Automated Generation of Non-Linear Loop Invariants Utilizing Hypergeometric Sequences

Andreas Humenberger, Maximilian Jaroschek, Laura Kovács*
Technische Universität Wien
Institut für Informationssysteme 184
Favoritenstraße 9–11
Vienna A–1040, Austria
ahumenbe@forsyte.at
maximilian@mjaroschek.com
lkovacs@forsyte.at

ABSTRACT

Analyzing and reasoning about safety properties of software systems becomes an especially challenging task for programs with complex flow and, in particular, with loops or recursion. For such programs one needs additional information, for example in the form of loop invariants, expressing properties to hold at intermediate program points. In this paper we study program loops with non-trivial arithmetic, implementing addition and multiplication among numeric program variables. We present a new approach for automatically generating all polynomial invariants of a class of such programs. Our approach turns programs into linear ordinary recurrence equations and computes closed form solutions of these equations. These closed forms express the most precise inductive property, and hence invariant. We apply Gröbner basis computation to obtain a basis of the polynomial invariant ideal, yielding thus a finite representation of all polynomial invariants. Our work significantly extends the class of so-called P-solvable loops by handling multiplication with the loop counter variable. We implemented our method in the Mathematica package Aligator and showcase the practical use of our approach.

CCS CONCEPTS

•**Theory of computation** → **Invariants;** *Automated reasoning; Program verification;* •**Mathematics of computing** → *Discrete mathematics;*

KEYWORDS

program analysis, loop invariants, recurrence relations, hypergeometric sequences

*All authors are supported by the ERC Starting Grant 2014 SYMCAR 639270. We also acknowledge funding from the Wallenberg Academy Fellowship 2014 TheProSE, the Swedish VR grant GenPro D0497701, and the Austrian FWF research project RiSE S11409-N23.

ISSAC '17, Kaiserslautern, Germany

DOI: http://dx.doi.org/10.1145/3087604.3087623

ACM Reference format:
Andreas Humenberger, Maximilian Jaroschek, Laura Kovács. 2017. Automated Generation of Non-Linear Loop Invariants Utilizing Hypergeometric Sequences. In *Proceedings of ISSAC '17, Kaiserslautern, Germany, July 25-28, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3087604.3087623

1 INTRODUCTION

1.1 Overview

Analysis and verification of software systems requires non-trivial automation. Automatic generation of program properties describing safety and/or liveness is a key step to such automation, in particular in the presence of program loops (or recursion). For programs with loops one needs additional information, in the form of loop invariants or conditions on ranking functions.

In this paper we focus on loop invariant generation for programs with assignments implementing numeric computations over scalar variables. Our programming model extends the class of so-called P-solvable loops. Our work is based on and extends results of [8, 17], in particular it relies on the fact that the sets of polynomial invariants of P-solvable loops form polynomial ideals, and we employ reasoning about C-finite and hypergeometric sequences to determine algebraic dependencies. We show how to compute the ideals of polynomial invariants of extended P-solvable loops as follows: we model programs as a system of recurrence equations and compute closed form sequence solutions of these recurrences. If these sequences are of a certain type, which includes, among others, polynomials, rational functions, exponential and factorial sequences, then we compute a set of generators of the polynomial invariant ideal via Gröbner bases. We implemented our approach in the Mathematica package Aligator [9] that is able to compute polynomial loop invariants for programs that, to the best of our knowledge, no other approach is able to handle.

This paper is organized as follows. In Section 2, we state basic definitions and facts about the algebra of linear ordinary recurrence operators as well as C-finite and hypergeometric sequences. We also give a precise definition of the programming model we take into consideration, particularly the notion of imperative loops with assignment statements only. This is followed by a description of the class of P-solvable loops and its reach and limitations in Section 3. In Section 4 we present our main contribution, an extension of the class of P-solvable loops by reasoning about hypergeometric sequences and we derive the necessary theoretical and algorithmic results to offer fully automated polynomial invariant generation therein. We conclude the paper with a presentation of our implementation in the Mathematica package Aligator in Section 5 and a summary of possible future research directions in Section 6.

1.2 Related Work

Many classical data flow analysis problems, such as constant propagation and finding definite equalities among program variables, can be seen as problems about polynomial identities expressing loop invariants. In [10, 18] a method built upon linear and polynomial algebra is developed for computing polynomial equalities of a bounded degree. The work of [2] also uses an a priori fixed bound on the degree of polynomial invariants and employs SMT-based constraint solving for computing concrete values of the unknown coefficients in the polynomial template invariants. A related approach was proposed by [16] using abstract interpretation. Abstract interpretation is also used in [3, 4] for computing polynomial invariants of programs whose assignments can be described by C-finite recurrences. In our work we do not rely on narrowing/widening techniques from abstract interpretation and do not require a bound on the degree of generated polynomial invariants. Instead, we use algebraic reasoning about holonomic sequences to infer the set of all polynomial invariants. For program loops with assignments only, our technique can handle programs with more complex arithmetic than the previously mentioned methods. Our work is currently restricted though to single-path loops.

Without an a priori fixed polynomial degree, in [17] the polynomial invariant ideal is approximated by a fixed point procedure based on polynomial algebra and abstract interpretation. In [8], the author defines the notion of P-solvable loops which strictly generalizes the programming model of [17]. Given a P-solvable loop with assignments and nested conditionals, the results in [8] yield an automatic approach for computing all polynomial loop invariants. Our work extends [8, 17] in new ways: it handles a richer class of P-solvable loops where multiplication with the loop counter is allowed. Our technique relies on manipulating hypergeometric sequences and relaxes the algebraic restrictions of [8, 17] on program operations. To the best of our knowledge, no other method is able to derive polynomial invariants for extended P-solvable loops. Unlike [8, 17] however, we only treat loops with assignments; that is, invariants for extended P-solvable loops with conditionals are not yet contained in our approach.

Another related line of research on polynomial invariant generation is presented in [20], where data from concrete program executions is used to generate candidate invariants. Machine learning on candidate invariants is further applied to infer polynomial invariant properties. Unlike [20], our approach is based only on static analysis and goes beyond polynomial arithmetic by handling rational functions and operations on hypergeometric terms.

2 PRELIMINARIES

In this section we give a brief overview of the algebra of linear ordinary recurrence operators as well as C-finite and hypergeometric sequences which we will use further on. We also describe our programming model in detail.

2.1 Recurrence Operators and Holonomic Sequences

Let $\mathbb{K}$ be a computable field of characteristic zero.

The algebra of linear ordinary recurrence operators in one variable will serve as the algebraic foundation to deal with recurrence equations. For details on general Ore algebras, see [1, 11].

