JOURNAL OF ENGINEERING annals OF FACULTY OF ENGINEERING HUNEDOARA

FUTHER CONSIDERATION ON THE FORMAL VERIFICATION OF NUMBER THERETICAL ALGORITHMS

Laura KOVACS[1], Adalbert KOVACS[2]

[1] École Polytechnique Fédérale de Lausanne, SWITZERLAND
[2] University "Politehnica" Timisoara, ROMANIA

Abstract
We discuss experimental results of an automatic approach for inferring polynomial loop invariants of P-solvable loops implementing interesting number- theoretic algorithms. The method relies on techniques from symbolic summation and polynomial algebra, and is implemented in the software package Aligator written in Mathematica.
Keywords:
program verification, invariant generation, symbolic summation, Groebner basis.

1. INTRODUCTION

To verify and/or analyze programs containing loops one needs to discover some properties of loops automatically. Such properties are known as loop invariants. Powerful techniques for finding loop invariants are thus crucial for further progress of software verification and program analysis.

Research into methods for automatically generating loop invariants has a long history, starting with the works of [18, 5]. However, due to the limited arithmetic operations among program variables, the problem of invariant inference has recently become a challenging research topic [12, 16, 4, 14, 15, 9].

In [9,11] we introduced an automatic approach for polynomial invariant generation, that, contrarily to [12, 16, 4] does not use a priori fixed invariant templates, and it is applicable to the a richer class of loops than the one introduced in [15]. These invariants are of the form $p_1 = 0 \wedge \ldots \wedge p_r = 0,$ where $p_1, \ldots, p_r$ are polynomials over the program variables. In the sequel we will call a *polynomial equality* any equality of the form $p = 0,$ where p is a polynomial. Thus an invariant is polynomial if it is a conjunction of polynomial equalities.

The loops for which our method automatically infers polynomial invariants are characterized by the following conditions: (i) they contain only assignments to variables and conditional statements; (ii) tests conditions are omitted; (iii) the variables in assignments range over numeric types, such as integers or rationals; (iv) the variables can be expressed as a polynomial of the initial values of variables (those when the loop is entered), the loop counter, and some new variables, where there are algebraic dependencies among the new variables. We call such loops *P-solvable*. There are many natural examples of P-solvable loops in real-life programs.

Our method for invariant generation first translates conditional statements within the loop into a sequence of loops, called inner loops. Then all loop conditions are ignored, turning the loop into a non-deterministic program. Next symbolic summation methods are applied to the inner loops to determine if the output values of their variables can be expressed using symbolic expressions in their input values and inner loop counters. If yes, a collection of potential polynomial invariants is generated using a Groebner basis algorithm to eliminate loop counters. The invariant property of these polynomials are then checked using the weakest precondition strategy, and valid polynomial invariants of P-solvable loops are thus derived. Moreover, we proved that under some conditions the method computes *all* polynomial invariants, i.e. it computes a Groebner basis of the ideal of polynomial invariants. However, we could not find any example of a P-solvable loop for which our approach fails to be complete. We thus conjecture that the imposed completeness conditions cover a large class of imperative programs, and the completeness proof of our approach without the additional assumptions is a challenging task for further research.

Exploiting the symbolic manipulation capabilities of the computer algebra system Mathematica, our approach is implemented in a new software package called *Aligator* [9]. Using Aligator, a complete

set of polynomial invariants is successfully generated for numerous imperative programs working on numbers.

The purpose of this paper is to illustrate our approach on examples implementing interesting algorithms working on numbers. Finding invariants for such loops may be a very hard and creative work since non-trivial mathematical knowledge and intuition may be required. The ultimate goal of this paper is to emphasize the value of applying algebraic techniques for computer aided verification. We hope that the experimental results will provide practical justification for the relevance of our method. For technical details and correctness of the approach we refer to [9, 11].

2. P-SOLVABLE LOOPS AND INVARIANTS

In this section we first briefly recall fundamental facts from polynomial algebra and recurrence solving. Then our algorithm for P-solvable loops will be presented.

