Invariant Generation for P-solvable Loops with Assignments

Laura Kovács *

EPFL, Swizterland
laura.kovacs@epfl.ch

Abstract. We discuss interesting properties of a general technique for inferring polynomial invariants for a subfamily of imperative loops, called the P-solvable loops, with assignments only. The approach combines algorithmic combinatorics, polynomial algebra and computational logic, and it is implemented in a new software package called `Aligator`. We present a collection of examples illustrating the power of the framework.

1 Introduction

Techniques for automatically checking and finding loop invariants and intermediate assertions have been studied and developed since the early works of [5, 9, 3]. Following the "trend" of developing powerful algorithms for automatically inferring polynomial invariants, in our work we apply advanced methods from symbolic summation [25, 17, 11] together with polynomial algebra algorithms [2, 24].

Based on the shape of the loop body and on the structure of assignment statements, in [13] we defined a certain family of loops, called *P-solvable*, for which all test conditions are ignored and the value of each program variable can be expressed as a polynomial of the initial values of variables, the loop counter, and some new variables where there are algebraic dependencies among the new variables. For such loops, we derived a systematic method for generating polynomial invariants. Our approach is implemented in a new software package `Aligator` [13], on top of the computer algebra system *Mathematica*. `Aligator` stands for **a**utomated **l**oop **i**nvariant **g**eneration by **a**lgebraic **t**echniques **o**ver the **r**ationals, and supports algebraic reasoning about loops. We successfully tried it on many programs working on numbers. Moreover, our approach to polynomial invariant generation is shown to be complete for some special cases. By completeness we mean that it generates a set of polynomials from which any polynomial invariant can be derived. For a detailed presentation of the soundness and completeness aspects of our approach we refer to [13, 14]. The automatically inferred valid polynomial relations, together with user-asserted non-polynomial invariants, can be used further in the verification of (partial) correctness of programs.

* Research was done while the author was at the Research Institute for Symbolic Computation (RISC), Linz, Austria, The work was supported by BMBWK (Austrian Ministry of Education, Science, and Culture), BMWA (Austrian Ministry of Economy and Work) and MEC (Romanian Ministry of Education and Research) in the frame of the e-Austria Timişoara project, and by FWF (Austrian National Science Foundation) - SFB project F1302.

In this paper, we focus our attention on P-solvable loops with *only* assignment statements. Several properties of such loops will be presented, followed by motivating examples. The case of affine loops is separately discussed. The examples from the paper highlight the power and limitations of our method. More examples and experimental results can be found in [13].

The current paper extends our conference paper [14] in two aspects.

- We give a detailed comparison with related work on generating polynomial invariants for loops with assignments only (see Sections 2 and 4).
- Most importantly, in Section 4 we present interesting properties of P-solvable loops with assignments only.

All theoretical results as well as comparison with related work are illustrated by examples.

The purpose of this paper is to demonstrate the applicability and the usefulness of several algebraic and combinatorial techniques for automatic generation of polynomial invariants, and, in more general terms, the power of combining techniques from computational logic with techniques from computer algebra in an imperative programming environment. We hope that the examples from the paper will justify this purpose.

The rest of the paper is structured as follows. Section 2 presents related work on polynomial invariant generation for loops with assignments only. Section 3 contains a short overview of our approach to invariant generation, and briefly introduces the reader to the algebraic consideration used throughout the paper. In Section 4 we present interesting properties of P-solvable loops, and demonstrate the results by a number of examples. Section 5 concludes the work.

2 Related Work

In [9], M. Karr proposed a general technique for finding affine relationships among program variables. However, Karr's work used quite complicated operations (transformations on invertible/non-invertible assignments, affine union of spaces) and had a limitation on arithmetical operations among the program variables. For these reasons, extension of his work has recently become a challenging research topic.

One line of work uses a *generic polynomial relation* of an a priori *fixed degree* [15, 16, 22, 8, 20]. The coefficients of this polynomial are replaced by variables, and constraints over the values of the coefficients are derived. The solution space of this constraint system characterizes the coefficients of all polynomial invariants up to the fixed degree. However, in case when the program has polynomial invariants of different degrees, these approaches have to be applied separately for the different degrees. This is not the case of our algorithm. Our restriction is not on the degree of sought polynomial relations, but on the type of assignments (recurrence equations) present in the loop body. The shape of assignments restricts our approach to the class of P-solvable loops, and thus we cannot handle loops with arbitrary polynomial assignments, nor tests in loop condition. However, for P-solvable loops with assignments only our method returns the generator set of all polynomial relations (of different degrees) among the program variables by the application of our method only once.