Definition 2.1. Let $\mathbb{K}(x)[S]$ be the set of univariate polynomials in the variable S over the set of rational functions $\mathbb{K}(x)$ in x and let $\sigma\colon \mathbb{K}(x) \to \mathbb{K}(x)$ be the forward shift operator in x, i.e. $\sigma(r(x)) = r(x+1)$ for $r(x) \in \mathbb{K}(x)$. The *Ore polynomial ring of ordinary recurrence operators* is defined as the ring $(\mathbb{K}(x)[S], +, \cdot)$ with component-wise addition and the unique distributive and associative extension of the multiplication rule

$$Sa = \sigma(a)S \qquad \text{for all } a \in \mathbb{K}(x),$$

to arbitrary polynomials in $\mathbb{K}(x)[S]$. To clearly distinguish this ring from the commutative polynomial ring over $\mathbb{K}(x)$, we denote it by $\mathbb{K}(x)[S;\sigma,0]$. The *order* of an operator $L \in \mathbb{K}(x)[S;\sigma,0]$ is its degree in S.

Without loss of generality, we assume that the leading coefficient of any given operator $L \in \mathbb{K}(x)[S;\sigma,0]$ is equal to 1. Otherwise, we can divide by the leading coefficient of L from the left. $\mathbb{K}(x)[S;\sigma,0]$ is a right Euclidean domain, i.e. we have the notion of the greatest common right divisor and the least common left multiple of operators and we are able to determine both algorithmically. Consequently, $\mathbb{K}(x)[S;\sigma,0]$ is a principal left ideal domain and every left ideal is generated by the greatest common right divisor of a given set of generators.

Consider the ring $\mathbb{K}^{\mathbb{N}}$ of all sequences in $\mathbb{K}$ with component-wise addition and the Hadamard product (i.e. component-wise product) as multiplication. We follow [14] in identifying sequences as equal if they only differ in finitely many terms. This will prove beneficial in two ways. Firstly, it allows us to define the action of operators on sequences in a natural way. Secondly, disregarding finitely many starting values makes it possible to identify unnecessary loop variables whose values are eventually equal to the values of other variables and therefore can be computed outside of any while loop. Let $\sim$ be the equivalence relation on $\mathbb{K}^{\mathbb{N}}$ defined by

$$s \sim t :\Leftrightarrow s - t \text{ has finitely many non-zero terms.}$$

We then set $\mathcal{S}$ to be the quotient ring $\mathbb{K}^{\mathbb{N}}/\!\sim$. Subsequently, it will not be necessary to distinguish between $t \in \mathbb{K}^{\mathbb{N}}$ and $\pi(t) \in \mathcal{S}$, where $\pi : \mathbb{K}^{\mathbb{N}} \to \mathcal{S}$ is the canonical homomorphism. The field $\mathbb{K}$ can be embedded in $\mathcal{S}$ via the map $c \mapsto (c)_{n\in\mathbb{N}}$. The action of an operator in $\mathbb{K}(x)[S;\sigma,0]$ on an element in $\mathcal{S}$ is defined by the map

$$\tau : \mathbb{K}(x)[S;\sigma,0] \times \mathcal{S} \to \mathcal{S}$$

$$\tau(L(S,x),t)(n) = \tau\left(\sum_{i=0}^{d} l_i(x)S^i, t\right)(n) := \sum_{i=0}^{d} l_i(n)t(n+i),$$

where the evaluation is well defined for all $n \geq n_0$ for some $n_0 \in \mathbb{N}$, and we set $L(t) := \tau(L,t) \in \mathcal{S}$. If $L(t) \equiv 0$, then we say that L is an *annihilator* of t (L *annihilates* t) and t is a *solution* of $L(t) = 0$. A sequence that is annihilated by a non-zero operator in $\mathbb{K}(x)[S;\sigma,0]$ is called a *holonomic sequence*. For a given sequence t, the set of all its annihilators forms a left ideal in $\mathbb{K}(x)[S;\sigma,0]$. We call it the *annihilator ideal* of t and denote it by $\text{ann}(t)$.

Example 2.2. Let $p(x)$ be a polynomial in $\mathbb{K}[x]$. The polynomial sequence $(p(n))_{n\in\mathbb{N}}$ is annihilated by the operator

$$L_1 = S - \frac{p(x+1)}{p(x)}.$$

L_1 is a generator of the annihilator ideal of p. Set $\Delta := S - 1$. Then $\tilde{p} = \Delta(p)$ is again a polynomial sequence with $\deg(\tilde{p}) < \deg(p)$. It follows that $L_2 = \Delta^{\deg(p)+1}$ is another annihilator of p in $\mathbb{K}(x)[S;\sigma,0]$ and its coefficients are independent of x. Since L_1 generates $\text{ann}(p)$, there exists an operator Q with $L_2 = QL_1$.

In our work, we focus on two different special kinds of holonomic sequences:

Definition 2.3. Let $t \in \mathcal{S}$. Then

- t is called *C-finite* if it is annihilated by an operator L in $\mathbb{K}(x)[S;\sigma,0]$ with only constant coefficients. ($l_i \in \mathbb{K}$)
- t is called *hypergeometric* if it is annihilated by an order 1 operator in $\mathbb{K}(x)[S;\sigma,0]$.

Example 2.4. We give some examples of commonly encountered sequences.

- As was shown in Example 2.2, polynomial sequences are both, C-finite and hypergeometric.
- Rational function sequences $(r(n))_{n\in\mathbb{N}}$, $r \in \mathbb{K}(x) \setminus \mathbb{K}[x]$, are hypergeometric but not C-finite.
- The factorial sequence $(n!)_{n\in\mathbb{N}}$ is hypergeometric but not C-finite.
- The Fibonacci sequence $(f(n))_{n\in\mathbb{N}}$ with
$$f(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right),$$
is C-finite but not hypergeometric.
- The sequence of harmonic numbers $(h(n))_{n\in\mathbb{N}}$ with
$$h(n) = \sum_{i=1}^{n} \frac{1}{i},$$
is neither hypergeometric nor C-finite.

