1.5) = \frac{2}{3}(1.5(w + k) - 1.5k) = w$$

Therefore, the polynomial $q = p_{r_1}(\mathbf{x}_1) \dots p_{r_k}(\mathbf{x}_k)$ is vanishing. Note that the polynomial q has exactly one monomial of maximal total degree. We conclude that $p'' = p' - a \cdot q$ is a polynomial which is still semantically equivalent to p . Moreover the total degree of p'' is not larger than the total degree of p' and if the respective total degrees are equal, then p'' has less monomials of maximal total degree – thus contradicting our assumption. \square

Example 3. Consider the polynomials $p = \mathbf{x}^4 + 3\mathbf{x}$ and $p' = 2\mathbf{x}^3 + \mathbf{x}^2 + \mathbf{x}$ over \mathbb{Z}_{2^2} . Subtracting p' from p results in $q = p - p' = \mathbf{x}^4 + 2\mathbf{x}^3 + 3\mathbf{x}^2 + 2\mathbf{x} \in I_v$. Thus, p and p' are equivalent. \square

In [16], Shekhar et al. prove that a polynomial p is vanishing iff p can be reduced to 0 by means of the polynomials in $G(k, w)$. In the worst case, this takes $\mathcal{O}((d+1)^k)$ reduction steps if d is the total degree of p . As each of these reductions involves $\mathcal{O}((d+1)^k)$ many different monomials in the worst case, checking a polynomial for vanishing by means of reduction costs $\mathcal{O}((d+1)^{2k})$. Here, we sketch an alternative method. It consists in evaluating the polynomial for a finite set of selected arguments. The latter technique is based on the following observation.

Lemma 3. *A polynomial $p \in \mathbb{Z}_{2^w}[\{\mathbf{x}\}]$ of degree d is semantically equivalent to the zero polynomial, i.e., $\forall x \in \mathbb{Z}_{2^w}. p(x) = 0$ iff $p(h) = 0$ for $h = 0, 1, \dots, d$.*

Proof. “ \Rightarrow ” is trivial.

“ \Leftarrow ” by induction on degree d :

case $d = 0$: If the degree is zero, the polynomial is just described by a constant function $p(\mathbf{x}) \equiv c$. This polynomial is only zero for any h , if the constant value c is zero. Therefore $p \equiv 0$ and the assertion follows.

case $d > 0$: Let $p(h) = 0$ for $h = 0, \dots, d$. We consider the polynomial q of degree $d - 1$ with $q(\mathbf{x}) = p(\mathbf{x} + 1) - p(\mathbf{x})$ that has $q(h') = 0$ at least for $h' = 0, \dots, d - 1$. By induction hypothesis, $q(x) = 0$ for all $x \in \mathbb{Z}_{2^w}$. Thus, $p(\mathbf{x})$ and $p(\mathbf{x} + 1)$ are semantically equivalent. Since $p(0) = 0$, then also $p(1) = 0$ and thus, by induction, $p(x) = 0$ for all $x \in \mathbb{Z}_{2^w}$, and the assertion follows. \square

Lemma 3 shows that an arbitrary polynomial p is vanishing iff it vanishes for suitably many argument vectors. Substituting in a polynomial p of degree d all k different variables by $d + 1$ values each indicates that $p \notin I_v$, if it does not evaluate to zero each time. Otherwise $p \in I_v$. As evaluating p can be done in $|p|$, we conclude:

Corollary 1. *Assume $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$ is a polynomial where each variable has a maximal degree in p bounded by d . Then $p \in I_v$ can be tested in time $\mathcal{O}((d+1)^k \cdot |p|)$. \square*

4 Verifying polynomial relations in \mathbb{Z}_{2^w}

Similarly to [9] we denote a *polynomial relation* over the vector space $\mathbb{Z}_{2^w}^k$ as an equation $p = 0$ for some $p \in \mathbb{Z}_{2^w}[\mathbf{X}]$, which is representable by p alone. The vector $y \in \mathbb{Z}_{2^w}^k$ satisfies the polynomial relation p iff $p(y) = 0$. The polynomial relation p is valid at a program point v iff p is satisfied by $\llbracket r \rrbracket x$ for every run r of the program from program start st to v and every vector $x \in \mathbb{Z}_{2^w}^k$. In [6], Rütting and Müller-Olm prove that deciding whether a polynomial relation over \mathbb{Q} is valid at a program point v of a polynomial program is at least *PSPACE*-hard. Their lower-bound construction is based on a reduction of the *language universality problem* of non-deterministic finite automata and uses only the values 0 and 1. Therefore, literally the same construction also shows that validity of a polynomial relation over \mathbb{Z}_{2^w} is also *PSPACE*-hard. Regarding an