A different line of research imposes structural constraints on the assignment statements of the loop. Based on the theory of Gröbner basis, in [21] a fixpoint procedure for invariant generation is presented for so–called *simple* loops having *solvable mappings with positive rational eigenvalues*. This fixpoint is the ideal of polynomial invariants. The restriction of assignment mappings being solvable with positive rational eigenvalues ensures that the program variables can be polynomially expressed in terms of the loop counter and some auxiliary *rational* variables. Hence, the concept of solvable mapping is similar to the definition of P-solvable loop. However, contrarily to [21], in our approach we compute closed form solutions of program variables for a wider class of recurrence equations (assignment statements). The restriction on the closed form solution for P-solvable loops brings our approach also to the case of having closed forms as polynomials in the loop counters and additional new variables, but, unlike [21], the new variables can be arbitrary *algebraic* numbers, and not just rationals.

3 Overview

We begin with a brief overview of our approach to *invariant generation for P-solvable loops with assignments only*. We present the algebraic notions used later in the paper, and also fix some relevant notation. Further, a short description of our invariant generation algorithm will be given. More details can be found in [13, 14].

We assume that $\mathbb{K}$ is a field of characteristic zero (e.g. $\mathbb{Q}$, $\mathbb{R}$, etc.), and by $\bar{\mathbb{K}}$ we denote its algebraic closure. Throughout this paper we denote by $X = \{x_1, \ldots, x_m\}$ ($m > 1$) the set of loop variables, and by $n \in \mathbb{N}$ the loop counter.

P-solvable Loops with Assignments Only. As already mentioned, in our approach for generating polynomial invariants, test conditions in the loops are ignored. When we ignore conditions of loops, we will essentially deal with non-deterministic programs. Using regular-expression like notation, in [13] we introduced the syntax and semantics of the class of non-deterministic programs that we consider. We called this class *basic non-deterministic* programs. Essentially, when we omit the condition b from a conditional statement $\texttt{If}[b\ \texttt{Then}\ S_1\ \texttt{Else}\ S_2]$, where S_1 and S_2 are sequences of assignments, we will write it as $\texttt{If}[\ldots\ \texttt{Then}\ S_1\ \texttt{Else}\ S_2]$ and mean the basic non-deterministic program $S_1|S_2$. Similarly, we omit the condition b from a loop $\texttt{While}[b, S]$, where S is a sequence of assignments, and write it in the form $\texttt{While}[\ldots, S]$ to mean the basic non-deterministic program S^*. In our work, we developed a systematic method for generating (all) polynomial invariants for loops of such a non-deterministic syntax and semantics. Namely, we identified a class of loops, called the P-solvable loops, for which tests are ignored (as presented above), and the value of each loop variable is expressed as a polynomial in the initial values of variables (those when the loop is entered), the loop counter, and some new variables, where there are polynomial relations among the new variables. A precise definition of P-solvable loops can be found in [13].

Polynomial Ideals and Invariants. As observed in [19], the set of polynomial invariants forms a polynomial ideal, called *polynomial invariant ideal*. By Hilbert's basis theorem [1], any ideal, and in particular thus the ideal of polynomial invariants, has a finite basis. Using the Buchberger Algorithm [2], a Gröbner basis $\{p_1, \ldots, p_r\}$

($p_i \in \mathbb{K}[X]$) of the polynomial invariant ideal can be effectively computed. Hence, the conjunction of the polynomial equations corresponding to the polynomials from the computed basis (i.e. $p_i(X) = 0$) characterizes completely the invariants of the loop. Namely, any other invariant can be derived from the computed basis $\{p_1, \ldots, p_r\}$. In the process of deriving a basis for the polynomial invariant ideal, we rely on efficient methods from algorithmic combinatorics, as presented below.

Sequences and Recurrences. From the assignments statements of a P-solvable loop, recurrence equations of the variables are built and solved, using the loop counter n as the recurrence index.