In a sufficiently large algebraic field extension $\overline{\mathbb{K}}/\mathbb{K}$, every C-finite sequence $(c(n))_{n\in\mathbb{N}}$ can be uniquely written (up to reordering) in the form

$$c(n) = p_1(n)\theta_1^n + p_2(n)\theta_2^n + \cdots + p_s(n)\theta_s^n,$$

for some $s \in \mathbb{N}$ and $p_i \in \mathbb{K}[x], \theta_i \in \overline{\mathbb{K}}$ for $i = 1,\dots,s$ with $\theta_i \neq \theta_j$ for $i \neq j$. For any $r \in \mathbb{K}(x)$ and $n \in \mathbb{N}$, $r(x)^{\underline{n}}$ is defined as $\prod_{i=0}^{n-1} r(x-i)$. Then every hypergeometric sequence $(h(n))_{n\in\mathbb{N}}$ can be uniquely written (up to reordering) in the form

$$h(n) = \theta^n r(n)((n+\zeta_1)^{\underline{n}})^{k_1}((n+\zeta_2)^{\underline{n}})^{k_2}\cdots((n+\zeta_\ell)^{\underline{n}})^{k_\ell},$$

for some $\ell \in \mathbb{N}$, $r(x) \in \mathbb{K}(x)$, $\theta \in \overline{\mathbb{K}}$, $\zeta_i \in \overline{\mathbb{K}}$ and $k_i \in \mathbb{Z}$ for $i = 1,\dots,\ell$, and the difference $\zeta_i - \zeta_j$ is not an integer for $i \neq j$. Without loss of generality, we can always assume that $\zeta_i \notin \{-1,-2,-3,\dots\} =: \mathbb{Z}^-$. Otherwise, $h(n)$ would be zero (for $k_i > 0$) or undefined (for $k_i < 0$) for all $n \geq -\zeta_i$ and wouldn't have to be computed in a while loop. From these closed forms it is immediate that finite sums and products of C-finite sequences are again C-finite and finite products of hypergeometric sequences are again hypergeometric. Sums of hypergeometric sequences are not necessarily hypergeometric, see Lemma 4.3. Subsequently, we will assume that $\mathbb{K}$ is large enough so that all occurring C-finite and hypergeometric sequences have a closed form representation in $\mathbb{K}$.

For more details on C-finite and hypergeometric sequences, as well as proofs for the facts given in this section, see [6].

For functions $f_1,\dots,f_m : \mathbb{U} \to \mathbb{K}$ with $\mathbb{N} \subset \mathbb{U} \subset \mathbb{K}$ that are algebraically independent over $\mathbb{K}$, we distinguish between the polynomial ring $\mathbb{K}[f_1,\dots,f_m]$, where $f_1,\dots,f_m$ are used as variables, and the ring $\mathbb{K}[f_1(n),\dots,f_m(n)] \subset \mathcal{S}$ of all sequences $(t(n))_{n\in\mathbb{N}}$ of the form $t(n) = p(f_1(n),\dots,f_m(n))$ with $p \in \mathbb{K}[f_1,\dots,f_m]$. This distinction is important, as e.g. the function $\sin(x\cdot\pi)$ is transcendental over $\mathbb{K}$, but the sequence $(\sin(n\cdot\pi))_{n\in\mathbb{N}} = (0,0,0,\dots)$ is not, and thus $\mathbb{K}[\sin(n\cdot\pi)]$ is isomorphic to $\mathbb{K}$, but $\mathbb{K}[\sin(x\cdot\pi)]$ is not.

Remark. *In the context of this paper, since the operators in question emerge from program loops, we can safely assume that the rational function coefficients of any operator do not have poles in $\mathbb{N}$. Otherwise, a division by zero error would occur for some program input.*

2.2 Programming Model

We consider a simple programming model of single-path loops with rational function assignments. That is, nested loops and/or loops with conditionals are not yet handled in our work. Our programming model is thus given by the following loop pattern, written in a C-like syntax:

$$\begin{array}{l}\textbf{while } pred(v_1,\dots,v_m) \textbf{ do}\\ \quad v_1 := f_1(v_1,\dots,v_m);\\ \quad \vdots\\ \quad v_m := f_m(v_1,\dots,v_m);\\ \textbf{end while}\end{array} \tag{1}$$

where $v_1,\dots,v_m$ are (scalar) variables with values from $\mathbb{K}$, the f_i are rational functions over $\mathbb{K}$ in m variables and *pred* is a Boolean formula (loop condition) over $v_1,\dots,v_m$. In our approach however, we ignore loop conditions and treat program loops as non-deterministic programs. In [10], it is shown that the set of all affine equality invariants is not computable if the programming model includes affine equality tests/conditions. With this consideration, our programming model from (1) becomes:

$$\begin{array}{c}\textbf{while } \text{true } \textbf{do}\\ \vdots\\ \textbf{end while}\end{array} \tag{2}$$

Due to its particular importance in our reasoning, we suppose that there is always a variable n denoting the loop iteration counter. The initial value of n will always be $n = 0$ and n will be incremented by 1 at the end of each iteration.

Each program variable gives rise to a sequence $(v_i(n))_{n\in\mathbb{N}}$. For a program variable v, we allow ourselves to abuse the notation and also use the identifier v as a variable in polynomial rings as well as an identifier for the sequence $(v(n))_{n\in\mathbb{N}}$.

A polynomial loop invariant is a polynomial p over $\mathbb{K}$ in m variables such that $p(v_1(n),\dots,v_m(n)) = 0$ for all n. As observed in [8, 17], the set of all polynomial invariants forms a polynomial ideal in $\mathbb{K}[v_1,\dots,v_m]$, called the *polynomial invariant ideal* and we denote it by $I(v_1,\dots,v_m)$. For a subset $\{\tilde{v}_1,\dots,\tilde{v}_k\} \subset \{v_1,\dots,v_m\}$, we define

$$I(\tilde{v}_1,\dots,\tilde{v}_k) := I(v_1,\dots,v_m) \cap \mathbb{K}[\tilde{v}_1,\dots,\tilde{v}_k].$$

In general, polynomial loop invariants depend on the initial values of program variables. To simplify the presentation, we fix $\mathbb{K}$ to be

$$\mathbb{K} = \mathbb{F}(v_{1,1},\dots,v_{1,d_1},v_{2,1},\dots,v_{2,d_2},\dots,v_{m,1},\dots,v_{m,d_m}),$$

for a computable field $\mathbb{F}$ of characteristic zero that allows us to represent all occurring C-finite and hypergeometric sequences in closed form, and sufficiently many variables $v_{1,1},\dots,v_{m,d_m}$ that represent the initial values of the program variables $v_1,\dots,v_m$, where d_i is the order of the recurrence for v_i.

3 POLYNOMIAL INVARIANTS FOR P-SOLVABLE LOOPS

We now turn our attention to the class of P-solvable loops introduced in [8] that allows for computing all polynomial invariants.

Definition 3.1. An imperative loop with assignment statements only is called *P-solvable* if the sequence of each recursively changed program variable is C-finite and the ideal of all polynomial invariants over $\mathbb{K}$ is not the zero ideal.

Example 3.2. In [8], it is shown that integer division with remainder is P-solvable. Given the program:

while $y \le rem$ **do**
  $rem := rem - y$;
  $quo := quo + 1$;
end while

The ideal of polynomial loop invariants is shown to be

$$I(quo, rem, x, y) = \langle rem + quo \cdot y - y \cdot quo(0) - rem(0)\rangle.$$

With $quo(0) = 0$ and $rem(0) = x$, this gives $\langle rem + quo \cdot y - x\rangle$.