We assume that K is a field of characteristic zero (e.g. field of rationals, reals, etc.), and by $\overline{\mathrm{K}}$ we denote its algebraic closure. Throughout this paper, $X = \{x_1, \ldots, x_m\}$ $(m > 1)$ denotes the set of loop variables with initial values X_0, and $\mathrm{K}[X]$ is the ring of polynomials in the variables X with coefficients from K.

Polynomial Ideals and Invariants. As observed in [13], the set of polynomials p such that $p = 0$ is a polynomial invariant forms a polynomial ideal, called *polynomial invariant ideal*. The challenging part in polynomial invariant inference is thus to systematically compute a *basis* of this ideal. For doing so we rely on [1], and try to algorithmically compute a Groebner basis $\{p_1, \ldots, p_r\}$ $(p_i \in \mathrm{K}[X])$ of the polynomial invariant ideal. The conjunction of the polynomial equations corresponding to the polynomials from such a computed basis (i.e. $p_i(X) = 0$) would thus completely characterize the polynomial invariants of the loop. Namely, any other polynomial invariant could be derived as a logical consequence of $p_1 = 0 \wedge \ldots \wedge p_r = 0$.

In the process of deriving such a finite basis of the polynomial invariant ideal, we leverage methods from algorithmic combinatorics, as presented below.

Recurrences and Closed Forms. From the assignments statements of a P-solvable loop, recurrence equations of the variables are built and solved, using *the loop counter* n $(n \in \mathrm{N})$ as the recurrence index. Solutions of recurrence equations are called *closed-forms*, and they express the value of each program variable in a loop iteration as a function of n and some given initial values.

In our research, we handle special classes of recurrence equations, i.e. those that are either C-finite [19] or Gosper-summable [3]. C-finite recurrences always admit closed forms [2], whereas closed forms of Gosper-summable recurrences can be computed, if they exist, using the decision algorithm given by [3]. For example, the closed form solution of the C-finite recurrence $x[n+1] = 2 * x[n] + 1$ (corresponding to the loop assignment $x := 2 * x + 1$) is $x[n] = 2^n * x[0] + 2^n - 1$, where $x[n]$ denotes the value of variable x at loop iteration n and thus $x[0]$ is the initial value of x (i.e. before entering the loop); whereas the closed form the Gosper-summable recurrence $y[n+1] = y[n] + 2$ (corresponding to the loop assignment $y := y + 2$) is $y[n] = y[0] + 2 * n$, where $y[n]$ denotes the value of variable y at loop iteration n and $y[0]$ is the initial value of y.

We only consider P-solvable loops whose assignment statements describe Gosper-summable or C-finite recurrences. Moreover, P-solvability of loops requires the existence of *polynomial* closed form solutions of each variable, i.e. closed forms are polynomial expressions in loop counter, initial values and some new variables, where there are polynomial relations, so-called *algebraic dependencies*, among the new variables. Computing the algebraic dependencies reduces once again to compute a Groebner basis of the ideal of all algebraic dependencies [7]. For example, the ideal of algebraic dependencies among $a = 2^n$ and $b = 4^n$ is generated by the polynomial relation $a^2 - b = 0$; note that this relation holds for *any* $n \in \mathrm{N}$.

P-solvable Loops and Invariants. We consider P-solvable loops as below.

$$While[b_0,\ s_0;\ If[b_1\ Then\ s_1\ Else\ \ldots\ If[b_{k-1}\ Then\ s_{k-1}\ Else\ s_k]\ldots];\ s_{k+1}] \quad (1)$$

where $b_0,\ldots,b_{k-1}$ are boolean expressions, and $s_0,\ldots,s_{k+1}$ are sequences of assignments $(k \geq 1)$. As mentioned, in our approach to invariant generation all tests are omitted, and the loops are turned into non-deterministic programs [11]. Using regular expression like notation, loop (1) can be thus equivalently written as

$$\left(S_1|S_2|\ldots|S_k\right)^*, \text{ where } S_i = s_0;s_i;s_{k+1} \text{ for all } i=1,\ldots,k. \qquad (2)$$

Loop (1) is P-solvable iff the *inner* loops S_i^* from (2) are P-solvable. Namely, the variables of each S_i^* can be expressed as a polynomial of the initial values of variables (those when the loop is entered), the inner loop counter, and some new variables, where there are algebraic dependencies among the new variables. We write $S_i^{j_i}$ to mean the j_i-times repeated execution of S_i, where $j_i \in \mathrm{N}$ denotes the loop counter of S_i.