In what follows, $f : \mathbb{N} \to \mathbb{K}$ defines a *(univariate) sequence* of values $f(n)$ from $\mathbb{K}$ ($n \in \mathbb{N}$). A *recurrence equation* for the sequence f is a rational function defining the values of $f(n+r)$ in terms of the previous values $f(n), f(n-1), \ldots, f(n+r-1)$, where $r \in \mathbb{N}$ is called the *order* of the recurrence. A solution of the recurrence equation $f(n)$, that is a *closed-form solution*, expresses the value of $f(n)$ as a function of the summation variable n and some given initial values, e.g. $f(0), \ldots, f(r-1)$. A detailed presentation of sequences and recurrences can be found in [4, 7]. In our research, we only consider special classes of recurrence equations, as follows.

A *C-finite recurrence* $f(n)$ is of the form $f(n+r) = a_0(n)f(n) + a_1(n)f(n+1) + \ldots + a_{r-1}(n)f(n+r-1)$, where the constants $a_0, \ldots, a_{r-1} \in \mathbb{K}$ do not depend on n. The closed form of a C-finite recurrence can always be computed [25, 4], and it is a linear combination of polynomials in n and algebraic exponential sequences $\theta^n \in \bar{\mathbb{K}}$, with $\theta \in \bar{\mathbb{K}}$, where there exist polynomial relations among the exponential sequences. Moreover, by adding any linear combination of polynomials and exponential sequences in n to the rhs of a C-finite recurrence, we obtain an *inhomogeneous linear recurrence with constant coefficients*. Such recurrences can be transformed into C-finite ones, and thus their closed forms can be always computed as well. In our work we rely on a *Mathematica* implementation given by the RISC Combinatorics group [10]. For example, the closed form of $f(n+1) = 4f(n) + 2$ is $f(n) = 4^n f(0) - \frac{2}{3}(4^n - 1)$, where $f(0)$ is the initial value of $f(n)$.

A *Gosper-summable* recurrence $f(n)$ is of the form $f(n+1) = f(n) + h(n)$, where the sequence $h(n)$ can be a product of rational function-terms, exponentials, factorials and binomials in the summation variable n (all these factors can be raised to an integer power). The closed form solution of a Gosper-summable recurrence can be exactly computed using the decision algorithm given by [6]. In our research, we use a *Mathematica* implementation of the Gosper-algorithm given by the RISC Combinatorics group [17]. For example, the closed form of $f(n+1) = f(n) + n^5$ is $f(n) = f(0) - \frac{1}{12}n^2 + \frac{5}{12}n^4 - \frac{1}{2}n^5 + \frac{1}{6}n^6$, where $f(0)$ is the initial value of $f(n)$.

In our work, we only consider P-solvable loops with the property that their assignment statements describe Gosper-summable or C-finite recurrences. Thus, the closed forms of loop variables can be computed as presented above.

Algebraic Dependencies. As mentioned already, the closed form solutions of the variables of a P-solvable loop are polynomial expressions in the summation variable n and algebraic exponential sequences in the summation variable n. We thus need to relate this sequences in a polynomial manner, such that the exponential sequences can be eliminated from the closed forms of the loop variables, and polynomial invariants

can be subsequently derived. In other words, we need to compute the *algebraic dependencies* among the exponential sequences $\theta_1^n, \ldots, \theta_s^n \in \bar{\mathbb{K}}$ of the algebraic numbers $\theta_1, \ldots, \theta_s \in \bar{\mathbb{K}}$ present in the closed forms. An *algebraic dependency* (or *algebraic relation*) of these sequences over $\bar{\mathbb{K}}$ is a polynomial $p \in \bar{\mathbb{K}}[y_1, \ldots, y_s]$ in s distinct variables $y_1, \ldots, y_s$, such that $p(\theta_1^n, \ldots, \theta_s^n) = 0, \forall n \in \mathbb{N}$. Computing the algebraic dependencies reduces thus to compute a Gröbner basis of the ideal of all algebraic dependencies. We integrated in our framework a *Mathematica* implementation for deriving such a basis [11]. For example, $\theta_1^n \theta_2^n - 1 = 0$ generates the ideal of algebraic dependencies among the exponential sequences of $\theta_1 = 4$ and $\theta_2 = \frac{1}{4}$, whereas there is no algebraic dependency among the exponential sequences of $\theta_1 = 4$ and $\theta_2 = 3$.