upper bound, we construct a Turing machine which non-deterministically computes a counterexample for the validity of a polynomial relation p . This counterexample can be found by simulating the original program on vectors over \mathbb{Z}_{2^w} representing the program state. The representation of such a program state can be done in polynomial space in \mathbb{Z}_{2^w} . The Turing machine accepts if it reaches program point v with a state $x \in \mathbb{Z}_{2^w}^k$ which does not satisfy p . Thus, we have a *PSPACE*-algorithm for dis-proving the validity of polynomial relations. Since the complexity class *PSPACE* is closed under complementation, we obtain:

Theorem 1. *Checking validity of polynomial invariants over \mathbb{Z}_{2^w} is PSPACE-complete.*
□

This is bad news for a general algorithm for the verification of polynomial invariants over \mathbb{Z}_{2^w} . The theoretical algorithm providing the upper bound in theorem 1 is not suitable for practical application. Therefore, we subsequently present an algorithm which has reasonable runtime behaviour at least for meaningful examples. In particular, it has polynomial complexity — given that the number of program variables is bounded by a constant.

This algorithm is based on the effective computation of weakest preconditions. Following [9], we characterise the weakest precondition of the validity of a relation p_t at program point t by means of a constraint system on ideals of polynomials.

In order to construct this constraint system, we rely on the weakest precondition transformers $\llbracket s \rrbracket^\top$ for tests, single assignments, or non-deterministic assignments:

$$\begin{aligned} \llbracket p \neq 0 \rrbracket^\top q &= \{p \cdot q\} \\ \llbracket \mathbf{x}_j := p \rrbracket^\top q &= \{q[p/\mathbf{x}_j]\} \\ \llbracket \mathbf{x}_j := ? \rrbracket^\top q &= \{q[h/\mathbf{x}_j] \mid h = 0, \dots, d\} \end{aligned}$$

where d is the maximal degree of \mathbf{x}_j in q . The transformers for assignments and disequality tests are the same which, e.g., have been used in [9]. Only for non-deterministic assignments $\mathbf{x}_j := ?$, extra considerations are necessary. The treatment of non-deterministic assignments in [9] for \mathbb{Q} consists in collecting all coefficient polynomials p_i not containing \mathbf{x}_j in the sum $q = \sum_{i \geq 0} p_i \cdot \mathbf{x}_j^i$. This idea does no longer work over \mathbb{Z}_{2^w} . Consider, e.g., $p = 2^{31} \mathbf{x}_1^2 \mathbf{x}_2 + 2^{31} \mathbf{x}_1 \mathbf{x}_2$. Equating every \mathbf{x}_1 -coefficient with zero would lead to the polynomial $2^{31} \mathbf{x}_2 = 0$ — which is not the weakest precondition, as $p = 0$ is trivially valid. The correctness of the new definition of $\llbracket \mathbf{x}_j := ? \rrbracket^\top$ for \mathbb{Z}_{2^w} on the other hand, follows from lemma 3.

The weakest precondition transformers $\llbracket s \rrbracket^\top$ for polynomials can be extended to transformers of ideals. Assume that the ideal I is given through the set G of generators. Then:

$$\llbracket s \rrbracket^\top I = \langle \bigcup \{ \llbracket s \rrbracket^\top g \mid g \in G \} \rangle$$

Note that $\llbracket s \rrbracket^\top q$ is vanishing whenever q is already vanishing. Therefore,

$$\llbracket s \rrbracket^\top (I_v) \subseteq I_v$$

for all s . Using the extended transformers, we put up the constraint system $\mathbf{R}_{p_t}^\sharp$ to represent the precondition for the validity of a polynomial p_t at program point t :

$$\begin{aligned} [\mathbf{R1}]^\sharp \mathbf{R}_{p_t}^\sharp(t) &\supseteq \langle \{p_t\} \rangle \\ [\mathbf{R2}]^\sharp \mathbf{R}_{p_t}^\sharp(u) &\supseteq \llbracket s \rrbracket^\top (\mathbf{R}_{p_t}^\sharp(v)), \text{ if } e = (u, v) \in E \wedge A(e) \equiv s \end{aligned}$$