We have now all necessary ingredients to synthesize our invariant generation algorithm for P-solvable loops with assignments and (nested) conditionals. This is achieved in Algorithm 2.1, as given below.

Algorithm 2.1. P-solvable Loops with Nested Conditionals

Input: P-solvable loop (1) with k conditional branches and assignments

Output: Set GI of polynomial invariants for (1) among X with initial values X_0

Assumption: $k \geq 1,\ j_i \in \mathrm{N},\ i=1,\ldots,k$

1 Transform loop (1) into loop (2) with k P-solvable inner loops $S_1^*,\ldots,S_k^*$

2 **for** each $S_i^{j_i}, i=1,\ldots,k$ **do**

3 Compute the closed form system of the P-solvable loop $S_i^{j_i}$:

$$\begin{cases} x_1[j_i] &= q_{i,1}(j_i, y_{i1},\ldots,y_{is}) \\ \vdots & \\ x_m[j_i] &= q_{i,m}(j_i, y_{i1},\ldots,y_{is}) \end{cases}, \text{ where } \begin{array}{l} y_{ir} \in \overline{\mathrm{K}}, \\ q_{i,l} \in \overline{\mathrm{K}}[j_i, y_{i1},\ldots,y_{is}], \\ q_{i,l} \text{ are parametriz ed by } X_0 \\ r=1,\ldots,s, \quad l=1,\ldots,m \end{array}$$

4 Compute the ideal $A_i = I\left(j_i, y_{i1},\ldots,y_{is}\right)$ of algebraic dependencies for the new variables y_{is} from the closed form of $S_i^{j_i}$

5 **endfor**

6 Compute the *merged* closed form of $S_1^{j_1};\ldots;S_k^{j_k}$:

$$\begin{cases} x_1[j_1,\ldots,j_k] &= f_1(j_1, y_{11},\ldots,y_{1s},\ldots\ldots,j_k, y_{k1},\ldots,y_{ks}) \\ \vdots & \\ x_m[j_1,\ldots,j_k] &= f_m(j_1, y_{11},\ldots,y_{1s},\ldots\ldots,j_k, y_{k1},\ldots,y_{ks}) \end{cases}, \text{ with}$$

$f_l \in \overline{\mathrm{K}}[j_1, y_{11},\ldots,y_{1s},\ldots\ldots,j_k, y_{k1},\ldots,y_{ks}],$

the coefficients of f_l are given by the initial values X_0 before $S_1^{j_1};\ldots;S_k^{j_k}$

7 Let $A = \sum_{i=1}^{k} A_i$

8 Compute $PI_1 = \left(\langle x_1 - f_1,\ldots,x_m - f_m\rangle + A\right) \bigcap \mathrm{K}[x_1,\ldots,x_m]$

9 **for** each permutation $(w_1,\ldots,w_k) \neq (1,\ldots,k)$ over $\{1,\ldots,k\}$ **do**

10 Compute the *merged* closed form of $S_{w_1}^{j_{w_1}};\ldots;S_{w_k}^{j_{w_k}}$:

$$\begin{cases} x_1[j_1,\ldots,j_k] &= f_1(j_{w_1}, y_{w_1 1},\ldots,y_{w_1 s},\ldots\ldots,j_{w_k}, y_{w_k 1},\ldots,y_{w_k s}) \\ \vdots & \\ x_m[j_1,\ldots,j_k] &= f_m(j_{w_1}, y_{w_1 1},\ldots,y_{w_1 s},\ldots\ldots,j_{w_k}, y_{w_k 1},\ldots,y_{w_k s}) \end{cases}, \text{ with}$$