While P-solvable loops cover a wide class of program loops, there are several significant cases which do not fall into this class. Notably, multiplication with the loop counter n will generally result in loops that are not P-solvable.

Example 3.3. Consider the following loop with relevant loop variables a, b, c, d. The variables t_1, t_2 are temporary variables used to access previous values of a. Along with the loop counter n, we will not take them into consideration for the loop invariants in this example, as it is reasonable to assume that they will not be used outside of the loop.

while true **do**
  $t_1 := t_2$; $t_2 := a$;
  $a := 5(n+2)\cdot t_2 + 6\cdot(n^2+3\cdot n+2)\cdot t_1$;
  $b := 2\cdot b$;
  $c := 3\cdot(n+2)\cdot c$;
  $d := (n+2)\cdot d$;
  $n := n+1$;
end while

The program then satisfies the following system of recurrences:

$$\begin{cases} a(n+2) - 5(n+2)\cdot a(n+1) - 6(n^2+3n+2)\cdot a(n) = 0,\\ b(n+1) - 2\cdot b(n) = 0,\\ c(n+1) - 3(n+1)\cdot c(n) = 0,\\ d(n+1) - (n+1)\cdot d(n) = 0. \end{cases}$$

This loop is not P-solvable as, for example, the values of the variable c are determined by a sequence that is not C-finite (due to the multiplication between the program variables n and c). To the best of our knowledge, none of the existing invariant generation techniques are able to compute all polynomial invariants for such loops. In the next section, we extend the class of P-solvable loops, covering also programs as the one above, and introduce an automated approach to derive all polynomial invariants.

4 EXTENSION OF P-SOLVABLE LOOPS

4.1 Definition of Extended P-Solvable Loops

Consider the sequences $(v_1(n))_{n\in\mathbb{N}},\dots,(v_m(n))_{n\in\mathbb{N}}$ with values in $\mathbb{K}$ given by

$$v_i(n) = \sum_{k\in\mathbb{Z}^\ell} p_{i,k}(n,\theta_1^n,\dots,\theta_s^n)((n+\zeta_1)^{\underline{n}})^{k_1}\cdots((n+\zeta_\ell)^{\underline{n}})^{k_\ell} \quad (3)$$

where $s,\ell \in \mathbb{N}$, the $p_{i,k}$ are polynomials in $\mathbb{K}(x)[y_1,\dots,y_s]$, not identically zero for finitely many $k \in \mathbb{Z}^\ell$, the θ_i are elements of $\mathbb{K}$ and the ζ_j are elements of $\mathbb{K}\setminus\mathbb{Z}^-$ with $\theta_i \ne \theta_j$ and $\zeta_i - \zeta_j \notin \mathbb{Z}$ for $i \ne j$. This class of sequences comprises C-finite sequences as well as hypergeometric sequences and sums and Hadamard products of C-finite and hypergeometric sequences, which could not be handled in automated invariant generation before. We give an extension of Definition 3.1 based on this class of sequences

Definition 4.1. An imperative loop with assignment statements only is called *extended P-solvable* if the sequence of each recursively changed program variable v is of the form (3).

Note that in Definition 4.1, we drop the requirement of Definition 3.1 that the ideal of algebraic relations is not the zero ideal. This change is just for convenience.

While it is obvious that the inclusion of hypergeometric terms in extended P-solvable loops allows assignments of the form $v := r(n)v$, where r is a rational function in $\mathbb{K}(x)$, it also allows assignments that turn into higher order recurrences, as illustrated in Example 4.2. It also allows for assignments of the form $v_2 := r(v_1)v_2$, with $r \in \mathbb{K}(x)$, as long as the closed form of v_1 is a rational function in n.

4.2 Detecting Extended P-Solvable Loops

In order to employ the ideas we develop in Section 4.3 for finding algebraic relations in extended P-solvable loops, we have to be able to detect sequences of the form (3). This means, given a recurrence operator R of order d and starting values $s_0,\dots,s_{d-1}$, compute, if possible, p_k, θ_i and ζ_j as in (3) such that v is the solution of $R(v) = 0$ with $v(n) = s_n$ for $n \in \{0,\dots,d-1\}$. We can write v as a sum of hypergeometric sequences:

$$v(n) = h_1(n) + \cdots + h_w(n), \text{ where}$$
$$h_i(n) = q_i(n)\tilde{\theta}_i^n((n+\zeta_1)^{\underline{n}})^{k_{i,1}}\cdots((n+\zeta_\ell)^{\underline{n}})^{k_{i,\ell}},$$

with $q_i \in \mathbb{K}(x)$, $\tilde{\theta}_i \in \mathbb{K}$, and $k_i \in \mathbb{Z}^\ell$. Note that we use $\tilde{\theta}_i$ instead of θ_i since the exponential sequence for each summand h_i can be a product of several θ_j^n. We can assume without loss of generality that the h_i are linearly independent over $\mathbb{K}(n)$. In fact, if $h_1(n) = r_2(n)h_2(n)+\cdots+r_w(n)h_w(n)$, we can set $\tilde{h}_1 = (1+r_2)h_2, \dots, \tilde{h}_{w-1} = (1+r_w)h_w$ and get $v(n) = \tilde{h}_1(n) + \cdots + \tilde{h}_{w-1}(n)$. Let L be the least common left multiple of the first order operators $L_1, \dots, L_w$ that annihilate $h_1, \dots, h_w$ respectively in the Ore algebra $\mathbb{K}(x)[S;\sigma,0]$, and let G be the generator of $\operatorname{ann}(v)$. We show that G and L are equal. Recall that we required all operators to have leading coefficient 1. By right division with remainder, we can write G as

$$\begin{aligned} G &= Q_1L_1 + q_1 \\ &= Q_2L_2 + q_2 \\ &\ \ \vdots \\ &= Q_wL_w + q_w, \end{aligned}$$

with $Q_1, \dots, Q_w \in \mathbb{K}(x)[S;\sigma,0]$ and some $q_1, \dots, q_w \in \mathbb{K}(x)$. We then get

$$0 = G(v) = G(h_1+\cdots+h_w) = G(h_1)+\cdots+G(h_w) = q_1h_1+\dots q_wh_w.$$

Since the h_i are linearly independent, we have $q_1 = \cdots = q_w = 0$, and so, $L_1, \dots, L_w$ are right factors of G. This shows that the order of L is less than or equal to the order of G, and, because of $L \in \operatorname{ann}(v)$, G is a right factor of L. Hence $L = G$.