For all program points, we may safely assume that all vanishing polynomials are valid. Therefore, we may consider the given constraint system over ideals I subsuming I_v , i.e., with $I_v \subseteq I$. This implies that we only consider ideals I where $p \in I$ whenever $p' \in I$ for every polynomial p' which is semantically equivalent to p . Note that the set of ideals subsuming I_v (ordered by the subset relation ‘ \subseteq ’) forms a complete lattice. Since all transformers $\llbracket s \rrbracket^\top$ are monotonic, this system has a unique least solution. Since all transformers $\llbracket s \rrbracket^\top$ transform I_v into (subsets of) I_v and distribute over sums of ideals, the least solution of the constraint system precisely characterises the weakest preconditions for the validity of p_t at program point t in a similar way as in [9]. We have:

Lemma 4. *Assume that $\mathbf{R}_{p_t}^\sharp(u)$, with u a program point, denotes the least solution of the constraint system $\mathbf{R}_{p_t}^\sharp$. Then the polynomial relation $p_t \in \mathbb{Z}_{2^w}[\mathbf{X}]$ is valid at the target node t iff $\mathbf{R}_{p_t}^\sharp(\text{st}) \subseteq I_v$. \square*

5 Computing with Ideals over $\mathbb{Z}_{2^w}[\mathbf{X}]$

In order to check the validity of the polynomial relation p_t at program point t , we must find succinct representations for the ideals occurring during fixpoint iteration which allow us first, to decide when the fixpoint computation can be terminated and secondly, to decide whether the ideal for the program start consists of vanishing polynomials only.

The basic idea consists in representing ideals through finite sets G of generators. In order to keep the set G small, we explicitly collect only polynomials *not* in I_v . Thus, G represents the ideal $\langle G \rangle_v = \langle G \rangle \oplus I_v = \{g + g_0 \mid g \in \langle G \rangle, g_0 \in I_v\}$.

Keeping the representation of vanishing polynomials implicit is crucial, since the number of necessary vanishing polynomials in $G(k, w)$ is exponential in k . By lemma 2, only polynomials up to degree $1.5(w + k)$ need to be chosen. By successively applying reduction, we may assume that G is reduced and consists of polynomials which cannot be (further) reduced by polynomials in $G(k, w)$ only. Let us call such sets of generators *normal-reduced*. Then by the characterisation of [16], $\langle G \rangle_v \subseteq I_v$ iff $G = \emptyset$.

Example 4. Consider the polynomials $p = \mathbf{x}^5 + \mathbf{x}^4 + 2\mathbf{x}^2 + 3\mathbf{x}$ and $\bar{p} = 6\mathbf{x}^5 + 7\mathbf{x}^3 + 2\mathbf{x}$ over \mathbb{Z}_{2^3} . In order to build a normal-reduced set of generators G , $\langle G \rangle_v = \langle \{p, \bar{p}\} \rangle_v$, we begin with a first round, reducing p and \bar{p} with the vanishing polynomial $p_v = \mathbf{x}^4 + 2\mathbf{x}^3 + 3\mathbf{x}^2 + 2\mathbf{x}$: $p' = p - (\mathbf{x} - 1)p_v = 7\mathbf{x}^3 + 3\mathbf{x}^2 + 5\mathbf{x}$ and $\bar{p}' = \bar{p} + (2\mathbf{x} + 4)p_v = 5\mathbf{x}^3 + 2\mathbf{x}$. Next, p' can be reduced by \bar{p}' , leading to $p'' = p' + 5\bar{p}' = 3\mathbf{x}^2 + 7\mathbf{x}$. \bar{p}' and p'' are nonreducible with respect to each other and I_v . Then $\langle \{p'', \bar{p}'\} \rangle_v = \langle \{p, \bar{p}\} \rangle_v$ where the set $\{p'', \bar{p}'\}$ is normal-reduced. \square

Concerning the computation of the fixpoint for $\mathbf{R}_{p_t}^\sharp$, consider an edge $e = (u, v)$ in the control-flow graph of the program labelled with $s = A(e)$. Each time when a new polynomial p is added to the ideal $\mathbf{R}_{p_t}^\sharp(v)$ associated with program point v which is not known to be contained in $\mathbf{R}_{p_t}^\sharp(v)$, all polynomials in $\llbracket s \rrbracket^\top p$ must be added to the ideal $\mathbf{R}_{p_t}^\sharp(u)$ at program u . The key issue for detecting termination of the fixpoint algorithm therefore is to check whether a polynomial p is contained in the ideal $\mathbf{R}_{p_t}^\sharp(u)$. Assume that the ideal $\mathbf{R}_{p_t}^\sharp(u)$ is represented by the normal-reduced set G of generators. Clearly, the polynomial p is contained in $\langle G \rangle_v = \mathbf{R}_{p_t}^\sharp(u)$ whenever p can be reduced by $G \cup G(k, w)$ to the 0 polynomial. The reverse, however, need not necessarily hold.