$f_l \in \overline{\mathrm{K}}[j_1, y_{11}, \ldots, y_{1s}, \ldots\ldots, j_k, y_{k1}, \ldots, y_{ks}],$

the coefficients of f_l are given by the initial values X_0 before $S_{w_1}^{j_{w_1}}; \ldots; S_{w_k}^{j_{w_k}}$

11 Let $A = \sum_{i=1}^{k} A_{w_i}$

12 Compute $G = \left(\langle x_1 - f_1, \ldots, x_m - f_m \rangle + A\right) \bigcap \mathrm{K}[x_1, \ldots, x_m]$

13 Compute $PI_1 = PI_1 \bigcap G$

14 **endfor**

15 From PI_1 keep the set GI of those polynomials whose comjunction is preserved by each $S_1, \ldots, S_k$:

$GI = \left\{ p \in PI_1 \,\middle|\, wp(S_i, p(X) = 0) \in \langle GI \rangle, i = 1, \ldots, k \right\} \subseteq PI_1,$ where

$wp(S_i, p(X) = 0)$ is the weakest precondition of S_i with postcondition $p(X) = 0$

16 *return* GI.

Algorithm 2.1 receives as input a P-solvable loop (1), and transforms it into (2) as its first step. Next, closed forms and algebraic dependencies of each inner loop $S_i^{j_i}$ are computed using the methods presented on page 2 (steps 2-5 of Algorithm 2.1). Further, inner loops are taken in all possible $k!$ permutation orders. For each permutation order, closed forms of inner loops are "merged" to express the behavior of a sequence of inner loops as a polynomial system in the inner loop counters, initial values X_0 and some new variables where there are algebraic dependencies among the new variables. Loop counters are next eliminated by Groebner basis computation, and the ideal of valid polynomial identities after arbitrary inner loop sequences is thus derived (steps 6-12 of Algorithm 2.1). Further, the intersection of the polynomial ideals of all inner loop sequences is computed (step 13 of Algorithm 2.1) and the ideal PI_1 of polynomial relations for the *first iteration* of loop (2) with initial values X_0 is obtained. Finally, using the weakest precondition strategy, the inductiveness property of polynomials from PI_1 is checked, and a set GI of polynomial invariants for (1) is derived (steps 15-16 of Algorithm 2.1).

Note that steps 9-16 are relevant only for P-solvable loops with nested conditionals (i.e. $k > 1$). For loops with assignment statements only, polynomial invariants are computed from the closed form system of the loop by eliminating the variables in the loop counter, using Groebner basis computation. Moreover, based on the theory of Groebner bases, the computed set of polynomials is a basis of the polynomial invariant ideal. Hence, our approach for invariant generation in case of P-solvable loops with assignments only is always complete: any further polynomial invariant can be inferred from the computed basis.

Moreover, under additional assumptions on the ideal of polynomial relations among X with initial values X_0 corresponding to sequences of k and $k+1$ inner loops, in [11] we proved that our method is also complete for a wide class of P-solvable loop with nested conditionals. Namely, it returns a basis for the polynomial invariant ideal for some special cases of P-solvable loops with conditional branches and assignments. It is worth to be mentioned though that these additional constraints cover a wide class of loops, and we could not find any example for which the completeness of our approach is violated.

3. EXPERIMENTAL RESULTS

We now turn our interest on the practical applicability of our method for invariant generation, and illustrate the technique on interesting examples implementing non-trivial number-theoretic algorithms.

Our approach is implemented in a new software package, called *Aligator* [10], written on top of the computer algebra system Mathematica. The package combines algorithms from symbolic summation and polynomial algebra with computational logic in a uniform framework, and is applicable to the rich class of P-solvable loops.

We begin with a loop without conditionals, and illustrate that Algorithm 2.1 is applicable for the special case when $k=1$ (i.e. loop with only assignments).