Every annihilator of v is a multiple of G and therefore also an annihilator of h_i, and so we can use Petkovšek's algorithm [15] to determine p_k,θ_i and ζ_j as in (3). More precisely, given an operator $R \in \mathbb{K}(x)[S;\sigma,0]$ of order d and starting values $s_0, \dots, s_{d-1}$, we compute v as in (3) such that $R(v) = 0$ (if possible), by computing all hypergeometric solutions of R. This gives θ_i, ζ_i and p_i, linearly dependent on parameters $u_1, \dots, u_w \in \mathbb{K}$. Next, we solve the linear system $v(i) = s_i$ in terms of the u_j. Any solution then gives rise to a sequence $(v(n))_{n\in\mathbb{N}}$ with the desired properties.

Example 4.2. For the recurrence for a in Example 3.3, we compute two hypergeometric solutions using Petkovšek's algorithm:

$$h_1 = (-1)^n n!, \quad h_2 = 6^n n!$$

Thus, we get

$$a(n) = ((-1)^n u_1 + 6^n u_2)n!$$

with the relations $a(0) = u_1 + u_2$ and $a(1) = 6u_2 - u_1$ stemming from the starting values of a. Since b, c, d are given by first order recurrences, their closed forms can be easily computed:

$$b(n) = 2^n b(0), \quad c(n) = 3^n n! c(0), \quad d(n) = n! d(0).$$

It follows that the program loop given in Example 3.3 is extended P-solvable.

4.3 The Ideal of Algebraic Relations

We now turn to the problem of, given sequences $v_1, \dots, v_m$ as in (3), how to compute a basis for the ideal $I(v_1, \dots, v_m)$ of all algebraic relations among the v_i. We proceed by identifying the terms $(n+\zeta_i)^{\underline{n}}$ that are algebraically independent over $\mathbb{K}(n, \theta_1^n, \dots, \theta_s^n)$. For this, we use basic properties of sums and products of hypergeometric terms. First, we state a necessary condition for a finite sum of hypergeometric terms to be again hypergeometric.

Lemma 4.3. *Let $h_1, \dots, h_w$ be hypergeometric sequences. If the sum $h_1 + \cdots + h_w$ is hypergeometric, then there exist integers $i, j \in \{1, \dots, w\}$, $i \neq j$, and a rational function $r(x) \in \mathbb{K}(x)$ such that $h_i(n) = r(n)h_j(n)$.*

Proof. We prove the claim by induction on w. For the case $w = 1$, there is nothing to show. Now suppose the claim holds for some $(w-1) \in \mathbb{N}^*$. There is a rational function $r_h(x) \in \mathbb{K}(x)$ such that

$$\sum_{i=1}^{w} h_i(n+1) = r_h(n) \sum_{i=1}^{w} h_i(n).$$

Let $r_i \in \mathbb{K}(x)$ be such that $h_i(n+1) = r_i(n)h_i(n)$. We then get

$$\sum_{i=1}^{w} (r_i(n) - r_h(n))h_i(n) = 0. \tag{4}$$

We first treat the case in which for all i, $(r_i(x) - r_h(x))$ is not zero. Then, bringing $(r_w(n) - r_h(n))h_w(n)$ in (4) to the other side yields

$$\sum_{i=1}^{w-1} (r_i(n) - r_h(n))h_i(n) = (r_h(n) - r_w(n))h_w(n).$$

The sequence $(r_h(n) - r_w(n))h_w(n)$ is hypergeometric, and by the induction hypothesis it follows that there are i, j and a rational function $\tilde{r}$ with $(r_i(n) - r_h(n))h_i(n) = \tilde{r}(n)(r_j(n) - r_h(n))h_j(n)$. Dividing by $r_i(n) - r_h(n)$ proves the claim. For the case that there is an i with $(r_i(x) - r_h(x)) = 0$, the left hand side of (4) is a sum of fewer than w hypergeometric terms and the right hand side is hypergeometric. The induction hypothesis then again yields suitable i, j and $r(x)$. □

Example 4.4. The sum $2n!+(n+3)!$ is hypergeometric and we have the relation $2n! = \frac{2}{(n+1)(n+2)}(n+3)!$. In contrast, $1 + n!$ is not hypergeometric because there would have to be rational function $r(n)$ with $1 = r(n)n!$, which would imply that $n!$ is a rational function. The sum $n! + (n+\frac{1}{2})^{\underline{n}} - (n+1)!$ is also not hypergeometric although $n! = \frac{1}{n+1}(n+1)!$. We can rewrite the sum as $-n \cdot n! + (n+\frac{1}{2})^{\underline{n}}$ and, as we will see in Lemma 4.5, there is no rational function $r(n)$ such that $-n \cdot n! = r(n)(n+\frac{1}{2})^{\underline{n}}$.

The next lemma gives a characterization of when the quotient of two hypergeometric sequences is a rational function sequence. Together with Lemma 4.3, this then will yield the algebraic independence of certain hypergeometric sequences in Lemma 4.6.

Lemma 4.5. *Let $\zeta_1, \dots, \zeta_\ell \in \mathbb{K} \setminus \mathbb{Z}^-$ be such that for all $i, j = 1, \dots, \ell$ with $i \neq j$, we have $\zeta_i - \zeta_j \notin \mathbb{Z}$. Then for $k_1, \dots, k_\ell \in \mathbb{N}$, $c_1, \dots, c_\ell \in \mathbb{N}$, and $\theta_1, \theta_2 \in \mathbb{K}$, there is a rational function $r(x) \in \mathbb{K}(x)$ such that*

$$\begin{aligned} &\theta_1^n \cdot ((n-\zeta_1)^{\underline{n}})^{k_1} \cdots ((n-\zeta_\ell)^{\underline{n}})^{k_\ell} = \\ &\qquad r(n) \cdot \theta_2^n \cdot ((n-\zeta_1)^{\underline{n}})^{c_1} \cdots ((n-\zeta_\ell)^{\underline{n}})^{c_\ell}, \end{aligned}$$

if and only if $\theta_1 = \theta_2$ and $(k_1, \dots, k_\ell) = (c_1, \dots, c_\ell)$.