Exact ideal membership based on *Gröbner bases* requires to extend the set G with *S-polynomials* [1]. However, for generating all S-polynomials, virtually all pairs of generators must be taken into account. This applies also to the vanishing polynomials. The number of vanishing polynomials in $G(k, w)$ of degree $\mathcal{O}(w + k)$, however, is still exponential in k and also may comprise polynomials with many monomials. This implies that any algorithm based on exhaustive generation of S-polynomials cannot provide decent mean- or best case complexity at least in some useful cases. Therefore, we have abandoned the generation of S-polynomials altogether, and hence also exact testing of ideal membership.

Instead of ideals themselves, we therefore work with the complete lattice \mathbb{D} of normal-reduced subsets of polynomials in $\mathbb{Z}_{2^w}[\mathbf{X}]$. The ordering on the lattice \mathbb{D} is defined by $G_1 \sqsubseteq G_2$ iff every element $g \in G_1$ can be reduced to 0 w.r.t. $G_2 \cup G(k, w)$. The least element w.r.t. this ordering is \emptyset . Thus by definition, $G_1 \sqsubseteq G_2$ implies $\langle G_1 \rangle_v \subseteq \langle G_2 \rangle_v$, and $\langle G \rangle_v = I_v$ iff $G = \emptyset$. In order to guarantee the termination of the modified fixpoint computation, we rely on the following observation:

Lemma 5. *Consider a strictly increasing chain:*

$$\emptyset \sqsubset G_1 \sqsubset \dots \sqsubset G_h$$

of normal-reduced generator systems over $\mathbb{Z}_{2^w}[\mathbf{X}]$. Then the maximal length h of this chain is bounded by $w \cdot r$ where r is the number of head monomials occurring in any G_i .

Proof. For each G_i consider the set H_i , which denotes the set of terms $t = 2^s \cdot \mathbf{x}_1^{r_1} \dots \mathbf{x}_k^{r_k}$ for which $a \cdot t$ (a invertible) is the head term of a polynomial in G_i . Then for every i , H_i contains a term t which has not yet occurred in any $H_j, j < i$. The value h thus is bounded by the cardinality of $H_1 \cup \dots \cup H_h$, which in turn is bounded by $w \cdot r$. \square

In case that we are given an upper bound d for the total degree of polynomials in lemma 5, then by prop. 1, the height h is bounded by $h \leq w \cdot r \leq w \cdot (d + 1)^k$ and thus is exponential in k only.

The representation of ideals through normal-reduced sets of generators allows us to compute normal-reduced sets of generators for the least solution of the constraint system $\mathbf{R}_{p_t}^\sharp$. We obtain:

Theorem 2. *Checking the validity of a polynomial invariant in a polynomial program with N nodes and k variables over \mathbb{Z}_{2^w} can be performed in time $\mathcal{O}(N \cdot k \cdot w^2 \cdot r^3)$ where r is the number of monomials occurring during fixpoint iteration.*

Proof. Verification of a polynomial invariant p_t at a program point t is done via fixpoint iteration on sets of generators. Considering a set $G[u]$ of generators representing the ideal $\mathbf{R}_{p_t}^\sharp(u)$ of preconditions at program point u , we know from lemma 5, that an increasing chain of normal-reduced generator systems is bounded by $w \cdot r$. Each time, that the addition of a polynomial p leads to an increase of $G[u]$, the evaluation of $\llbracket s \rrbracket^\top(p)$ is triggered for each edge (u, v) labelled with s . Each precondition transformer creates only one precondition polynomial, except for the nondeterministic assignment. Essentially, each $\llbracket \mathbf{x}_j := ? \rrbracket^\top$ causes $d + 1$ (d the maximal degree of \mathbf{x}_j) polynomials to be added to the set of generators at the source u of the corresponding control flow edge. Since the degree d of any variable \mathbf{x}_j in an occurring generator polynomial is bounded by $1.5(w + 1)$, we conclude that the total number of increases for the set $G[u]$ of generators for program point u along the control flow edge (u, v) amounts to $\mathcal{O}(w^2 \cdot r)$. As we can estimate the complexity of a complete reduction by $\mathcal{O}(k \cdot r^2)$ with the help of lemma 1, we find that the amount of work induced by a single control flow edge therefore is bounded by $\mathcal{O}(k \cdot w^2 \cdot r^3)$. This provides us with the upper complexity bound stated in this theorem. \square

Assume that the maximal degree of a polynomial occurring in an assignment of the input program has degree 2. Then the maximal total degree d of any monomial occurring during fixpoint iteration is bounded by $1.5(w + k) + 3w + 2 = 4.5w + 1.5k + 2$. By prop. 1, the number r of monomials in the complexity estimation of theorem 2 is thus bounded by $(4.5w + k + 3)^k$. From that, we deduce that our algorithm runs in polynomial time – at least in case of constantly many variables. Of course, for three variables and $w = 2^5$, the number r of possibly occurring monomials is already beyond $2^{7 \cdot 4} = 2^{28}$ — which is far beyond what one might expect to be practical.