Example 3.1 [8] Consider the loop implementing an algorithm for computing the integer square root k of a given integer number a.

$$k := 0; j := 1; m := 1;$$
$$While[m \le a, \quad k := k+1; j := j+2; m := m+j].$$

By applying Algorithm 2.1, the polynomial invariants are derived as follows.

Step 1: Omitting test conditions.

$$S_1: \quad k := k+1; j := j+2; m := m+j.$$

Steps 2-5: Closed form computation.

System of recurrences:

$$\begin{cases} k[n+1] &= k[n]+1 \\ j[n+1] &= j[n]+2 \\ m[n+1] &= m[n]+j[n+1] \end{cases}$$

where $n \in \mathbb{N}$ is the loop counter

Recurrence solving:

$$\begin{cases} k[n] &\underset{Gosper}{=} k[0]+n \\ j[n] &\underset{Gosper}{=} j[0]+2n \\ m[n] &\underset{Gosper}{=} m[0]+j[0]*n+n(n+1) \end{cases}$$

with $A=\{\}$ (i.e. the empty set)

$k[0], j[0], m[0]$ denote respectively the initial values of k, j, m.

Steps 6-8: Variable elimination.

$$\begin{aligned} PI_1 &= \langle k-k[0]-n,\ j-j[0]-2n,\ m-m[0]-j[0]*n-n(n+1)\rangle + A \bigcap Z[k,j,m] \\ &= \langle 2j+j^2-4m-2j[0]-j[0]^2+4m[0],\ -j+2k+j[0]-2k[0]\rangle \end{aligned}$$

Steps 9-16: Polynomial invariant ideal.

$$GI = \{2j+j^2-4m-2j[0]-j[0]^2+4m[0],\ -j+2k+j[0]-2k[0]\} \quad \text{and} \quad PI_1 = \langle GI \rangle$$

In [11] we proved that affine loops are P-solvable. Note that Example 3.1 is affine, and thus P-solvable. Hence, the correctness of Algorithm 2.1 ensures that the returned set GI of invariants is a basis for the polynomial invariant ideal, and our method is complete: any further polynomial invariant can be derived from GI. The automatically inferred invariant for Example 3.1 is thus

$$2j+j^2-4m-2j[0]-j[0]^2+4m[0]=0 \wedge -j+2k+j[0]-2k[0]=0,$$

yielding $1+2j+j^2-4m=0 \wedge 1-j+2k=0,$ by initial values substitutions.

Many interesting algorithms implementing special numbers from algebraic combinatorics can be encoded using P-solvable loops, as also illustrated in the next example.

Example 3.2 [17, 6] Consider the loop computing the number of HC-polyominoes for $m \ge 2$.

$$g := 1; b := 5/16; a := 3/4; r := 2;$$
$$While[i \le m, \quad s := t; t := r; r := 5*r-7*a+4*b; a := t; b := s; g := 4*g; i := i+1]$$

We omit a detailed presentation of the steps of Algorithm 2.1, and discuss only the main results obtained from symbolic summation.

The loop body is described by the recurrence equations:

$$\begin{cases} r[n+3] &= 5*r[n+2]-7*r[n+1]+4*r[n] \\ a[n+3] &= r[n+2] \\ b[n+3] &= r[n+1] \\ g[n+3] &= 4*g[n+2] \\ i[n+3] &= i[n+2]+1 \end{cases}$$

where $n \ge 0$ is the loop counter. The recurrences of r and g are C-finite, and hence they can be solved by the methods presented on page 2. The recurrence of i is both Gosper-summable and C-finite, and thus it admits closed form solution which is computed in our approach by the Gosper-algorithm [3]. The closed forms of r and g yield algebraic exponential sequences in n, whose ideal of algebraic dependencies is derived as discussed on page 3. Further, by Groebner basis computation,

the variables in n are eliminated from the closed form system of the loop. Finally, after initial value substitution, the polynomial invariant ideal is generated by the polynomial relation:

$$496a^3+1104a^2b-896ab^2+256b^3-g+512a^2r-752abr+320b^2r-160r^2a+112r^2b+16r^3=0.$$

Finally, let us give an example of a P-solvable loop with nested conditionals.

