Invariant Generation for Multi-Path Loops with Polynomial Assignments

Andreas Humenberger, Maximilian Jaroschek, and 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. Program analysis requires the generation of program properties expressing conditions to hold at intermediate program locations. When it comes to programs with loops, these properties are typically expressed as loop invariants. In this paper we study a class of multi-path program loops with numeric variables, in particular nested loops with conditionals, where assignments to program variables are polynomial expressions over program variables. We call this class of loops *extended P-solvable* and introduce an algorithm for generating all polynomial invariants of such loops. By an iterative procedure employing Gröbner basis computation, our approach computes the polynomial ideal of the polynomial invariants of each program path and combines these ideals sequentially until a fixed point is reached. This fixed point represents the polynomial ideal of all polynomial invariants of the given extended P-solvable loop. We prove termination of our method and show that the maximal number of iterations for reaching the fixed point depends linearly on the number of program variables and the number of inner loops. In particular, for a loop with m program variables and r conditional branches we prove an upper bound of $m \cdot r$ iterations. We implemented our approach in the Aligator software package. Furthermore, we evaluated it on 18 programs with polynomial arithmetic and compared it to existing methods in invariant generation. The results show the efficiency of our approach.

1 Introduction

Reasoning about programs with loops requires loop invariants expressing properties that hold before and after every loop iteration. The difficulty of generating such properties automatically comes from the use of non-linear arithmetic,

* All authors are supported by the ERC Starting Grant 2014 SYMCAR 639270. Furthermore, we acknowledge funding from the Wallenberg Academy Fellowship 2014 TheProSE, the Swedish VR grant GenPro D0497701, and the Austrian FWF research project RiSE S11409-N23. We also acknowledge support from the FWF project W1255-N23.

unbounded data structures, complex control flow, just to name few of the reasons. In this paper we focus on multi-path loops with numeric variables and polynomial arithmetic and introduce an automated approach inferring *all* loop invariants as polynomial equalities among program variables. For doing so, we identify a class of multi-path loops with nested conditionals, where assignments to program variables are polynomial expressions over program variables. Based on our previous work [4], we call this class of loops *extended P-solvable*. Compared to [4] where only single-path programs with polynomial arithmetic were treated, in this paper we generalize the notion of extended P-solvable loops to multi-path loops; single-path loops being thus a special case of our method.

For the class of extended P-solvable loops, we introduce an automated approach computing all polynomial invariants. Our work exploits the results of [17,9] showing that the set of polynomial invariants forms a polynomial ideal, called the polynomial invariant ideal. Hence, the task of generating all polynomial invariants reduces to the problem of generating a basis of the polynomial invariant ideal. Following this observation, given an extended P-solvable loop with nested conditionals, we proceed as follows: we (i) turn the multi-path loop into a sequence of single-path loops, (ii) generate the polynomial invariant ideal of each single-path loop and (iii) combine these ideals iteratively until the polynomial invariant ideal of the multi-path loop is derived.

A crucial property of extended P-solvable loops is that the single-path loops corresponding to one path of the multi-path loop are also extended P-solvable. For generating the polynomial invariant ideal of extended P-solvable single-path loops, we model loops by a system of algebraic recurrences, compute the closed forms of these recurrences by symbolic computation as described in [4] and compute the Gröbner basis of the polynomial invariant ideal from the system of closed forms. When combining the polynomial invariant ideals of each extended P-solvable single-path loop, we prove that the "composition" maintains the properties of extended P-solvable loops. Further, by exploiting the algebraic structures of the polynomial invariant ideals of extended P-solvable loops, we prove that the process of iteratively combining the polynomial invariant ideals of each extended P-solvable single-path loop is finite. That is, a fixed point is reached in a finite number of steps. We prove that this fixed point is the polynomial invariant ideal of the extended P-solvable loop with nested conditionals. We also show that reaching the fixed point depends linearly on the number of program variables and the number of inner loops. In particular, for a loop with m program variables and r inner loops (paths) we prove an upper bound of $m \cdot r$ iterations. The termination proof of our method implies the completeness of our approach: for an extended P-solvable loop with nested conditionals, our method computes all its polynomial invariants. This result generalizes and corrects the result of [10] on programs for more restricted arithmetic than extended P-solvable loops. Our class of programs extends the programming model of [10] with richer arithmetic and our invariant generation procedure also applies to [10]. As such, our proof of termination also yields a termination proof for [10].

We implemented our approach in the open source Mathematica package ALIGATOR and evaluated our method on 18 challenging examples. When compared to state-of-the-art tools in invariant generation, ALIGATOR performed much better in 14 examples out of 18.

The paper is organized as follows: We start by giving the necessary details about our programming model in Section 2.1 and provide background about polynomial rings and ideals in Section 2.2. In Section 3.1 we recall the notion of extended P-solvable loops from [4]. The lemmas and propositions of Section 3.2 will then help us to prove termination of our invariant generation procedure in Section 3.3. Finally, Section 4 describes our implementation in ALIGATOR, together with an experimental evaluation of our approach.

Related Work. Generation of non-linear loop invariants has been addressed in previous research. We discuss here some of the most related works that we are aware of.

The methods of [11,18] compute polynomial equality invariants by fixing an a priori bound on the degree of the polynomials. Using this bound, a template invariant of fixed degree is constructed. Properties of polynomial invariants, e.g. inductiveness, are used to generate constraints over the unknown coefficients of the template coefficients and these constraints are then solved in linear or polynomial algebra. An a priori fixed polynomial degree is also used in [16,2]. Unlike these approaches, in our work we do not fix the degree of polynomial invariants but generate all polynomial invariants (and not just invariants up to a fixed degree). Our restrictions come in the programming model, namely treating only loops with nested conditionals and polynomial arithmetic. For such programs, our approach is complete.