Name	Computation	verified invariant
power-1	$\mathbf{x}_1 = \sum_{k=0}^K 1$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad \mathbf{x}_1 = \mathbf{x}_2$
power-2	$\mathbf{x}_1 = \sum_{k=0}^K k$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad 2\mathbf{x}_1 = \mathbf{x}_2^2 + \mathbf{x}_2$
power-3	$\mathbf{x}_1 = \sum_{k=0}^K k^2$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad 6\mathbf{x}_1 = 2\mathbf{x}_2^3 + 3\mathbf{x}_2^2 + \mathbf{x}_2$
power-4	$\mathbf{x}_1 = \sum_{k=0}^K k^3$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad 4\mathbf{x}_1 = \mathbf{x}_2^4 + 2\mathbf{x}_2^3 + \mathbf{x}_2^2$
power-5	$\mathbf{x}_1 = \sum_{k=0}^K k^4$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad 30\mathbf{x}_1 = 6\mathbf{x}_2^5 - 15\mathbf{x}_2^4 - 10\mathbf{x}_2^3 + \mathbf{x}_2$
power-6	$\mathbf{x}_1 = \sum_{k=0}^K k^5$	$\mathbf{x}_2 = \sum_{k=0}^K 1 \quad 12\mathbf{x}_1 = 2\mathbf{x}_2^6 - 6\mathbf{x}_2^5 - 5\mathbf{x}_2^4 + \mathbf{x}_2^2$
geo-1	$\mathbf{x}_1 = (\mathbf{x}_3 - 1) \sum_{k=0}^K \mathbf{x}_3^k$	$\mathbf{x}_2 = \mathbf{x}_3^{K-1} \quad \mathbf{x}_1 = \mathbf{x}_2 + 1$
geo-2	$\mathbf{x}_1 = \sum_{k=0}^K \mathbf{x}_3^k$	$\mathbf{x}_2 = \mathbf{x}_3^{K-1} \quad \mathbf{x}_1 \cdot (\mathbf{x}_3 - 1) = \mathbf{x}_2 \mathbf{x}_3 - 1$
geo-3	$\mathbf{x}_1 = \sum_{k=0}^K \mathbf{x}_4 \cdot \mathbf{x}_3^k$	$\mathbf{x}_2 = \mathbf{x}_3^{K-1} \quad \mathbf{x}_1 \cdot (\mathbf{x}_3 - 1) = \mathbf{x}_4 \mathbf{x}_3 \mathbf{x}_2 - \mathbf{x}_4$

Table 1. Test programs and verified invariants in $w = 32$

We implemented our approach and evaluated it on selected benchmark programs, similar to the ones from [12]. We considered the series of programs `power- i` which compute sums of $(i - 1)$ -th powers, i.e., the value $\mathbf{x} = \sum_{\mathbf{y}} \mathbf{y}^{i-1}$. In the case $i = 6$, for example, the invariant $12\mathbf{x} - 2\mathbf{y}^6 - 6\mathbf{y}^5 - 5\mathbf{y}^4 + \mathbf{y}^2$ could be verified for the end point of the program. Additionally, we considered programs `geo- i` for computing variants of

the geometrical sum. An overview with verified invariants is shown in table 1. All these invariants could be verified instantly on a contemporary desktop computer with 2.4 GHz and 2GB of main memory.

6 Inferring polynomial relations over \mathbb{Z}_{2^w}

Still, no algorithm is known which, for a given polynomial program, infers all valid polynomial relations over \mathbb{Q} . In [9] it is shown, however, that at least all polynomial relations up to a *maximal total degree* can be computed. For the finite ring \mathbb{Z}_{2^w} , on the other hand, we know from lemma 2 that every polynomial has an equivalent polynomial of total degree at most $1.5(w + k)$. Therefore over \mathbb{Z}_{2^w} , any algorithm which computes all polynomial invariants up to a given total degree is sufficient to compute all valid polynomial invariants.