Invariant Generation for P-solvable Loops with Assignments Only. We have now all necessary ingredients to synthesize our invariant generation algorithm for P-solvable loops with assignments only. This is achieved in Algorithm 3.1. Algorithm 3.1 starts first with extracting and solving the recurrence equations of the P-solvable loop's variables X. Next, the ideal A of algebraic dependencies among the exponential sequences from the derived closed forms is computed. Finally, from the polynomial closed form system and A, the loop counter and the exponential sequences are eliminated, using Gröbner basis computation. The ideal G of polynomial invariants is thus derived.

Algorithm 3.1 P-solvable Loops with Assignments Only
Input: P-solvable loop with only assignment statements S
Output: The ideal $G \trianglelefteq \mathbb{K}[X]$ of polynomial invariants among X
Assumption: The recurrence equations of X are of order at least 1, $n \in \mathbb{N}$, X_0 are the initial values of X

1 Extract and solve the recurrence equations of the loop variables. We obtain:

$$\begin{cases} x_1[n] = \ q_1(n, \theta_1^n, \ldots, \theta_s^n) \\ \vdots \\ x_m[n] = q_m(n, \theta_1^n, \ldots, \theta_s^n) \end{cases}, \text{ where } \begin{array}{l} \theta_j \in \bar{\mathbb{K}},\ q_i \in \bar{\mathbb{K}}[n, \theta_1^n, \ldots, \theta_s^n], \\ q_i \text{ are parameterized by } X_0, \\ j = 1, \ldots, s,\ \ i = 1, \ldots, m \end{array}$$

2 Compute the ideal A of algebraic dependencies among $n, \theta_1^n, \ldots, \theta_s^n$.
3 Denote $z_0 = n, z_1 = \theta_1^n, \ldots, z_s = \theta_s^n$.
4 Consider $I = \langle x_1 - q_1, \ldots, x_m - q_m) \rangle + A \trianglelefteq \bar{\mathbb{K}}[z_0, z_1, \ldots, z_s, x_1, \ldots, x_m]$
5 **return** $G = I \cap \mathbb{K}[x_1, \ldots, x_m]$.

EXAMPLE 3.1 Consider the program fragment taken from [18], implementing an algorithm for computing the sum of the first n integers at power 5.

$$x := 0;\ y := 0;\ \texttt{While}[y \neq k,\ y := y + 1;\ x := x + y^5].$$

By applying Algorithm 3.1 and using `Aligator`, the polynomial invariants of the above loop are obtained as follows.

Step 1:
$$\begin{cases} y[n+1] = y[n] + 1 \\ x[n+1] = x[n] + y[n]^5 \end{cases} \Longrightarrow \begin{cases} y[n] \underset{Gosper}{=} y[0] + n \\ x[n] \underset{Gosper}{=} x[0] - \frac{1}{12}n^2 + \frac{5}{12}n^4 - \frac{1}{2}n^5 + \frac{1}{6}n^6 \end{cases}$$
where $y[0], x[0]$ denote the initial values of y, x before the loop.
Steps 2, 3: $z_0 = n, A = \langle z_0 - n \rangle$

Steps 4: The Gröbner basis computation with $z_0 \succ x \succ y$ yields:
$G = \langle 12\,x + y^2 - 5\,y^4 + 6\,y^5 - 2\,y^6 - 12\,x[0] - y[0]^2 + 5\,y[0]^4 - 6\,y[0]^5 + 2\,y[0]^6 \rangle$,
that is
$$G = \langle 12x + y^2 - 5y^4 + 6y^5 - 2y^6 \rangle,$$
by initial value substitutions.

4 P-solvable Loop Properties and Examples

In this section interesting properties of P-solvable loops will be presented, justified by motivating examples.

THEOREM 4.1 Given an imperative loop with assignment statements only.
If the closed form system of the recursively changed loop variables $\{x_1, \ldots, x_m\}$ is as in step 1 of Algorithm 3.1, with the property that there are no algebraic dependencies among the exponential sequences θ_i^n, then the loop has no polynomial invariant.