Another line of research uses abstract interpretation in conjunction with recurrence solving and/or polynomial algebra. The work of [17] generates all polynomial invariants of so-called *simple loops* with nested conditionals. The approach combines abstract interpretation with polynomial ideal theory. Our model of extended P-solvable loops is much more general than simple loops, for example we allow multiplication with the loop counter and treat algebraic, and not only rational, numbers in closed form solutions. Abstract interpretation is also used in [3,12,7] to infer non-linear invariants. The programming model of these works handle loops whose assignments induce linear recurrences with constant coefficients. Extended P-solvable loops can however yield more complex recurrence equations. In particular, when comparing our work to [7], we note that the recurrence equations of program variables in [7] correspond to a subclass of linear recurrences with constant coefficients: namely, recurrences whose closed form representations do not include non-rational algebraic numbers. Our work treats the entire class of linear recurrences with constant coefficients and even handles programs whose arithmetic operations induce a class of linear recurrences with polynomial coefficients in the loop counter. While the non-linear arithmetic of our work is more general than the one in [7], we note that the programming model of [7] can handle programs that are more complex than the

ones treated in our work, in particular due to the presence of nested loops and function/procedure calls. Further, the invariant generation approach of [7] is property-guided: invariants are generated in order to prove the safety assertion of the program. Contrarily to this, we generate all invariants of the program and not only the ones implying the safety assertion.

Solving recurrences and computing polynomial invariant ideals from a system of closed form solution is also described in [9]. Our work builds upon the results of [9] but generalizes [9] to extended P-solvable loops. Moreover, we also prove that our invariant generation procedure terminates. Our termination result generalizes [10] by handling programs with more complex polynomial arithmetic. Furthermore, instead of computing the invariant ideals of all permutations of a given set of inner loops and extending this set until a polynomial ideal as a fixed point is reached, we generate the polynomial invariant ideal of just one permutation iteratively until we reach the fixed point. As a result we have to perform less Gröbner basis computations in the process of invariant generation.

A data-driven approach to invariant generation is given in [20], where concrete program executions are used to generate invariant candidates. Machine learning is then used to infer polynomial invariants from the candidate ones. In our work we do not use invariant candidates. While the program flow in our programming model is more restricted then [20], to the best of our knowledge, none of the above cited methods can fully handle the polynomial arithmetic of extended P-solvable loops.

2 Preliminaries

2.1 Programming Model and Invariants

Let $\mathbb{K}$ be a computable field of characteristic zero. This means that addition and multiplication can be carried out algorithmically, that there exists an algorithm to test if an element in $\mathbb{K}$ is zero, and that the field of rational numbers $\mathbb{Q}$ is a subfield of $\mathbb{K}$. For variables $x_1, \ldots, x_n$, the ring of multivariate polynomials over $\mathbb{K}$ is denoted by $\mathbb{K}[x_1, \ldots, x_n]$, or, if the number of variables is clear from (or irrelevant in) the context, by $\mathbb{K}[\boldsymbol{x}]$. Correspondingly, $\mathbb{K}(x_1, \ldots, x_m)$ or $\mathbb{K}(\boldsymbol{x})$ denotes the field of rational functions over $\mathbb{K}$ in $x_1, \ldots, x_m$. If every polynomial in $\mathbb{K}[x]$ with a degree ≥ 1 has at least one root in $\mathbb{K}$, then $\mathbb{K}$ is called algebraically closed. An example for such a field is $\overline{\mathbb{Q}}$, the field of algebraic numbers. In contrast, the field of complex numbers $\mathbb{C}$ is algebraically closed, but not computable, and $\mathbb{Q}$ is computable, but not algebraically closed. We suppose that $\mathbb{K}$ is always algebraically closed. This is not necessary for our theory, as we only need the existence of roots for certain polynomials, which is achieved by choosing $\mathbb{K}$ to be an appropriate algebraic extension field of $\mathbb{Q}$. It does, however, greatly simplify the statement of our results.

In our framework, we consider a program B to be a loop of the form

$$\begin{array}{l} \textbf{while} \ldots \textbf{ do} \\ \quad B' \\ \textbf{end while} \end{array} \tag{1}$$

where B' is a program block that is either the empty block ϵ, an assignment $v_i = f(v_1, \ldots, v_m)$ for a rational function $f \in \mathbb{K}(x_1, \ldots, x_m)$ and program variables $v_1, \ldots, v_m$, or has one of the composite forms

sequential	inner loop	conditional
	while ... **do**	**if** ... **then**
$B_1; B_2$	B_1	B_1 **else** B_2
	end while	**end if**

for some program blocks B_1 and B_2 and the usual semantics. We omit conditions for the loop and if statements, as the problem of computing all polynomial invariants is undecidable when taking affine equality tests into account [11]. Consequently, we regard loops as non-deterministic programs in which each block of consecutive assignments can be executed arbitrarily often. More precisely, grouping consecutive assignments into blocks $B_1, \ldots, B_r$, any execution path of B can be written in the form

$$B_1^{n_1}; B_2^{n_2}; \ldots; B_r^{n_r}; B_1^{n_{r+1}}; B_2^{n_{r+2}}; \ldots$$

for a sequence $(n_i)_{i \in \mathbb{N}}$ of non-negative integers with finitely many non-zero elements. To that effect, we interpret any given program (1) as the set of its execution paths, written as

$$B = (B_1^*; B_2^*; \ldots; B_r^*)^*.$$

We adapt the well-established Hoare triple notation

$$\{P\}B\{Q\}, \tag{2}$$

for program specifications, where P and Q are logical formulas, called the pre- and postcondition respectively, and B is a program. In this paper we focus on partial correctness of programs, that is a Hoare triple (2) is correct if every terminating computation of B which starts in a state satisfying P terminates in a state that satisfies Q.

In this paper we are concerned with computing polynomial invariants for a considerable subset of loops of the form (1). These invariants are algebraic dependencies among the loop variables that hold after any number of loop iterations.

Definition 1. *A polynomial $p \in \mathbb{K}[x_1, \ldots, x_m]$ is a polynomial loop invariant for a loop $B = B_1^*; \ldots; B_r^*$ in the program variables $v_1, \ldots, v_m$ with initial values $v_1(0), \ldots, v_m(0)$, if for every sequence $(n_i)_{i \in \mathbb{N}}$ of non-negative integers with finitely many non-zero elements, the Hoare triple*

$$\{p(v_1, \ldots, v_m) = 0 \wedge \bigwedge_{i=0}^{m} v_i = v_i(0)\}$$
$$B_1^{n_1}; B_2^{n_2} \ldots, B_r^{n_r}; B_1^{n_{r+1}}; \ldots$$
$$\{p(v_1, \ldots, v_m) = 0\}$$

is correct.

2.2 Polynomial Rings and Ideals

Polynomial invariants are algebraic dependencies among the values of the variables at each loop iteration. Obviously, non-trivial dependencies do not always exist.