For a comparison, we remark that, since \mathbb{Z}_{2^w} is finite, the collecting semantics of a polynomial program of length N is finite and computable by ordinary fixpoint iteration in time $N \cdot 2^{\mathcal{O}(wk)}$. Given the set $X \subseteq \mathbb{Z}_{2^w}^k$ of states possibly reaching a program point v , we can determine all polynomials p of total degree at most $1.5(w + k)$ with $p(x) = 0$ for all $x \in X$ by solving an appropriate linear system of $|X| \leq 2^{wk}$ equations for the coefficients of p . For every program point u , this can be done in time $2^{\mathcal{O}(wk)}$. Here, our goal is to improve on this trivial (and intractable) upper bound. Our contribution is to remove the w in the exponent and to provide an algorithm whose runtime, though exponential in k in the worst case, may still be much faster on meaningful examples.

For constructing this algorithm, we are geared to the approach from [9]. This means that we fix a template for the form of polynomials that we want to infer. Such a template is given by a set M of monomials $m = \mathbf{x}_1^{r_1} \dots \mathbf{x}_k^{r_k}$ with $r_1 + \dots + r_k \leq 1.5(w + k)$. Note that for small maximal total degree d the cardinality of M is bounded by $(k + 1)^d$ while without restriction on d , the cardinality is bounded by an exponential in k (see prop. 1). Given the set M , we introduce a set $\mathbf{A}_M = \{\mathbf{a}_m \mid m \in M\}$ of auxiliary fresh variables \mathbf{a}_m for the coefficients of the monomials m in a possible invariant. The template polynomial p_M for M then is given by $p_M = \sum_{m \in M} \mathbf{a}_m \cdot m$.

Example 5. Consider the program variables \mathbf{x}_1 and \mathbf{x}_2 . Then the template polynomial for the set of all monomials of total degree at most 2 is given by: $\mathbf{a}_1 \mathbf{x}_1^2 + \mathbf{a}_2 \mathbf{x}_2^2 + \mathbf{a}_3 \mathbf{x}_1 \mathbf{x}_2 + \mathbf{a}_4 \mathbf{x}_1 + \mathbf{a}_5 \mathbf{x}_2 + \mathbf{a}_0$. \square

With the help of the verification algorithm from section 4, we can compute the weakest precondition for a given template polynomial p_m . Since during fixpoint computation, no substitutions of the generic parameters \mathbf{a}_m are involved, each polynomial p in any occurring set of generators is always of the form $p = \sum_{m \in M} \mathbf{a}_m \cdot q_m$ for polynomials $q_m \in \mathbb{Z}_{2^w}[\mathbf{X}]$. In particular, this holds for the set of generators computed by the fixpoint algorithm for the ideal at the start point st of the program. We have:

Lemma 6. *Assume that G is a set of generators of the ideal $\mathbf{R}_{pt}^\#(\text{st})$ for the template polynomial p_M at program point t . Then for any $a_m \in \mathbb{Z}_{2^w}$, $m \in M$, the polynomial $\sum_{m \in M} a_m m$ is valid at program point t iff for all $g \in G$, the polynomial $g[a_m/\mathbf{a}_m]_{m \in M}$ is a vanishing polynomial. \square*

It remains to determine the values $a_m, m \in M$ for which all polynomials g in a finite set G are vanishing. First assume that the polynomials in G may contain variables from \mathbf{X} . Assume w.l.o.g. that it is \mathbf{x}_k which occurs in some polynomial in G where the maximal degree of \mathbf{x}_k in polynomials of G is bounded by d . Then we construct a set G' by:

$$G' = \{g[j/\mathbf{x}_k] \mid g \in G, j = 0, \dots, d\}$$

The set G' consists of polynomials g' which contain variables from $\mathbf{X} \setminus \{\mathbf{x}_k\}$ only. Moreover by lemma 3, $g[a_m/\mathbf{a}_m]_{m \in M}$ is vanishing for all $g \in G$ iff $g'[a_m/\mathbf{a}_m]_{m \in M}$ is vanishing for all $g' \in G'$. Repeating this procedure, we successively may remove all variables from \mathbf{X} to eventually arrive at a set \bar{G} of polynomials without variables from \mathbf{X} . This means each $g \in \bar{G}$ is of the form $g = \sum_{m \in M} \mathbf{a}_m \cdot c_m$ for $c_m \in \mathbb{Z}_{2^w}$. Therefore, we can apply the methods from [11] for *linear* systems of equations over \mathbb{Z}_{2^w} (now with variables from \mathbf{A}_M) to determine a set of generators for the \mathbb{Z}_{2^w} -module of solutions. Thus, we obtain the following result:

Theorem 3. *Assume p is a polynomial program of length N with k variables over the ring \mathbb{Z}_{2^w} . Further assume that M is a subset of monomials of total degree bounded by $1.5(w+k)$. Then all valid polynomial invariants $\sum_{m \in M} c_m m$ with $c_m \in \mathbb{Z}_{2^w}$ can be computed in time $\mathcal{O}(N \cdot k \cdot w^2 (r_0 r)^3)$ where r_0 is the cardinality of M and r is the maximal number of monomials occurring during fixpoint iteration.*

Proof. Generator sets of polynomials over M and X are always composed of polynomials p of the form $p = \sum_{m \in M} \mathbf{a}_m \cdot \mathbf{x}_1^{d_1} \dots \mathbf{x}_k^{d_k}$. Thus, the number of occurring different monomials is bounded by $r_0 \cdot r$. Therefore, the maximal length of a strictly increasing chain of normal-reduced sets of generators is bounded by $w \cdot r_0 \cdot r$. As the number of monomials in a polynomial is bounded by $r_0 \cdot r$, the costs for updating a normal-reduced set of generators with a single polynomial is now $\mathcal{O}(k \cdot (r_0 \cdot r)^2)$. Again, we have to account $1.5(w+1)$ for the number of polynomials which can be produced by weakest precondition transformers in a single step. Altogether, we therefore have costs $\mathcal{O}(w \cdot w r_0 r \cdot k (r_0 r)^2)$ which are incurred at each of the control flow edges of the program to be analysed. \square

Finding all valid polynomial invariants means to compute the precondition for a template with all monomials up to degree $d = 1.5(w+k)$. We thus obtain $(1.5(w+k)+1)^k$ as an upper bound for the number r_0 of monomials to be considered in the postcondition. For input programs where the maximal degree of polynomials in assignments or disequalities is bounded by 2, the number r of occurring monomials can be bounded by $(4.5w+k+3)^k$. Summarising, we find that all polynomial invariants which are valid at a given program point can be inferred by an algorithm whose runtime is only exponential in k . This means that this algorithm is polynomial whenever the number of variables is bounded by a constant.

Example 6. Consider the program `geo-1` next to this paragraph which computes the geometrical sum. At program end, we obtain the invariant $x - y + 1 = 0$ as expected. Beyond that, we obtain the additional invariant $2^{31}y + 2^{31}x = 0$ which is valid over $\mathbb{Z}_{2^{32}}$ only. This invariant expresses that x and y are either both odd or both even at a specific program state. \square

```

1  int count = ?;
2  int x = 1, y = z;
3  while (count != 0){
4      count = count - 1;
5      x = x*z + 1;
6      y = z*y;
7  }
8  x = x*(z-1);

```

We used a prototypical implementation of the presented approach for conducting a test series whose results on our 2,4 GHz 2 GB machine are shown in table 2. The algorithm quickly terminates when inferring all invariants up to degree i for sums of powers of degree $i - 1$ for $i = 1, 2$ and 3 and also for the two variants of geometrical sums. Interestingly, it failed to terminate within reasonable time bounds for $i = 4$. In those cases when terminating, it inferred the invariants known from the analysis of polynomial relations over \mathbb{Q} — and quite a few extra non-trivial invariants which could not be inferred before. It remains a challenge for future work to improve on our methods so that also more complicated programs such as e.g. `power-4` can be analysed precisely.

Name	inferred polynomial	time	space
power-1	$x_0 - x_1$	0.065 sec	51 MB
power-2	$x_1^2 - 2x_0 + x_1$	0.195 sec	63 MB
power-3	$2x_1^3 - 3x_1^2 - x_1 - 6x_0,$ $2^{30}x_1^2 - 2^{31}x_0 + 2^{30}x_1,$ $3 \cdot 2^{29}x_0x_1 + 15 \cdot 2^{27}x_1^2 - 5 \cdot 2^{28}x_0 + 15 \cdot 2^{27}x_1,$ $3 \cdot 2^{28}x_0^2 - 25 \cdot 1^{26}x_0x_1 - 77 \cdot 2^{24}x_1^2 - 25 \cdot$ $2^{25}x_0 - 77 \cdot 2^{24}x_1,$ $21 \cdot 2^{24}x_1^3 + 191 \cdot 2^{23}x_1^2 + 65 \cdot 2^{24}x_0 + 149 \cdot 2^{23}x_1,$ $- 19 \cdot 2^{26}x_0^3 + 2^{24}x_0^2x_1 - 235 \cdot 2^{22}x_0x_1^2 - 191 \cdot$ $2^{23}x_0x_1 + 57 \cdot 2^{25}x_1^2 - 27 \cdot 2^{26}x_0 + 37 \cdot 2^{25}x_1$	1.115 sec	89 MB
power-4	n.a.	>24 h	> 1 GB
geo-1	$x_0 - x_1 - 1,$ $2^{31}x_1 + 2^{31}x_2$	0.064 sec	48 MB
geo-2	$2^{31}x_1x_2 + 2^{31}x_2, 2^{31}x_1 + 2^{31}x_2,$ $2^{28}x_1^2 + 230x_1x_2 - 7 \cdot 2^{28}x_2^2 - 3 \cdot 2^{29}x_2 + 2^{31},$ $x_0x_2 - x_1x_2 - x_0 + 1$	0.636 sec	65 MB
geo-3	23 polynomials ...	2.064 sec	96 MB