Proof. Consider a loop with closed form system as in step 1 of Algorithm 3.1, with the property that there are no algebraic dependencies among $\theta_1^n, \ldots, \theta_s^n$. Using notation from Algorithm 3.1, we thus have $A = \emptyset$.
Let's assume there is a polynomial relations $g \in \mathbb{K}[X]$ among $x_1, \ldots, x_m$, i.e. g is valid at any loop iteration n. Using the closed form solutions of X, we derive that $g \in A$ whenever evaluated at $(x_1, \ldots, x_m) = (q_1, \ldots, q_m)$, contradicting the assumption that $A = \emptyset$. ∎

EXAMPLE 4.2 Consider the loop `While`$[\ldots, x := 4x; y = 3y]$. The recurrence equations of the loop are C-finite, thus solvable, and their closed form solutions involve the exponential sequence 3^n and 4^n. As mentioned on page 4, there are no algebraic dependencies among these sequences. Hence, by Theorem 4.1, the loop has no polynomial invariant.

In the process of automatically inferring polynomial invariants for P-solvable loops, efficient techniques from symbolic summations are involved. Hence, Algorithm 3.1 can be applied on a rich class of practical imperative loops with assignments, implementing non-trivial algorithms working on numbers. From these assignments, the values of variables are expressed in terms of their previously computed values, and the loop is thus modeled by means of recurrence equations.

Moreover, using recurrence equations of order greater or equal to 2, we are able to handle imperative loops where auxiliary variables are used in order to refer to values of variables computed at finitely many loop iterations before (the number of previous loop iterations is given by the order of the recurrence). Such a loop behavior is presented below.

PROPOSITION 4.3 Given the P-solvable loop

$$\texttt{While}[\ldots,\ t := r;\ r := b * r + a * q;\ q := t;\ x := a * x],$$

where t, r, q, x are loop variables and $a, b \in \mathbb{K}$ are constants.
Then there is always a polynomial invariant $p \in \mathbb{K}[r, q, x]$ among r, q and x.

Proof. Denoting by $n \geq 0$ the loop counter, the given imperative loop can be expressed in terms of C-finite recurrence equations as follows.

$$\begin{cases} r[n+2] = b * r[n+1] + a * r[n] \\ q[n+2] = r[n+1] \\ x[n+2] = a * x[n+1]. \end{cases} \tag{1}$$

Note that the recurrence of r is of order 2. Therefore, in its closed form computation two initial values of r are needed (that are the initial values of r and q before entering the loop). Thus, the one-to-one correspondence between the loop counter and the recurrence index does not hold anymore, but, denoting by j the recurrence index, we have $j = n + 1$, where $j \geq 1$ and $n \geq 0$.
Solving (1) reduces to solving C-finite recurrences, and thus can always be solved. The derived closed forms are linear combinations of polynomials in j and algebraic exponential sequences, whose algebraic dependencies are obtained using the method discussed on page 4. Hence, (1) is P-solvable. Applying Algorithm 3.1, the obtained polynomial relation among the loop variables r, q, x is:

$$\begin{aligned} &-a^2\ x^2\ q[0]^4 - 2\ a\ b\ x^2\ q[0]^3\ r[0] + 2\ a\ x^2\ q[0]^2\ r[0]^2 - b^2\ x^2\ q[0]^2\ r[0]^2 + \\ &2\ b\ x^2\ q[0]\ r[0]^3 - x^2\ r[0]^4 + a^2\ q^4\ x[0]^2 + 2\ a\ b\ q^3\ r\ x[0]^2 - 2\ a\ q^2\ r^2\ x[0]^2 + \\ &b^2\ q^2\ r^2\ x[0]^2 - 2\ b\ q\ r^3\ x[0]^2 + r^4\ x[0]^2. \end{aligned}$$

■

Note that computing the above invariant requires computation of algebraic dependencies among arbitrary algebraic sequences, and thus not just rationals. Hence, [21] would fail in generating polynomial relations as invariants. Moreover, [15, 22] would need to guess first the degree of the polynomial, i.e. in this case 4, and then, fixing to 4 the degree of the generic polynomial invariant, a large number of constraints on the unknown coefficients of the polynomial would be generated. Proposition 4.3 highlights thus one of the advantages of combining algorithmic combinatorics and polynomial algebra for inferring invariant properties.