Definition 2. *Let $\mathbb{L}$ / $\mathbb{K}$ be a field extension. Then $a_1, \ldots, a_n \in \mathbb{L}$ are* algebraically dependent *over $\mathbb{K}$ if there exists a $p \in \mathbb{K}[x_1, \ldots, x_n] \setminus \{0\}$ such that $p(a_1, \ldots, a_n) = 0$. Otherwise they are called* algebraically independent*.*

In [8,17], it is observed that the set of all polynomial loop invariants for a given loop forms an ideal. It is this fact that facilitates all of our subsequent reasoning.

Definition 3. *A subset $\mathcal{I}$ of a commutative ring R is called an* ideal*, written $\mathcal{I} \lhd R$, if it satisfies the following three properties:*

1. $0 \in \mathcal{I}$.
2. *For all* $a, b \in \mathcal{I}$: $a + b \in \mathcal{I}$.
3. *For all* $a \in \mathcal{I}$ *and* $b \in R$: $a \cdot b \in \mathcal{I}$.

Definition 4. *Let $\mathcal{I} \lhd R$. Then $\mathcal{I}$ is called*

- proper *if it is not equal to R,*
- prime *if $a \cdot b \in \mathcal{I}$ implies $a \in \mathcal{I}$ or $b \in \mathcal{I}$, and*
- radical *if $a^n \in \mathcal{I}$ implies $a \in \mathcal{I}$.*

The height $\mathrm{hg}(\mathcal{I}) \in \mathbb{N}$ *of a prime ideal $\mathcal{I}$ is equal to n if n is the maximal length of all possible chains of prime ideals $\mathcal{I}_0 \subset \mathcal{I}_2 \subset \cdots \subset \mathcal{I}_n = \mathcal{I}$.*

Example 5. The set of even integers $2\mathbb{Z}$ is an ideal of $\mathbb{Z}$. In general $n\mathbb{Z}$ for a fixed integer n is an ideal of $\mathbb{Z}$. It is prime if and only if n is a prime number.

Polynomial ideals can informally be interpreted as the set of all consequences when it is known that certain polynomial equations hold. In fact, if we have given a set P of polynomials of which we know that they serve as algebraic dependencies among the variables of a given loop, the ideal generated by P then contains all the polynomials that consequently have to be polynomial invariants as well.

Definition 6. *A subset $B \subseteq \mathcal{I}$ of an ideal $\mathcal{I} \lhd R$ is called a* basis *for $\mathcal{I}$ if*

$$\mathcal{I} = \langle B \rangle := \{a_0 b_0 + \cdots + a_m b_m \mid m \in \mathbb{N}, a_0, \ldots, a_m \in R, b_0 \ldots, b_m \in B\}.$$

We say that B generates *$\mathcal{I}$.*

A basis for a given ideal in a ring does not necessarily have to be finite. However, a key result in commutative algebra makes sure that in our setting we only have to consider finitely generated ideals.

Theorem 7 (Hilbert's Basis Theorem – Special case). *Every ideal in* $\mathbb{K}[\boldsymbol{x}]$ *has a finite basis.*

Subsequently, whenever we say we are given an ideal $\mathcal{I}$, we mean that we have given a finite basis of $\mathcal{I}$.

There is usually more than one basis for a given ideal and some are more useful for certain purposes than others. In his seminal PhD thesis [1], Buchberger introduced the notion of Gröbner bases for polynomial ideals and an algorithm to compute them. While, for reasons of brevity, we will not formally define these bases, it is important to note that with their help, central questions concerning polynomial ideals can be answered algorithmically.

Theorem 8. *Let* $p \in \mathbb{K}[x_1, \ldots, x_n]$ *and* $\mathcal{I}, \mathcal{J} \lhd \mathbb{K}[x_1, \ldots, x_n]$. *There exist algorithms to decide the following problems.*

1. *Decide if* p *is an element of* $\mathcal{I}$.
2. *Compute a basis of* $\mathcal{I} + \mathcal{J}$.
3. *Compute a basis of* $\mathcal{I} \cap \mathcal{J}$.
4. *For* $\{\tilde{x}_1, \ldots \tilde{x}_m\} \subseteq \{x_1, \ldots, x_n\}$, *compute a basis of* $\mathcal{I} \cap \mathbb{K}[\tilde{x}_1, \ldots, \tilde{x}_m]$.
5. *Let* $q \in \mathbb{K}[\boldsymbol{x}]$. *Compute a basis for*

$$\mathcal{I} : \langle q \rangle^\infty := \{q \in \mathbb{K}[\boldsymbol{x}] \mid \exists n \in \mathbb{N} : q^n p \in \mathcal{I}\}.$$

The ideal $\mathcal{I} : \langle q \rangle^\infty$ *is called the* saturation *of* $\mathcal{I}$ *with respect to* q.

We will use Gröbner bases to compute the ideal of all algebraic relations among given rational functions. For this, we use the polynomials $q_i y_i - p_i$ to model the equations $y_i = q_i/p_i$ by multiplying the equation with the denominator. In order to model the fact that the denominator is not identically zero, and therefore allowing us to divide by it again, we use the saturation with respect to the least common multiple of all denominators. To see why this is necessary, consider $y_1 = y_2 = \frac{x_1}{x_2}$. An algebraic relation among y_1 and y_2 is $y_1 - y_2$, but with the polynomials $x_2 y_1 - x_1$ and $x_2 y_2 - x_1$, we only can derive $x_2(y_1 - y_2)$. We have to divide by x_2.

Theorem 9. *Let* $r_1, \ldots, r_m \in \mathbb{K}(\boldsymbol{x})$ *and let the numerator of* r_i *be given by* $p_i \in \mathbb{K}[\boldsymbol{x}]$ *and the denominator by* $q_i \in \mathbb{K}[\boldsymbol{x}]$. *The ideal of all polynomials* p *in* $\mathbb{K}[\boldsymbol{y}]$ *with* $p(r_1, \ldots, r_m) = 0$ *is given by*

$$\left(\sum_{i=1}^{m} \langle q_i y_i - p_i \rangle\right) : \langle \operatorname{lcm}(q_1, \ldots, q_m) \rangle^\infty \cap \mathbb{K}[\boldsymbol{y}],$$

where $\operatorname{lcm}(\ldots)$ *denotes the least common multiple.*

Proof. Write $d := \operatorname{lcm}(q_1, \ldots, q_m)$. The theorem can be easily verified from the fact that, for any given p with $p(r_1, \ldots, r_m) = 0$, there exists a $k \in \mathbb{N}$ such that $d^k p(r_1, \ldots, r_m) = 0$ is an algebraic relation for $p_1, \ldots, p_m$ (by clearing denominators in the equation $p(r_1, \ldots, r_m) = 0$). □

A polynomial ideal $\mathcal{I} \lhd \mathbb{K}[\boldsymbol{x}]$ gives rise to a set of points in $\mathbb{K}^n$ for which all polynomials in $\mathcal{I}$ vanish simultaneously. This set is called a *variety*.