Example 3.3 [20] Consider the imperative loop implementing an algorithm for computing the square root q with precision err for a real number a.

$$r := a-1; q := 1; p := 1/2;$$
$$While[2*p*r \geq err,$$
$$If[2*r-2*q*p \geq 0,$$
$$Then \quad r := 2*r-2*q-p; q := q+p; p := p/2$$
$$Else \quad r := 2*r; p := p/2]]$$

Step 1: Omitting test conditions and loop transformation.

$$S_1: \quad r := 2*r-2*q-p; q := q+p; p := p/2$$
$$S_2: \quad r := 2*r; p := p/2$$

Steps 2-5: Closed form computation of inner loops.
System of recurrences:

Inner loop $S_1^{j_1}$:

$j_1 \in \mathbb{N}$

$$\begin{cases} p[j_1+1] &= p[j_1]/2 \\ q[j_1+1] &= q[j_1]+p[j_1] \\ r[j_1+1] &= 2*r[j_1]-2*q[j_1]-p[j_1] \end{cases}$$

Inner loop $S_2^{j_2}$:

$j_2 \in \mathbb{N}$

$$\begin{cases} p[j_2+1] &= p[j_2]/2 \\ q[j_2+1] &= q[j_2] \\ r[j_2+1] &= 2*r[j_2] \end{cases}$$

where j_1 and j_2 represent the loop counters of the inner loops.

Recurrence solving:

Inner loop $S_1^{j_1}$:

$j_1 \in \mathbb{N}$

$$\begin{cases} p[j_1] &\underset{C-finite}{=} \frac{1}{2^{j_1}}*p[0_{j_1}] \\ q[j_1] &\underset{Gosper}{=} q[0_{j_1}]+2*p[0_{j_1}]-\frac{1}{2^{j_1-1}}*p[0_{j_1}] \\ r[j_1] &\underset{C-finite}{=} 2^{j_1}*\left(r[0_{j_1}]-2*q[0_{j_1}]-2*p[0_{j_1}]\right)- \\ & \quad \frac{1}{2^{j_1-1}}*p[0_{j_1}]+2*q[0_{j_1}]+4*p[0_{j_1}] \end{cases}$$

Inner loop $S_2^{j_2}$:

$j_2 \in \mathbb{N}$

$$\begin{cases} p[j_2] &\underset{C-finite}{=} \frac{1}{2^{j_2}}*p[0_{j_2}] \\ q[j_2] &= q[0_{j_2}] \\ r[j_2] &\underset{C-finite}{=} 2^{j_2}*r[0_{j_2}] \end{cases}$$

where $p[0_{j_1}], q[0_{j_1}], r[0_{j_1}]$ are the initial values of p,q,r before entering the inner loop $S_1^{j_1}$. Similarly, $p[0_{j_2}], q[0_{j_2}], r[0_{j_2}]$ are the initial values of p,q,r before entering the inner loop $S_2^{j_2}$.

Introducing new variables and computing algebraic dependencies:

Inner loop $S_1^{j_1}$:

$j_1 \in \mathbb{N}$

$x = 2^{j_1}, y = 2^{-j_1}$

$$\begin{cases} p[j_1] &= y*p[0_{j_1}] \\ q[j_1] &= q[0_{j_1}]+2*p[0_{j_1}]-2*y*p[0_{j_1}] \\ r[j_1] &= x*\left(r[0_{j_1}]-2*q[0_{j_1}]-2*p[0_{j_1}]\right)- \\ & \quad 2*y*p[0_{j_1}]+2*q[0_{j_1}]+4*p[0_{j_1}] \end{cases}$$

$A_1 = \langle x*y-1 \rangle$

Inner loop $S_2^{j_2}$:

$j_2 \in \mathbb{N}$

$u = 2^{j_2}, v = 2^{-j_2}$

$$\begin{cases} p[j_2] &= v*p[0_{j_2}] \\ q[j_2] &= q[0_{j_2}] \\ r[j_2] &= u*r[0_{j_2}] \end{cases}$$

$A_2 = \langle u*v-1 \rangle$

Steps 6-8: Polynomial relations of $S_1^{j_1}; S_2^{j_2}$.