By substituting concrete values for the symbolic constants a and b, (1) generates a rich class of interesting examples. For example, taking $a = b = 1$ (and simplifying the loop body), we derive polynomial invariants for a loop implementing the computation of Fibonacci numbers, as presented below.

EXAMPLE 4.4 Fibonacci Numbers [7, 13].
Given the program fragment:

$$r := 1;\ q := 0;\ \texttt{While}[\ldots,\ t := r;\ r := r + q;\ q := t].$$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariant

$$q^4+2\,q^3\,r-q^2\,r^2-2\,q\,r^3+r^4-q[0]^4-2\,q[0]^3\,r[0]+q[0]^2\,r[0]^2+2\,q[0]\,r[0]^3-r[0]^4,$$

that is $-1 + q^4 + 2\,q^3 r - q^2 r^2 - 2\,q\,r^3 + r^4$, by initial values substitutions.

Similarly to Proposition 4.3, using recurrences of order $1, 2, 3, \ldots$, our work thus offers an algorithmic approach for inferring polynomial invariants of programs computing other special numbers, e.g. the Tribonacci numbers.

EXAMPLE 4.5 Tribonacci numbers [23, 13].
Given the program fragment:

$$r := 1;\ a := 1;\ b := 0;\ \texttt{While}[\ldots,\ s := t;\ t := r;\ r := r + a + b;\ a := t;\ b := s\ ].$$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariant

$$\begin{aligned}&2a^3 + 2a^2b + 2ab^2 + b^3 - 2abr + b^2r - 2ar^2 - br^2 + r^3 - 2a[0]^3 - 2a[0]^2b[0] -\\ &2a[0]b[0]^2 - b[0]^3 + 2a[0]b[0]r[0] - b[0]^2r[0] + 2a[0]r[0]^2 + b[0]r[0]^2 - r[0]^3,\end{aligned}$$

that is $-1 + 2a^3 + 2a^2b + 2ab^2 + b^3 - 2abr + b^2r - 2ar^2 - br^2 + r^3$, by initial values substitutions.

We finally focus our attention on a special class of loops, called *the affine loops*, defined as below.

DEFINITION 4.6 An imperative loop with only affine assignments is an *affine loop*. Considering $n \geq 0$ as the loop counter, the variables X of an affine loop satisfy the matrix equation

$$X[n+1] = A * X[n] + B, \tag{2}$$

with the matrix $A \in \mathbb{K}^{m\times m}$ and the (column) vector $B \in \mathbb{K}^m$.

In what follows, whenever we refer to an affine loop, we have in my mind an affine loop with *ignored test condition*.

Conform Section 3, affine assignments of variables define C-finite recurrences. Hence, any closed form solution of a loop variable defined by an affine relation among the loop variables is a linear combination of polynomials and algebraically related exponentials in the summation variable n. For such cases the P-solvable loop properties are hence fulfilled. We have the following theorem.

THEOREM 4.7 Affine loops are P-solvable. The ideal of polynomial invariants for an affine loop is algorithmically computable by Algorithm 3.1.

Unlike [15, 22], in the process of inferring automatically a basis for the polynomial invariant ideal of an affine loop, our method does not need to fix a priori the degree of sought polynomials. Moreover, due to powerful techniques from algorithmic combinatorics (C-finite solving and computing algebraic dependencies of exponential sequences), unlike [21] where the affine assignments have to be with positive rational eigenvalues, we do not impose any restriction on the shape of affine assignments. Based on Theorem 4.7, many interesting properties of affine loops can be thus automatically derived by symbolic summation algorithms.

EXAMPLE 4.8 Consider the affine loop given below, implementing an algorithm for computing the cubic root r for a given integer number a [12, 21].

$$x := a;\ r := 1;\ s := 13/4;$$
$$\texttt{While}[x - s > 0,\ x := x - s;\ s := s + 6 * r + 3;\ r := r + 1].$$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariants

$$-3r^2 + s + 3r[0]^2 - s[0],$$
$$r - 3r^2 + 2r^3 + 2x - r[0] + 3r[0]^2 - 6rr[0]^2 + 4r[0]^3 + 2rs[0] - 2r[0]s[0] - 2x[0],$$

yielding

$$-1 - 12r^2 + 4s, -1 - 4a + 3r - 6r^2 + 4r^3 + 4x,$$

by initial values substitutions.