Definition 10. *Let $\mathcal{I} \lhd \mathbb{K}[x_1, \ldots, x_n]$ be an ideal. The set*

$$V(\mathcal{I}) = \{(a_1, \ldots, a_n) \in \mathbb{K}^n \mid p(a_1, \ldots, a_n) = 0 \text{ for all } p \in \mathcal{I}\},$$

is the variety *defined by $\mathcal{I}$.*

Varieties are one of the central objects of study in algebraic geometry. Certain geometric shapes like points, lines, circles or balls can be described by prime ideals and come with an intuitive notion of a dimension, e.g. points have dimension zero, lines and circles have dimension one and balls have dimension two. The notion of the Krull dimension of a ring formalizes this intuition when being applied to the quotient ring $\mathbb{K}[\boldsymbol{x}]/\mathcal{I}$. In this paper, we will use the Krull dimension to provide an upper bound for the number of necessary iterations of our algorithm.

Definition 11. *The* Krull dimension *of a commutative ring R is the supremum of the lengths of all chains $\mathcal{I}_0 \subset \mathcal{I}_1 \subset \ldots$ of prime ideals.*

Theorem 12. *The Krull dimension of $\mathbb{K}[x_1, \ldots, x_n]$ is equal to n.*

3 Extended P-Solvable Loops

In [4] the class of *P-solvable* loops [9] was extended to so-called *extended P-solvable* loops. So far, this class captures loops with assignments only, i.e. loops without any nesting of conditionals and loops. In Section 3.3 we close this gap by introducing a new approach for computing invariants of multi-path loops which generalizes the algorithm proposed in [10]. Before dealing with multi-path loops, we recall the notion of extended P-solvable loops in Section 3.1 and showcase the invariant ideal computation.

3.1 Loops with assignments only

In this section, we restrain ourselves to loops whose bodies are comprised of rational function assignments only. This means that we restrict the valid composite forms in a program of the form (1) to sequential compositions and, for the moment, exclude inner loops and conditional branches. We therefore consider a loop $L = B_1^*$ where B_1 is a single block containing only variable assignments.

Each variable v_i in a given loop of the form (1) gives rise to a sequence $(v_i(n))_{n\in\mathbb{N}}$, where n is the number of loop iterations. The class of eligible loops is then defined based on the form of these sequences. Let $r(x)^{\underline{n}}$ denote the *falling factorial* defined as $\prod_{i=0}^{n-1} r(x-i)$ for any $r \in \mathbb{K}(x)$ and $n \in \mathbb{N}$.

Definition 13. *A loop with assignments only is called* extended P-solvable *if each of its recursively changed variables determines a sequence of the form*

$$v_i(n) = \sum_{j\in\mathbb{Z}^\ell} p_{i,j}(n, \theta_1^n, \ldots, \theta_k^n)((n+\zeta_1)^{\underline{n}})^{j_1} \cdots ((n+\zeta_\ell)^{\underline{n}})^{j_\ell} \tag{3}$$

where $k, \ell \in \mathbb{N}$, the $p_{i,j}$ are polynomials in $\mathbb{K}(x)[y_1, \ldots, y_k]$, not identically zero for finitely many $j \in \mathbb{Z}^\ell$, the θ_i are elements of $\mathbb{K}$ and the ζ_i are elements of $\mathbb{K} \setminus \mathbb{Z}^-$ with $\theta_i \neq \theta_j$ and $\zeta_i - \zeta_j \notin \mathbb{Z}$ for $i \neq j$.

Definition 13 extends the class of P-solvable loops in the sense that each sequence induced by an extended P-solvable loop is the sum of a finitely many hypergeometric sequences. This comprises C-finite sequences as well as hypergeometric sequences and sums and Hadamard products of C-finite and hypergeometric sequences. In contrast, P-solvable loops induce C-finite sequences only. For details on C-finite and hypergeometric sequences we refer to [5].

Every sequence of the form (3) can be written as

$$v_j^{(1)} = r_j(\boldsymbol{v}^{(0)}, \boldsymbol{\theta}, (n + \boldsymbol{\zeta})^{\underline{n}}, n)$$

where $r_j = p_i/q_i$ is a rational function, and $v^{(0)}$ and $v^{(1)}$ denote the values of v before and after the execution of the loop. Let $I(\boldsymbol{\theta}, \boldsymbol{\zeta}) \lhd \mathbb{K}[y_0, \ldots, y_{k+\ell}]$ be the ideal of all algebraic dependencies in the variables $y_0, \ldots, y_{k+\ell}$ between the sequence $(n)_{n\in\mathbb{N}}$, the exponential sequences $\theta_1^n, \ldots, \theta_k^n$ and the sequences $(n+\zeta_1)^{\underline{n}}, \ldots, (n+\zeta_\ell)^{\underline{n}}$. Note that it was shown in [4] that this ideal is the same as the extension of the ideal $I(\boldsymbol{\theta}) \lhd \mathbb{K}[y_0, \ldots, y_k]$ of all algebraic dependencies between the θ^n in $\mathbb{K}[y_0, \ldots, y_k]$ to $\mathbb{K}[y_0, \ldots, y_{k+\ell}]$, as the factorial sequences $(n+\zeta_i)^{\underline{n}}$ are algebraically independent from the exponential sequences θ_i^n. Now the following proposition states how the invariant ideal of an extended P-solvable loop can be computed.