Table 2. Test programs and inferred invariants in $w = 32$

7 Conclusion

We have shown that verifying polynomial program invariants over \mathbb{Z}_{2^w} is *PSPACE*-complete. By that, we have provided a clarification of the complexity of another analysis problem in the taxonomy of [6]. Beyond the theoretical algorithm for the upper bound, we have provided a realistic method by means of normal-reduced generator sets. In case of constantly many variables, this algorithm runs in polynomial time. Indeed, our prototypical implementation was amazingly fast on all tested examples.

We extended the method for checking invariants to a method for inferring polynomial invariants of bounded degree — which in case of the ring \mathbb{Z}_{2^w} also allows to infer all polynomial invariants. Beyond the vanishing polynomials, the algorithm finds further invariants over \mathbb{Z}_{2^w} , which would not be valid over the field \mathbb{Q} and thus cannot be detected by any analyser over \mathbb{Q} . While still being polynomial for constantly many variables, our method turned out to be decently efficient only for small numbers of variables and low degree invariants. It remains for future work to improve on the method for inferring invariants in order to deal with larger numbers of variables and moderate degrees at least for certain meaningful examples.

References

1. T. Becker and V. Weispfenning. *Gröbner Bases – a computational approach to commutative algebra*. Springer Verlag, New York, 1993.
2. M. Colon. Polynomial approximations of the relational semantics of imperative programs. *Science of Computer Programming*, 64:76–96, Elsevier, 2007.
3. N. Hungerbühler and E. Specker. A generalization of the smarandache function to several variables. *Integers: Electronic Journal Combinatorial Number Theory*, 6, 2006.
4. M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.
5. H. Matsumura. *Commutative Ring Theory*. Cambridge University Press, 1989.
6. M. Müller-Olm and O. Rüdthing. On the complexity of constant propagation. In *10th European Symposium on Programming (ESOP)*, 2001.
7. M. Müller-Olm and H. Seidl. Polynomial Constants are Decidable. In *9th Static Analysis Symposium (SAS)*, pages 4–19. LNCS 2477, Springer-Verlag, 2002.
8. M. Müller-Olm, M. Petter, and H. Seidl. Interprocedurally analyzing polynomial identities. In *23rd Ann. Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 50–67, 2006.
9. M. Müller-Olm and H. Seidl. Computing Polynomial Program Invariants. *Information Processing Letters (IPL)*, 91(5):233–244, 2004.
10. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. In *14th European Symposium on Programming (ESOP)*, 2005.
11. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(5), 2007.
12. M. Petter. Berechnung von polynomiellen Invarianten. Master’s thesis, Technische Universität München, München, 2004.
13. E. Rodríguez-Carbonell and D. Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *International Symposium on Static Analysis (SAS)*, 2004.
14. E. Rodríguez-Carbonell and D. Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Science of Computer Programming*, 64:54–75, Elsevier, 2007.
15. S. Sankaranarayanan and Z. M. Henry. B. Sipma. Non-linear loop invariant generation using gröbner bases. In *ACM SIGPLAN Principles of Programming Languages (POPL)*, 2004.
16. N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan. Equivalence verification of polynomial datapaths with fixed-size bit-vectors using finite ring algebra. In *International Conference on Computer-Aided Design (ICCAD)*, pages 291–296, 2005.
17. D. Singmaster. On polynomial functions (mod m). *Journal of Number Theory*, 6:345–352, 1974.
18. O. Wienand. The Groebner basis of the ideal of vanishing polynomials. *to appear in Journal of Symbolic Computation (JSC)*, 2007. Preprint in arxiv math.AC/0801.1177.