Proof. If $\theta_1 = \theta_2$ and $(k_1, \dots, k_\ell) = (c_1, \dots, c_\ell)$, then we can set $r(x) = 1$. For the other direction, we have

$$\underbrace{\left(\frac{\theta_1}{\theta_2}\right)^n ((n-\zeta_1)^{\underline{n}})^{k_1-c_1} \cdots ((n-\zeta_\ell)^{\underline{n}})^{k_\ell-c_\ell}}_{\text{hypergeometric}} = r(n).$$

A hypergeometric term h is a rational function if and only if its shift quotient $h(x+1)/h(x)$ can be written in the form

$$q(x) = \frac{g(x)f(x+1)}{g(x+1)f(x)},$$

with $f, g \in \mathbb{K}[x]$. Therefore, for any root in the numerator of $q(x)$ there is a root in integer distance in the denominator of $q(x)$, which, by the condition on the ζ_i, is not possible if $(k_1, \ldots, k_\ell) \neq (c_1, \ldots, c_\ell)$. The quotient θ_1/θ_2 is equal to the quotient of the leading coefficients of $g(x)f(x+1)$ and of $g(x+1)f(x)$, which in turn is equal to 1. It follows that $\theta_1 = \theta_2$. □

Lemma 4.6. *Let $\theta_1, \ldots, \theta_s \in \mathbb{K}$ and $\zeta_1, \ldots, \zeta_\ell \in \mathbb{K} \setminus \mathbb{Z}^-$. The sequences $(n+\zeta_1)^{\underline{n}}, (n+\zeta_2)^{\underline{n}}, \ldots, (n+\zeta_\ell)^{\underline{n}}$ are algebraically independent over $\mathbb{K}(n, \theta_1^n, \ldots, \theta_s^n)$ if and only if there are no $i, j \in \{1, \ldots, \ell\}$, $i \neq j$ such that $\zeta_i - \zeta_j \in \mathbb{Z}$.*

Proof. If there are $i, j \in \{1, \ldots, \ell\}$, $i \neq j$ with $\zeta_i - \zeta_j = k \in \mathbb{N}$, then we get the algebraic relation

$$(n+\zeta_i)^{\underline{n}} \cdot \prod_{w=1}^{k} (\zeta_j + w) = (n+\zeta_j)^{\underline{n}} \cdot \prod_{w=1}^{k} (n + w + \zeta_j).$$

For $k < 0$ we can simply change the roles of ζ_i and ζ_j.
Conversely, let p be a nonzero polynomial over $\mathbb{K}(n, \theta_1^n, \ldots, \theta_s^n)$ in ℓ variables. After clearing denominators in the coefficients of p, we can write $p((n+\zeta_1)^{\underline{n}}, \ldots, (n+\zeta_\ell)^{\underline{n}})$ as a sum of the form

$$\sum_{i \in \mathbb{N}, k \in \mathbb{N}^\ell} p_{i,k}(n) \tilde{\theta}_i^n ((n+\zeta_1)^{\underline{n}})^{k_1} \cdots ((n+\zeta_\ell)^{\underline{n}})^{k_\ell}.$$

Assume that $p((n+\zeta_1)^{\underline{n}}, \ldots, (n+\zeta_\ell)^{\underline{n}}) = 0$. Then, by Lemma 4.3, there have to be terms $(i, k), (j, c) \in \mathbb{N}^{\ell+1}$, $(i, k) \neq (j, c)$ and a rational function $r(x) \in \mathbb{K}(x)$ with

$$\begin{aligned} p_{i,k}(n) \tilde{\theta}_i^n ((n-\zeta_1)^{\underline{n}})^{k_1} \cdots ((n-\zeta_\ell)^{\underline{n}})^{k_\ell} = \\ r(n) p_{j,c}(n) \tilde{\theta}_j^n ((n-\zeta_1)^{\underline{n}})^{c_1} \cdots ((n-\zeta_\ell)^{\underline{n}})^{c_\ell}, \end{aligned}$$

By Lemma 4.5, this can only be the case if there are $\zeta_i \neq \zeta_j$ in integer distance, which contradicts the condition on the ζ_i. □

Example 4.7. Let h_1, h_2, h_3 be hypergeometric sequences given by $h_1(0) = h_2(0) = h_3(0) = 1$ and

$$h_1(n+1) = (n^2 + \frac{3}{2}n + \frac{1}{2}) h_1(n), \; h_2(n+1) = (n+1) h_2(n),$$

$$h_3(n+1) = \frac{2n^3 + 9n^2 + 10n + 3}{2n+4} h_3(n).$$

The closed forms then are

$$h_1(n) = \prod_{i=0}^{n} (i^2 + \frac{3}{2}i + \frac{1}{2}) = \prod_{i=0}^{n} (i+1)(i + \frac{1}{2}) = (n+1)^{\underline{n}} (n + \frac{1}{2})^{\underline{n}},$$

$$h_2(n) = \prod_{i=0}^{n} (i+1) = (n+1)^{\underline{n}},$$

$$h_3(n) = \prod_{i=0}^{n} \frac{2x^3 + 13i^2 - 24i + 9}{2i+4} = \prod_{i=0}^{n} \frac{(i+3)(i+\frac{1}{2})(2(i+1)+4)}{2i+4} = 2(n+2)(n+3)^{\underline{n}} (n + \frac{1}{2})^{\underline{n}}.$$

From Lemma 4.6 it follows that $(n + \frac{1}{2})^{\underline{n}}$ and $(n+1)^{\underline{n}}$ and consequently h_1, h_2 are algebraically independent over $\mathbb{K}(n)$, but h_1, h_3 are not. In fact

$$(n+2)^2(n+3)h_1(n) - 3h_3(n) = 0.$$

Lemma 4.6 allows us to represent the sequences arising in extended P-solvable loops as rational function sequences over the field $\mathbb{K}(n, \theta_1^n, \ldots, \theta_s^n)$ as follows: Let $v_1, \ldots, v_m$ be of the form (3) and let $\tilde{Z} = \{\tilde{\zeta}_1, \ldots \tilde{\zeta}_w\}$ be a subset of $Z = \{\zeta_1, \ldots, \zeta_\ell\}$ such that there are no $i, j = 1, \ldots, w$, $i \neq j$, with $\tilde{\zeta}_i - \tilde{\zeta}_j \in \mathbb{Z}$ and for each $\zeta \in Z \setminus \tilde{Z}$ there exists an i such that $\tilde{\zeta}_i - \zeta \in \mathbb{Z}$. Let $z_1, \ldots, z_\ell \in \mathbb{K}(x, y_1, \ldots, y_w)$ be such that

$$z_i(n, (n-\tilde{\zeta}_1)^{\underline{n}}, \ldots, (n-\tilde{\zeta}_w)^{\underline{n}}) = (n-\zeta_i)^{\underline{n}},$$

for all $n \in \mathbb{N}$ and $i = 1, \ldots, \ell$. Then there exist $k_1, \ldots, k_m \in \mathbb{Z}^\ell$ with

$$\begin{aligned} v_i(n) = \sum_{j \in \mathbb{Z}} & p_{i,j}(n, \theta_1^n, \ldots, \theta_s^n) \cdot \\ & \prod_{1 \leq t \leq \ell} z_t(n, (n-\tilde{\zeta}_1)^{\underline{n}}, \ldots, (n-\tilde{\zeta}_w)^{\underline{n}})^{k_{i,t}}. \end{aligned}$$

Substituting variables v_i for $v_i(n)$, h_i for $(n-\tilde{\zeta}_i)^{\underline{n}}$, e_i for θ_i^n and x for n then gives

$$[v_i = r_i(x, e_1, \ldots, e_s, h_1, \ldots, h_w)]_{\substack{v_i \to v_i(n), h_i \to (n-\tilde{\zeta}_i)^{\underline{n}}, \\ e_i \to \theta_i^n, x \to n}}$$

where r_i is a rational function over $\mathbb{K}$ in $1 + s + w$ variables. Now, with the help of the ideal of algebraic relations among $\theta_1^n, \ldots, \theta_s^n$, which can be computed via the algorithm given in [7], we can compute the ideal of all algebraic dependencies among the program variables of an extended P-solvable loop as the ideal of algebraic relations among rational functions.