Proposition 14 ([4]). *For an extended P-solvable loop with program variables $v_1, \ldots, v_m$ the invariant ideal is given by*

$$\left(\left(\sum_{j=1}^{m} \langle q_j(\boldsymbol{v}^{(0)}, \boldsymbol{y}) v_j^{(1)} - p_j(\boldsymbol{v}^{(0)}, \boldsymbol{y})\rangle\right) : \langle \operatorname{lcm}(q_1, \ldots, q_m)\rangle^\infty + I(\boldsymbol{\theta}, \boldsymbol{\zeta})\right) \cap \mathbb{K}[\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(0)}].$$

Example 15. Consider the following loop with relevant program variables a, b and c.

while true **do**
  $a := 2 \cdot (n+1)(n+\frac{3}{2}) \cdot a$
  $b := 4 \cdot (n+1) \cdot b$
  $c := \frac{1}{2} \cdot (n+\frac{3}{2}) \cdot c$
  $n := n + 1$
end while

The extracted recurrence relations admit the following system of closed form solutions:

$$a_n = 2^n \cdot a_0 \cdot (n)^{\underline{n}} \cdot (n + \frac{1}{2})^{\underline{n}},$$
$$b_n = 4^n \cdot b_0 \cdot (n)^{\underline{n}},$$
$$c_n = 2^{-n} \cdot c_0 \cdot (n + \frac{1}{2})^{\underline{n}}.$$

Since every closed form solution is of the form (3) we have an extended P-solvable loop, and we can apply Proposition 14 to compute the invariant ideal:

$$(\mathcal{I} + I(\boldsymbol{\theta}, \boldsymbol{\zeta})) \cap \mathbb{K}[a^{(1)}, b^{(1)}, c^{(1)}, a^{(0)}, b^{(0)}, c^{(0)}] = \langle b^{(1)} \cdot c^{(1)} \cdot a^{(0)} - a^{(1)} \cdot b^{(0)} \cdot c^{(0)} \rangle,$$

where

$$\begin{aligned} \mathcal{I} &= \langle a^{(1)} - y_1 \cdot a^{(0)} \cdot z_1 z_2, b^{(1)} - y_2 \cdot b^{(0)} \cdot z_1, c^{(1)} - y_3 \cdot c^{(0)} \cdot z_2 \rangle, \\ I(\boldsymbol{\theta}, \boldsymbol{\zeta}) &= \langle y_1^2 - y_2, y_1 y_3 - 1, y_2 y_3 - y_1 \rangle. \end{aligned}$$

The ideal $I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ in variables y_1, y_2, y_3 is the set of all algebraic dependencies among $2^n, 4^n$ and 2^{-n}, and $\mathcal{I}$ is generated by the closed form solutions where exponential and factorial sequences are replaced by variables y_1, y_2, y_3 and z_1, z_2.

3.2 Algebraic Dependencies of Composed Rational Functions with Side Conditions

In this section we give the prerequisites for proving termination of the invariant generation method for multi-path loops (Section 3.3). The results of this section will allow us to proof termination by applying Theorem 12.

Let $\boldsymbol{v}^{(i)} = v_1^{(i)}, \ldots, v_m^{(i)}$ and $\boldsymbol{y}^{(i)} = y_1^{(i)}, \ldots, y_\ell^{(i)}$ for $i \in \mathbb{N}$. We model the situation in which the value of the jth loop variable after the execution of the ith block in (1) is given by a rational function in the $\boldsymbol{y}^{(i)}$ (which, for us, will be the exponential and factorial sequences as well as the loop counter) and the 'old' variable values $\boldsymbol{v}^{(i-1)}$ and is assigned to $v_j^{(i)}$. Set $\mathcal{I}_0 = \sum_{j=1}^m \langle v_j^{(1)} - v_j^{(0)} \rangle$ and let $I_i \lhd \mathbb{K}[\boldsymbol{y}^{(i)}]$ for $i \in \mathbb{N}^*$. Furthermore, let $q_j^{(i)}, p_j^{(i)} \in \mathbb{K}[\boldsymbol{v}^{(i)}, \boldsymbol{y}^{(i)}]$ such that for fixed i there exists a $\boldsymbol{y} \in V(I_i)$ with $p_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}) / q_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}) = \boldsymbol{v}_j^{(i)}$ for all j and with $d_i := \operatorname{lcm}(q_1^{(i)}, \ldots, q_m^{(i)})$ we have $d_i \notin I_i$ and $d_i(\boldsymbol{v_i}, \boldsymbol{y}) = 1$. Set

$$J_i = \sum_{j=1}^m \langle q_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}^{(i)}) v_j^{(i+1)} - p_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}^{(i)}) \rangle.$$

Remark 16. The requirement for the existence of a point $\boldsymbol{y}$ in $V(I_i)$ such that $p_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}) / q_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}) = \boldsymbol{v}_j^{(i)}$ for all j and $d_i(\boldsymbol{v_i}, \boldsymbol{y}) = 1$ is always fulfilled in our context, as it is a formalization of the fact that the execution of a loop L^* also allows that it is executed zero times, meaning the values of the program variables do not change.

In order to develop some intuition about the following, consider a list of consecutive loops $L_1; L_2; L_3; \ldots$ where each of them is extended P-solvable. Intuitively, the ideals I_i then correspond to the ideal of algebraic dependencies among the exponential and factorial sequences occurring in L_i, whereas J_i stands for the ideal generated by the closed form solutions of L_i. Moreover, the variables $v_j^{(i+1)}$ correspond to the values of the loop variables after the execution of the loop L_i. The following iterative computation then allows us to generate the invariant ideal for $L_1; L_2; L_3; \ldots$

$$\mathcal{I}_i := ((J_i + \mathcal{I}_{i-1} + I_i) : \langle d_i \rangle^\infty) \cap \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}]$$

Now the remaining part of this section is devoted to proving properties of the ideals $\mathcal{I}_i$ which will help us to show that there exists an index k such that $\mathcal{I}_k = \mathcal{I}_{k'}$ for all $k' > k$ for a list of consecutive loops $L_1; \ldots; L_r; L_1; \ldots; L_r; \ldots$ with $r \in \mathbb{N}$.

First note that the ideal $\mathcal{I}_i$ can be rewritten as

$$\begin{aligned}\mathcal{I}_i = \{p \in \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}] \mid \exists q \in \mathcal{I}_{i-1}, k \in \mathbb{N} : \\ q \equiv d_i^k p(r_1^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}^{(i)}), \ldots, r_m^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}^{(i)}), \boldsymbol{v}^{(0)}) \pmod{I_i}\}. \end{aligned} \tag{4}$$

If I_i is radical, an equation $\bmod I_i$ is, informally speaking, the same as substituting $\boldsymbol{y}$ with values from $V(I_i)$, so (4) translates to

$$\begin{aligned}\mathcal{I}_i = \{p \in \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}] \mid \exists q \in \mathcal{I}_{i-1}, k \in \mathbb{N} : \\ \forall \boldsymbol{y} \in V(I_i) : q = d_i^k p(r_1^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}), \ldots, r_m^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}), \boldsymbol{v}^{(0)})\}. \end{aligned} \tag{5}$$

We now get the following subset relation between two consecutively computed ideals $\mathcal{I}_i$.