Merging closed forms:

For the inner loop sequence $S_1^{j_1}; S_2^{j_2}$, the initial values $p[0_{j_2}], q[0_{j_2}], r[0_{j_2}]$ of the loop variables p, q, r before entering loop $S_2^{j_2}$ are given respectively by the values $p[j_1], q[j_1], r[j_1]$ after $S_1^{j_1}$. Thus, by replacing the closed form expressions of $p[j_1], q[j_1], r[j_1]$ in the closed forms $p[j_2], q[j_2], r[j_2]$, we get the closed form system for the values $p[j_1, j_2], q[j_1, j_2], r[j_1, j_2]$ of loop variables p, q, r after j_1-times repeated execution of S_1 followed by j_2-times repeated execution of S_2. Writing respectively p, q, r instead of $p[j_1, j_2], q[j_1, j_2], r[j_1, j_2]$, the obtained merged closed form is:

Closed form system of $S_1^{j_1}; S_2^{j_2}$:

$$\begin{cases} p &= v * y * p[0] \\ q &= q[0] + 2 * p[0] - 2 * y * p[0] \\ r &= u * \left(x * (r[0] - 2 * q[0] - 2 * p[0]) - 2 * y * p[0] + 2 * q[0] + 4 * p[0]\right) \end{cases}$$

with $A = \langle u * v - 1, x * y - 1 \rangle$

where $p[0], q[0], r[0]$ are respectively the initial values of the loop variables p, q, r before the first iteration of the P-solvable loop with nested conditionals (i.e. before $S_1^{j_1}; S_2^{j_2}$). We denote:

$$\begin{aligned} I_1 = \; & \langle p - v * y * p[0],\ q - \left(q[0] + 2 * p[0] - 2 * y * p[0]\right), \\ & r - u * \left(x * (r[0] - 2 * q[0] - 2 * p[0]) - 2 * y * p[0] + 2 * q[0] + 4 * p[0]\right), \\ & u * v - 1,\ x * y - 1 \rangle. \end{aligned}$$

Variable elimination:

$$PI_1 = I_1 \bigcap \mathrm{R[p,q,r]} = \langle q^2 - q[0]^2 + 2 * p * r - 2 * p[0] * r[0] \rangle.$$

Steps 9-14: Polynomial relations of $S_2^{j_2}; S_1^{j_1}$.

Merging closed forms:

Closed form system of $S_2^{j_2}; S_1^{j_1}$:

$$\begin{cases} p &= y * v * p[0] \\ q &= q[0] + 2 * p[0] - 2 * y * p[0] \\ r &= x * \left(u * r[0] - 2 * q[0] - 2 * v * p[0]\right) - 2 * y * v * p[0] + 2 * q[0] + 4 * v * p[0] \end{cases}$$

with $A = \langle u * v - 1, x * y - 1 \rangle$

where $p[0], q[0], r[0]$ are the initial values of the loop variables p, q, r before the first iteration of the P-solvable loop with nested conditionals (i.e. before $S_2^{j_2}; S_1^{j_1}$). We denote:

$$\begin{aligned} I_2 = \; & \langle p - v * y * p[0],\ q - \left(q[0] + 2 * p[0] - 2 * y * p[0]\right), \\ & r - \left(x * \left(u * r[0] - 2 * q[0] - 2 * v * p[0]\right) - 2 * y * v * p[0] + 2 * q[0] + 4 * v * p[0]\right) \\ & u * v - 1,\ x * y - 1 \rangle. \end{aligned}$$

Variable elimination:

$$G = I_2 \bigcap \mathrm{R[p,q,r]} = \langle q^2 - q[0]^2 + 2 * p * r - 2 * p[0] * r[0] \rangle.$$

Polynomial relations of inner loop sequences:

$$PI_1 = PI_1 \bigcap \mathrm{G} = \langle q^2 - q[0]^2 + 2 * p * r - 2 * p[0] * r[0] \rangle.$$