5 Conclusions

In this paper, we present interesting properties of P-solvable loops with assignments only. These properties involve algebraic reasoning for automatically inferring polynomial invariants. Our approach is implemented in a new software package `Aligator`, in top of the computer algebra system *Mathematica*.

Our experiments show that many non-trivial algorithms working on numbers can be implemented using P-solvable loops. A collection of examples successfully worked out using the framework is presented in [13], some of them are given in this paper. For all of these examples a basis of polynomial invariants have been generated automatically and the generated invariants, together with additional non-polynomial properties asserted by the user, can be subsequently used for verifying properties of programs.

References

1. W. W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*. Graduate Studies in Math. 3. AMS, 1994.
2. B. Buchberger. An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006. Phd thesis 1965, University of Innsbruck, Austria.

3. P. Cousot and R. Cousot. Automatic Synthesis of Optimal Invariant Assertions: Mathematical Foundations. In *Proc. of the 1977 Symposium on Artificial Intelligence and Programming Languages*, pages 1–12, 1977.
4. G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. *Recurrence Sequences*, volume 104 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2003.
5. S. M. German and B. Wegbreit. A Synthesizer of Inductive Assertions. In *IEEE Transactions on Software Engineering*, volume 1, pages 68–75, 1975.
6. R. W. Gosper. Decision Procedures for Indefinite Hypergeometric Summation. *Journal of Symbolic Computation*, 75:40–42, 1978.
7. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley Publishing Company, 2nd edition, 1989.
8. D. Kapur. A Quantifier Elimination based Heuristic for Automatically Generating Inductive Assertions for Programs. *Journal of Systems Science and Complexity*, 19(3):307–330, 2006.
9. M. Karr. Affine Relationships Among Variables of Programs. *Acta Informatica*, 6:133–151, 1976.
10. M. Kauers. SumCracker: A Package for Manipulating Symbolic Sums and Related Objects. *Journal of Symbolic Computation*, 41:1039–1057, 2006.
11. M. Kauers and B. Zimmermann. Computing the Algebraic Relations of C-finite Sequences and Multisequences. Technical Report 2006-24, SFB F013, 2006.
12. D. E. Knuth. *The Art of Computer Programming*, volume 2. Seminumerical Algorithms. Addison-Wesley, 3rd edition, 1998.
13. L. Kovács. *Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema*. PhD thesis, RISC, Johannes Kepler University Linz, 2007.
14. L. Kovacs. Reasoning Algebraically About P-Solvable Loops. In *Proc. of TACAS*, Budapest, Hungary, 2008. To appear.
15. M. Müller-Olm and H. Seidl. Computing Polynomial Program Invariants. *Indormation Processing Letters*, 91(5):233–244, 2004.
16. M. Müller-Olm and H. Seidl. Precise Interprocedural Analysis through Linear Algebra. In *Proc. of the 31st POPL*, pages 330–341, 2004.
17. P. Paule and M. Schorn. A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities. *Journal of Symbolic Computation*, 20(5-6):673–698, 1995.
18. M. Petter. Berechnung von polynomiellen Invarianten. Master's thesis, Technical University Münich, Germany, 2004.
19. E. Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *Proc. of ISSAC 04*, 2004.
20. E. Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Invariants of Bounded Degree using Abstract Interpretation. *Science of Computer Programming*, 64(1), 2007.
21. E. Rodriguez-Carbonell and D. Kapur. Generating All Polynomial Invariants in Simple Loops. *J. of Symbolic Computation*, 42(4):443–476, 2007.
22. S. Sankaranaryanan, H. B. Sipma, and Z. Manna. Non-Linear Loop Invariant Generation using Gröbner Bases. In *Proc. of POPL 2004*, 2004.
23. R. P. Stanley. *Enumerative Combinatorics*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1997.
24. F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer, 1996.
25. D. Zeilberger. A Holonomic System Approach to Special Functions. *Journal of Computational and Applied Mathematics*, 32:321–368, 1990.