Lemma 17. *If I_i is radical, then $\mathcal{I}_i \subseteq \mathcal{I}_{i-1}|_{\boldsymbol{v}^{(i-1)} \leftarrow \boldsymbol{v}^{(i)}}$.*

Proof. Let $p \in \mathcal{I}_i$. We have to show that there is an $r \in \mathcal{I}_{i-2}$ and a $k \in \mathbb{N}$ such that

$$r \equiv d_{i-1}^k p(r_1^{(i-1)}(\boldsymbol{v}^{(i-1)}, \boldsymbol{y}^{(i-1)}), \ldots, r_m^{(i-1)}(\boldsymbol{v}^{(i-1)}, \boldsymbol{y}^{(i-1)}), \boldsymbol{v}^{(0)}) \pmod{I_{i-1}}.$$

Since I_i is radical, there is a $q \in \mathcal{I}_{i-1}$, a $z \in \mathbb{N}$, and a $\boldsymbol{y} \in V(I_i)$ with

$$q = d_i^z p(r_1^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}), \ldots, r_m^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{y}), \boldsymbol{v}^{(0)}) = p(\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(0)}).$$

Then, by Equation (4) for $\mathcal{I}_{i-1}$, there is an $r \in \mathcal{I}_{i-2}$ with the desired property. □

For prime ideals, we get an additional property:

Lemma 18. *If $\mathcal{I}_{i-1}$ and I_i are prime, then so is $\mathcal{I}_i$.*

Proof. Let $a \cdot b \in \mathcal{I}_i$ and denote by $a|_r$ and $b|_r$ the rational functions where each $v_j^{(i+1)}$ is substituted by $r_j^{(i)}$ in a, b respectively. Then there is a $q \in \mathcal{I}_{i-1}$ and a $k = k_1 + k_2 \in \mathbb{N}$ with $d_i^{k_1} a|_r, d_i^{k_2} b|_r \in \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}]$

$$q \equiv d_i^k (a \cdot b)|_r \equiv d_i^{k_1} a|_r \cdot d_i^{k_2} b|_r \pmod{I_i}$$

If $d_i^k a|_r$ is zero modulo I_i, then a is an element of $\mathcal{I}_i$, as $0 \in \mathcal{I}_{i-1}$. The same argument holds for b. Suppose that $d_i^{k_1} a|_r, d_i^{k_2} b|_r \not\equiv 0 \pmod{I_i}$. Then, since I_i is prime, $\mathbb{K}[\boldsymbol{y}^{(i)}]/I_i$ is an integral domain, and so it follows that $q \not\equiv 0 \pmod{I_i}$. Now, because $\mathcal{I}_{i-1}$ is prime, it follows without loss of generality that $d_i^{k_1} a|_r \in \mathcal{I}_{i-1}$, from which we get $a \in \mathcal{I}_i$. □

We now use Lemmas 17 and 18 to give details about the minimal decomposition of $\mathcal{I}_i$.

Proposition 19. *For fixed $i_0 \in \mathbb{N}$, let all I_i, $0 \leq i \leq i_0$ be radical and let $\mathcal{I}_{i_0} = \bigcap_{k=0}^{n} P_k$ be the minimal decomposition of $\mathcal{I}_{i_0}$. Then*

1. *for each k there exist prime ideals $I_{k,1}, I_{k,2}, \ldots$ such that P_k is equal to a $\mathcal{I}_{k,i_0}$ constructed as above with $J_1, \ldots, J_{i_0}$ and $I_{k,1}, \ldots, I_{k,i_0}$.*
2. *if I_{i_0+1} is radical and $\mathcal{I}_{i_0+1} = \bigcap_{j=0}^{n'} P'_j$ is the minimal decomposition of $\mathcal{I}_{i+1}$, then, for each P'_j there exists a P_k such that $P'_j \subseteq P_k|_{\boldsymbol{v}^{(i_0)} \leftarrow \boldsymbol{v}^{(i_0+1)}}$.*

Proof. We prove 1. by induction. For $i_0 = 0$, there is nothing to show. Now assume the claim holds for some $i_0 \in \mathbb{N}$ and let $I_{i_0+1} = \bigcap_{j=0}^{w} Q_j$ be the minimal decomposition of I_{i_0+1}. With this we get

$$\begin{aligned}
\mathcal{I}_{i_0+1} &= (J_{i_0+1} + \mathcal{I}_{i_0} + I_{i_0+1}) : \langle d_{i_0+1} \rangle^\infty \cap \mathbb{K}[\boldsymbol{v}^{(i_0+1)}, \boldsymbol{v}^{(0)}] \\
&= \left(\bigcap_{k=0}^{n} J_{i_0+1} + P_k + \bigcap_{j=0}^{w} Q_j \right) : \langle d_{i_0+1} \rangle^\infty \cap \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}] \\
&= \left(\bigcap_{k=0}^{n} \bigcap_{j=0}^{w} \underbrace{(J_{i_0+1} + P_k + Q_j) : \langle d_{i_0+1} \rangle^\infty \cap \mathbb{K}[\boldsymbol{v}^{(i_0+1)}, \boldsymbol{v}^{(0)}]}_{\tilde{I}_{k,j}} \right).
\end{aligned}$$

By the induction hypothesis, each P_k admits a construction as above, and thus so does $\tilde{I}_{k,j}$. By Lemma 18, $\tilde{I}_{k,j}$ is prime. This shows 1. The second claim then follows from the fact that the prime ideals in the minimal decomposition of $\mathcal{I}_{i_0+1}$ are obtained from the P_k via J_{i_0+1} and Q_j. Since the Q_j are prime, they are also radical, and the claim follows from Lemma 17. □

3.3 Loops with conditional branches

In this section, we extend the results of Section 3.1 to loops with conditional branches. Without loss of generality, we define our algorithm for a loop of the form

$$\textbf{while } \ldots \textbf{ do } L_1; L_2; \ldots; L_r \textbf{ end while}$$

where $L_i = B_i^*$ and B_i is a block containing variable assignments only.