Proposition 4.8. *Let $(v_1(n))_{n\in\mathbb{N}}, \ldots, (v_m(n))_{n\in\mathbb{N}}$ be sequences of the form (3) and consider the corresponding rational functions $r_1, \ldots, r_m$ in $\mathbb{K}(x, e_1, \ldots, e_s, h_1, \ldots, h_w)$ as above. For each $i = 1, \ldots, m$, write $r_i = f_i/g_i$ with coprime polynomials f_i, g_i over $\mathbb{K}$. Denote by $I(\theta_1^n, \ldots, \theta_s^n)$ the ideal of algebraic relations among the $\theta_1^n, \ldots, \theta_s^n$ in $\mathbb{K}[e_1, \ldots, e_s]$. Then the ideal of algebraic relations among the sequences $(v_1(n))_{n\in\mathbb{N}}, \ldots, (v_m(n))_{n\in\mathbb{N}}$ in $\mathbb{K}[v_1 \ldots, v_m]$ is given by*

$$\begin{aligned} I(v_1, \ldots, v_m) = (I(\theta_1^n, \ldots, \theta_s^n) + \\ \langle g_1 v_1 - f_1, \ldots, g_m v_m - f_m \rangle) \cap \mathbb{K}[v_1, \ldots, v_m]. \end{aligned}$$

Proof. The proposition follows immediately from the fact that the ideal of algebraic dependencies among a set of rational functions

$$\frac{f_1(x, e_1, \ldots, e_s, h_1, \ldots, h_w)}{g_1(x, e_1, \ldots, e_s, h_1, \ldots, h_w)}, \ldots, \frac{f_m(x, e_1, \ldots, e_s, h_1, \ldots, h_w)}{g_m(x, e_1, \ldots, e_s, h_1, \ldots, h_w)},$$

in the polynomial ring $\mathbb{K}[v_1, \ldots, v_m]$ is given by

$$\begin{aligned} &\langle g_1(x, e_1, \ldots, e_s, h_1, \ldots, h_w) v_1 - f_1(x, e_1, \ldots, e_s, h_1, \ldots, h_w), \ldots, \\ &g_m(x, e_1, \ldots, e_s, h_1, \ldots, h_w) v_m - f_m(x, e_1, \ldots, e_s, h_1, \ldots, h_w) \rangle \\ &\cap \mathbb{K}[v_1, \ldots, v_m], \end{aligned}$$

and that by Lemma 4.6 there are no algebraic relations over the field $\mathbb{K}(n, \theta_1^n, \ldots, \theta_s^n)$ among the terms $(n-\tilde{\zeta}_i)^{\underline{n}}$ with $\tilde{\zeta}_i$ as above for $i = 1, \ldots, w$. □

Example 4.9. We compute the ideal of algebraic relations among a, b, c, d given in Example 3.3. First, we compute the ideal of algebraic relations among $(-1)^n, 2^n, 3^n$ and 6^n with corresponding variables e_{-1}, e_2, e_3, e_6. We get

$$I((-1)^n, 2^n, 3^n, 6^n) = \langle e_{-1}^2 - 1, e_2e_3 - e_6\rangle.$$

Now we can compute the ideal of algebraic relations among a, b, c, d by adding the relations $a - (u_1e_{-1} - u_2e_6)f, u_1 + u_2 - a(0), -u_1 + 6u_2 - a(1), b - b(0)e_3, c - c(0)e_2f, d - d(0)f$, where f is used to model $n!$, and eliminate the variables $u_1, u_2, e_{-1}, e_2, e_3, e_6$ and f.

$$\begin{aligned}
&I(a, b, c, d) = \\
&(I(2^n, 3^n, 1 + 6^n) + \langle a - (u_1e_{-1} + u_2e_6)f, u_1 + u_2 - a(0), \\
&\quad - u_1 + 6u_2 - a(1), b - b(0)e_2, c - c(0)e_3f, d - d(0)f\rangle) \\
&\quad \cap \mathbb{K}[a, b, c, d] = \\
&\langle d(0)^2((-7b(0)c(0)a + a(0)bc)^2 + a(1)bc(bc(a(1) + 2a(0))- \\
&\quad 14b(0)c(0)a)) - (b(0)c(0)d(-6a(0) + a(1)))^2\rangle.
\end{aligned}$$

For instance, with the starting values $a(0) = 2, a(1) = 5$ and $b(0) = c(0) = d(0) = 1$ we get the relation

$$b^2c^2 - 2abc + a^2 - d^2,$$

with

$$a = ((-1)^n + 6^n)n!, \quad b = 2^n, \quad c = 3^n n!, \quad d = n!.$$

Remark. *Proposition 4.8 can easily be turned into an algorithm with the help of Gröbner bases, which allow the elimination of variables. While computationally demanding, the use of Gröbner bases is viable in part because of the highly optimized tools that are available in modern computer algebra systems and in part because, as observed empirically in our experiments, the polynomial systems arising in practice in this context are typically small and easy to compute.*

5 IMPLEMENTATION

The techniques presented in this paper are implemented in the open source Mathematica software package Aligator[1] [9], available for download at

https://ahumenberger.github.io/aligator/

We give an illustrative example of the provided facilities.

Example 5.1. We compute the ideal of algebraic relations among the program variables a, b, c, d, e, f as given in the following loop. The loop exhibits two first-order and two second-order recurrence relations (a, e and b, d resp.), which Aligator could not handle before. Furthermore we have two first-order C-finite recurrence relations (c, f).

```
In[1]:= Aligator[
  WHILE[True,
    a := 3(n + 3/2)a;
    s1 := s2; s2 := b;
    b := 5(3/2 + n)s2 - 3/2(1 + 2n)(3 + 2n)s1;
    c := -3c + 2;
    t1 := t2; t2 := d;
    d := 4(4 + n)t2 - 3(3 + n)(4 + n)t1;
    e := (n + 4)e;
    f := 2f],
  LoopCounter → n,
  IniVal → {
    t1 := 1; t2 := 1;
    s1 := 1; s2 := 2;
    a := 3; b := 1;
    c := 1; d := 3;
    e := 2; f := 5}]
```

[1]Aligator requires the Mathematica packages Hyper [13], Dependencies [7] and FastZeil [12], of which the latter two are part of the compilation package ErgoSum [5].