Steps 15-16: Polynomial invariants of P-solvable loop with conditionals.

$$GI = \{q^2 - q[0]^2 + 2 * p * r - 2 * p[0] * r[0]\},$$

and thus $PI_1 = \langle GI \rangle$. The derived set GI is hence a basis of the polynomial invariant ideal, and our method is complete: any further polynomial invariant can be derived from GI. Finally, by substituting the concrete values for the symbolically treated initial values $p[0], q[0], r[0]$, the automatically inferred polynomial invariant for Example 3.3 is thus:

$$a - 2 * p * r - q^2 = 0.$$

4. CONCLUSIONS

An automatic approach for polynomial invariant generation for P-solvable loops was discussed, and illustrated on a number of examples implementing non-trivial number-theoretic algorithms. For all examples we could find, and thus in particular for the examples discussed in the paper, a basis for the polynomial invariant ideal was derived by using recurrence solving, polynomial algebra and computational logic. The successful application of our approach underlines the value of using algebraic techniques for computer aided verification.

REFERENCES/BIBLIOGRAPHY

[1] B. Buchberger. An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. J. of Symbolic Computation, 41(3-4):475–511, 2006.

[2] G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. Recurrence Sequences, volume 104 of Mathematical Surveys and Monographs. American Mathematical Society, 2003.

[3] R. W. Gosper. Decision Procedures for Indefinite Hypergeometric Summation. J. of Symbolic Computation, 75:40–42, 1978.

[4] D. Kapur. A Quantifier Elimination based Heuristic for Automatically Generating Inductive Assertions for Programs. J. of Systems Science and Complexity, 19(3):307–330, 2006.

[5] M. Karr. Affine Relationships Amomg Variables of Programs. Acta Informatica, 6:133–151, 1976.

[6] M. Kauers. Algorithms for Nonlinear Higher Order Difference Equations. PhD thesis, RISC-Linz, Johannes Kepler University Linz, Austria, 2005.

[7] M. Kauers and B. Zimmermann. Computing the Algebraic Relations of C-finite Sequences and Multisequences. Technical Report 2006-24, SFB F013, 2006.

[8] D. E. Knuth. The Art of Computer Programming, volume 2. Seminumerical Algorithms. Addison-Wesley, 3rd edition, 1998.

[9] L. Kovács. Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema. PhD thesis, RISC, Johannes Kepler University Linz, 2007.

[10] L. Kovács. Aligator: A Mathematica Package for Invariant Generation. In Proc. of IJCAR, volume 5195 of LNCS, pages 275-282, Sydney, Australia, 2008.

[11] L. Kovács. Reasoning Algebraically About P-Solvable Loops. In Proc. of TACAS, volume 4963 of LNCS, pages 249–264, Budapest, Hungary, 2008.

[12] M. Müller-Olm and H. Seidl. Computing Polynomial Program Invariants. Indormation Processing Letters, 91(5):233–244, 2004.

[13] E. Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In Proc. of ISSAC 04, 2004.

[14] E. Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Invariants of Bounded Degree using Abstract Interpretation. Science of Computer Programming, 64(1), 2007.

[15] [E. Rodriguez-Carbonell and D. Kapur. Generating All Polynomial Invariants in Simple Loops. J. of Symbolic Computation, 42(4):443–476, 2007.

[16] S. Sankaranaryanan, H. B. Sipma, and Z. Manna. Non-Linear Loop Invariant Generation using Gr¨obner Bases. In Proc. of POPL 2004, 2004.

[17] R. P. Stanley. Enumerative Combinatorics, volume 2 of Cambridge Studies in Advanced Mathematics 62. Cambridge University Press, 1997.

[18] B.Wegbreit. The Synthesis of Loop Predicates. Communication of the ACM, 2(17):102–112, 1974.

[19] D. Zeilberger. A Holonomic System Approach to Special Functions. Journal of Computational and Applied Mathematics, 32:321–368, 1990.

[20] K. Zuse. The Computer - My Life. Springer, 1993