Let $I(\boldsymbol{\theta}_i, \boldsymbol{\zeta}_i)$ denote the ideal of all algebraic dependencies as described in Section 3.1 for a inner loop L_i. As every inner loop provides its own loop counter, we have that the exponential and factorial sequences of distinct inner loops are algebraically independent. Therefore $I(\boldsymbol{\theta}, \boldsymbol{\zeta}) := \sum_{i=0}^{r} I(\boldsymbol{\theta}_i, \boldsymbol{\zeta}_i)$ denotes the set of all algebraic dependencies between exponential and factorial sequences among the inner loops $L_1, \ldots, L_r$.

Consider loop bodies $B_1, \ldots, B_r$ with common loop variables $v_1, \ldots, v_m$. Suppose the closed form of v_j in the ith loop body is given by a rational function in $m + k + \ell + 1$ variables:

$$v_j^{(i+1)} = r_j^{(i)}(\boldsymbol{v}^{(i)}, \boldsymbol{\theta}^n, (n + \boldsymbol{\zeta})^{\underline{n}}, n),$$

where $v_j^{(i)}$ and $v_j^{(i+1)}$ are variables for the value of v_j before and after the execution of the loop body. Then we can compute the ideal of all polynomial invariants of the non-deterministic program $(B_1^*; B_2^*; \ldots; B_r^*)^*$ with Algorithm 1.

Algorithm 1 Invariant generation via fixed point computation

Input: Loop bodies $B_1, \dots, B_r$ as described.
Output: The ideal of all polynomial invariants of $(B_1^*; B_2^*; \dots; B_r^*)^*$.

1: Compute $I := I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ as described above
2: $\mathcal{I}_{old} = \{0\}$, $\mathcal{I}_{new} = \sum_{j=1}^{m} \langle v_j^{(1)} - v_i^{(0)} \rangle$, $j = 0$
3: **WHILE** $\mathcal{I}_{old}|_{\boldsymbol{v}^{((j-1)\cdot r+1)} \leftarrow \boldsymbol{v}^{(j\cdot r+1)}} \neq \mathcal{I}_{new}$ **AND** $\mathcal{I}_{new} \neq \{0\}$ **DO**
4: $\quad \mathcal{I}_{old} \leftarrow \mathcal{I}_{new}$, $j \leftarrow j + 1$
5: $\quad$ **FOR** $i = 1, \dots, r$ **DO**
6: $\quad\quad \mathcal{I}_{new} \leftarrow (J_{i\cdot j} + \mathcal{I}_{old} + I) \cap \mathbb{K}[\boldsymbol{v}^{(i\cdot j+1)}, \boldsymbol{v}^{(0)}]$
7: **RETURN** $\mathcal{I}_{new}$

Lemma 20. *$I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ is a radical ideal.*

Proof. The elements of $I(\boldsymbol{\theta})$ represent C-finite sequences, i.e. sequences of the form

$$f_1(n)\theta_1^n + \dots + f_k^n \theta_k^n,$$

for univariate polynomials $f_1, \dots, f_k \in \mathbb{K}[y_0]$ and pairwise distinct $\theta_1, \dots, \theta_k \in \mathbb{K}$. The claim is then proven by the fact that the Hadamard-product $a^2(n, a(0))$ of a C-finite sequence $a(n, a(0))$ with itself is zero if and only if $a(n, a(0))$ is zero, and $I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ is the extension of $I(\boldsymbol{\theta})$ to $\mathbb{K}[y_0, \dots, y_{k+\ell}]$. □

Theorem 21. *Algorithm 1 is correct and terminates.*

Proof. The algorithm iteratively computes the ideals $\mathcal{I}_1, \mathcal{I}_2, \dots$ as in Section 3.2, so we will refer to I_{old} and I_{new} as $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$.

Termination: $\mathcal{I}_0$ is a prime ideal of height m. Suppose after an execution of the outer loop, the condition $\mathcal{I}_i|_{\boldsymbol{v}^{(i)} \leftarrow \boldsymbol{v}^{(i+1)}} \neq \mathcal{I}_{i+1}$ holds. As $I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ is radical by Lemma 20, we then get $\mathcal{I}_{i+1} \subset \mathcal{I}_i|_{\boldsymbol{v}^{(i)} \leftarrow \boldsymbol{v}^{(i+1)}}$ by Lemma 17. Thus there is a $p \in \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}]$ with $p \in \mathcal{I}_i|_{\boldsymbol{v}^{(i)} \leftarrow \boldsymbol{v}^{(i+1)}}$ and $p \notin \mathcal{I}_{i+1}$. Then, by Proposition 19, all prime ideals P_k in the minimal decomposition of $\mathcal{I}_{i+1}$ are have to be subsets of the prime ideals in the minimal decomposition of $\mathcal{I}_i|_{\boldsymbol{v}^{(i)} \leftarrow \boldsymbol{v}^{(i+1)}}$, where at least one of the subset relations is proper. Since $p \notin \mathcal{I}_{i+1}$, the height of at least one P_k has to be reduced. The height of each prime ideal is bounded by the height of $\mathcal{I}_0$.

Correctness: Let $i \in \mathbb{N}$ be fixed and denote by $I(B; i) \lhd \mathbb{K}[\boldsymbol{v}^{(i+1)}, \boldsymbol{v}^{(0)}]$ the ideal of all polynomial invariants for the non-deterministic program

$$(B_1^*; \dots; B_r^*)^{i/r}; B_1^*; \dots; B_{i \bmod r}^*.$$

It suffices to show that $\mathcal{I}_i$ is equal to $I(B; i)$. In fact, after i_0 iterations with $\mathcal{I}_{i_0} = \mathcal{I}_{i_0+1} = \mathcal{I}_{i_0+2} = \dots$, it follows that $\mathcal{I}_{i_0}$ is the ideal of polynomial invariants for $(B_1^*; \dots; B_r^*)^*$. Let $p \in I(B; i)$. The value of the program variable v_j in the program $B_1^*; \dots; B_{i \bmod r}^*$ is given as the value of a composition of the closed forms of each B_k:

$$v_j = p_j^{(i)}\left(p^{(i-1)}\Big(\dots\big(p^{(1)}(\boldsymbol{v}^{(0)}, \boldsymbol{s}_{n_1}), \dots\big), \boldsymbol{s}_{n_{i-1}}\Big), \boldsymbol{s}_{n_i}\right),$$

with $\boldsymbol{s}_n = n, \boldsymbol{\theta}^n, (n+\boldsymbol{\zeta})^{\underline{n}}$ and $n_1, \ldots, n_i \in \mathbb{N}$. The correctness then follows from the fact that that $\mathcal{I}_i$ is the ideal of all such compositions under the side condition that $(\boldsymbol{\theta}^n, (n+\boldsymbol{\zeta})^{\underline{n}}, n) \in V(I(\boldsymbol{\theta}, \boldsymbol{\zeta}))$ for any $n \in \mathbb{N}$. □

Revisiting the subset relations of the prime ideals in the minimal decomposition of $\mathcal{I}_0, \mathcal{I}_1, \ldots$ gives an upper bound for the necessary number of iterations in the algorithm.