The input is given to Aligator in form of a while loop, and two optional arguments: **LoopCounter** (default: **i**) and **IniVal** (default: {}). The former is for specifying which variable within the loop corresponds to the loop counter, whereas the latter is for specifying the initial values of the program variables. If no initial values are given, then the invariants contain the starting values in the form of **a[0]**, representing the initial value of a.

The following output of Aligator is a reduced Gröbner basis (up to normalization of coefficients) of the ideal of all algebraic relations among a, b, c, d, e and f. The loop counter is eliminated via Gröbner basis computation.

$$\begin{aligned}
\text{Out[1]=}\ &\{-(2d - 3e)^2, \\
&30 (a + b) (2d - 3e), \\
&8af (2d - 3e), \\
&450ab (1 - 2c)^2 + 225b^2 (1 - 2c)^2 + \\
&\quad a^2 (225 (1 - 2c)^2 - 16f^2)\}
\end{aligned}$$

By setting the option **EqualityInvariants → True**, a conjunction of simplified equality constraints induced by the elements of the basis is printed.

In its current version, Aligator requires the occurring recurrences to be linear and uncoupled. It is planned to loosen this restrictions in future versions.

6 CONCLUSION AND FUTURE WORK

We extended the class of P-solvable loops to include sums and products of hypergeometric and C-finite sequences. This was made possible by identifying algebraically independent factors in hypergeometric terms and then viewing the sequences in question as rational function sequences over a transcendental field extension. The implementation in Mathematica underlines the practicality of the approach.

There are several promising directions in which we plan to expand this line of research. Obviously, it is very desirable to include more types of recurrences in P-solvable loops. These include further subclasses of the class of holonomic sequences as well as partial and non-linear recurrence equations. It is advisable to conduct a careful study on which kind of recurrences are relevant in practice and also good-natured from a mathematical perspective. Uncoupling techniques for systems of recurrence equations can also prove to be helpful in this context.

Another possible extension is to consider nested loops. With the help of $\Pi\Sigma^*$-theory [19], it might be possible to derive invariants for the outermost loop, although the inner loops are not P-solvable by themselves.

Acknowledgments We want to thank the reviewers for their helpful comments and remarks. In particular, we are very grateful for the exceptionally careful and rigorous reading of this manuscript by one reviewer which helped us to greatly improve our work.

REFERENCES

[1] M. Bronstein and M. Petkovšek. 1996. An Introduction to Pseudo-Linear Algebra. *Theoretical Computer Science* 157 (1996), 3–33.

[2] D. Cachera, T. P. Jensen, A. Jobin, and F. Kirchner. 2012. Inference of Polynomial Invariants for Imperative Programs: A Farewell to Gröbner Bases. In *Static Analysis - 19th International Symposium, SAS 2012 (LNCS)*, Vol. 7460. 58–74.

[3] S. de Oliveira, S. Bensalem, and V. Prevosto. 2016. Polynomial Invariants by Linear Algebra. In *Proc. of ATVA*, C. Artho, A. Legay, and D. Peled (Eds.). Springer, 479–494. DOI: http://dx.doi.org/10.1007/978-3-319-46520-3_30

[4] A. Farzan and Z. Kincaid. 2015. Compositional Recurrence Analysis. In *Proc. of FMCAD*. FMCAD Inc, Austin, TX, 57–64. http://dl.acm.org/citation.cfm?id=2893529.2893544

[5] Research Institute for Symbolic Computation. 2016. Mathematic Package Ergo-Sum. (2016). http://www.risc.jku.at/research/combinat/software/ergosum/

[6] M. Kauers and P. Paule. 2011. *The Concrete Tetrahedron* (1st ed.). Springer Wien.

[7] M Kauers and B. Zimmermann. 2008. Computing the algebraic relations of C-finite sequences and multisequences. *Journal of Symbolic Computation* 43, 11 (2008), 787 – 803. DOI: http://dx.doi.org/10.1016/j.jsc.2008.03.002

[8] L. Kovács. 2007. *Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema.* Ph.D. Dissertation. RISC, Johannes Kepler University Linz.

[9] L. Kovács. 2008. Aligator: A Mathematica Package for Invariant Generation (System Description). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings (Lecture Notes in Computer Science)*, A. Armando, P. Baumgartner, and G. Dowek (Eds.), Vol. 5195. Springer, 275–282. DOI: http://dx.doi.org/10.1007/978-3-540-71070-7_22

[10] M. Müller-Olm and H. Seidl. 2004. *A Note on Karr's Algorithm.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1016–1028. DOI: http://dx.doi.org/10.1007/978-3-540-27836-8_85

[11] Ø. Ore. 1933. Theory of Non-Commutative Polynomials. *Annals of Mathematics* 34(3) (1933), 480–508.

[12] P. Paule and M. Schorn. 1995. A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities. *Journal of Symbolic Computation* 20 (1995), 673 – 698.

[13] M. Petkovšek. 1998. Mathematic Package Hyper. (1998). http://www.fmf.uni-lj.si/~petkovsek/

[14] M. Petkovšek, H.S. Wilf, and D. Zeilberger. 1996. *A = B.* Peters. https://www.math.upenn.edu/~wilf/Downld.html

[15] M. Petkovšek. 1992. Hypergeometric solutions of linear recurrences with polynomial coefficients. *Journal of Symbolic Computation* 14, 2–3 (1992), 243 – 264.

[16] E. Rodriguez-Carbonell and D. Kapur. 2007. Automatic Generation of Polynomial Invariants of Bounded Degree using Abstract Interpretation. *J. Science of Computer Programming* 64, 1 (2007), 54–75.

[17] E. Rodríguez-Carbonell and D. Kapur. 2007. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation* 42, 4 (2007), 443 – 476. DOI: http://dx.doi.org/10.1016/j.jsc.2007.01.002

[18] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. 2004. Non-linear Loop Invariant Generation Using Gröbner Bases. In *Proc. of POPL*. ACM, New York, NY, USA, 318–329. DOI: http://dx.doi.org/10.1145/964001.964028

[19] C. Schneider. 2017. Summation Theory II: Characterizations of $R\Pi\Sigma$-extensions and algorithmic aspects. *J. Symb. Comput.* 80, 3 (2017), 616–664. http://arxiv.org/abs/1603.04285 arXiv:1603.04285 [cs.SC].

[20] R. Sharma, S. Gupta, B. Hariharan, A. Aiken, P. Liang, and A. V. Nori. 2013. A Data Driven Approach for Algebraic Loop Invariants. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013 (LNCS)*, Vol. 7792. 574–592.