Corollary 22. *Algorithm 1 terminates after at most m iterations of the while-loop at line 3.*

Proof. Suppose the algorithm terminates after k_0 iterations of the outer loop. We look at the ideals $\mathcal{I}_{r \cdot k}$, $k \in \{0, \ldots, k_0\}$. For a prime ideal P in the minimal decomposition of any $\mathcal{I}_{r \cdot (k+1)}$, there is a prime ideal Q in the minimal decomposition of $\mathcal{I}_{r \cdot k}$ such that $P \subseteq Q$. If $P = Q$, then P is a prime ideal in the minimal decomposition of each $\mathcal{I}_{r \cdot (k')}$, $k' > k$. This holds because there are only r many J_i. So if Q does not get replaced by smaller prime ideals in $\mathcal{I}_{r \cdot k+1}, \mathcal{I}_{r \cdot k+2} \ldots, \mathcal{I}_{r \cdot (k+1)}$, it has to be part of the minimal decomposition for any subsequent $\mathcal{I}_i$. From this it follows that, for each k, there is a prime ideal P_k in the minimal decomposition in $\mathcal{I}_{r \cdot k}$, such that $P_0 \supset P_1 \supset \cdots \supset P_{k_0}$ is a chain of proper superset relations, which then proves the claim since the height of $P_0 = \mathcal{I}_0$ is m. □

Example 23. Consider a multi-path loop L

$$\textbf{while } \ldots \textbf{ do } L_1; L_2 \textbf{ end while}$$

containing the following nested loops L_1 and L_2 and the corresponding closed form solutions:

while … **do**		**while** … **do**	
$a := a - b$	$a_n = a_0 - nb_0$	$b := b - a$	$b_m = b_0 - ma_0$
$p := p - q$	$p_n = p_0 - nq_0$	$q := q - p$	$q_m = q_0 - mp_0$
$r := r - s$	$r_n = r_0 - ns_0$	$s := s - r$	$s_m = s_0 - mr_0$
end while		**end while**	

For simplicity we chose inner loops without algebraic dependencies, i.e. I at line 1 will be the zero ideal and we therefore neglect it in the following computation. Moreover, we write a_i instead of $a^{(i)}$. We start with

$$\mathcal{I}_0 = \langle a_1 - a_0, b_1 - b_0, p_1 - p_0, q_1 - q_0, r_1 - r_0, s_1 - s_0 \rangle$$

followed by the first loop iteration:

$$\begin{aligned} \mathcal{I}_1 &= (J_1 + \mathcal{I}_0) \cap \mathbb{K}[a_0, b_0, p_0, q_0, r_0, s_0, a_2, b_2, p_2, q_2, r_2, s_2] \\ &= \langle b_0 - b_2, q_0 - q_2, s_0 - s_2, -p_0 s_2 + p_2 s_2 + q_2 r_0 - q_2 r_2, \\ &\qquad a_0 s_2 - a_2 s_2 - b_2 r_0 + b_2 r_2, a_0 q_2 - a_2 q_2 - b_2 p_0 + b_2 p_2 \rangle \end{aligned}$$

where

$$J_1 = \langle a_2 - a_1 + b_1 n, p_2 - p_1 + q_1 n, r_2 - r_1 + s_1 n, b_2 - b_1, q_2 - q_1, s_2 - s_1 \rangle$$

The following ideal $\mathcal{I}_2$ is then the invariant ideal for the first iteration of the outer loop L.

$$\begin{aligned}
\mathcal{I}_2 &= (J_2 + \mathcal{I}_1) \cap \mathbb{K}[a_0, b_0, p_0, q_0, r_0, s_0, a_3, b_3, p_3, q_3, r_3, s_3] \\
&= \langle -p_0 r_3 s_0 + p_3 r_3 s_3 + p_3 r_0 s_0 - p_3 r_0 s_3 - q_3 r_3^2 + q_3 r_0 r_3, \\
&\quad - p_3 s_0 + p_3 s_3 + q_0 r_3 - q_3 r_3, -p_0 s_0 + p_3 s_3 + q_0 r_0 - q_3 r_3, \\
&\quad a_3 s_0 - a_3 s_3 - b_0 r_3 + b_3 r_3, a_0 q_0 - a_3 q_3 - b_0 p_0 + b_3 p_3, \\
&\quad a_3 p_0 s_3 - a_3 p_3 s_3 - a_3 q_3 r_0 + a_3 q_3 r_3 - b_0 p_3 r_0 + b_3 p_3 r_0 + b_0 p_0 r_3 - b_3 p_0 r_3, \\
&\quad a_3 q_0 - a_3 q_3 - b_0 p_3 + b_3 p_3, a_0 s_0 - a_3 s_3 - b_0 r_0 + b_3 r_3, \\
&\quad - a_0 p_3 s_3 + a_3 p_3 s_3 + a_0 q_3 r_3 - a_3 q_3 r_3 + b_0 p_3 r_0 - b_0 p_0 r_3, \\
&\quad - a_3 b_0 r_0 + a_3 b_3 r_3 + a_0 b_0 r_3 - a_0 b_3 r_3 - a_3^2 s_3 + a_0 a_3 s_3, \\
&\quad - a_3 b_0 p_0 + a_3 b_3 p_3 + a_0 b_0 p_3 - a_0 b_3 p_3 - a_3^2 q_3 + a_0 a_3 q_3 \rangle
\end{aligned}$$

where

$$J_2 = \langle b_3 - b_2 + a_2 m, q_3 - q_2 + p_2 m, s_3 - s_2 + r_2 m, a_3 - a_2, p_3 - p_2, r_3 - r_2 \rangle$$

By continuing this computation we get the following ideals $\mathcal{I}_4$ and $\mathcal{I}_6$ which are the invariant ideals after two and three iterations of the outer loop L respectively.

$$\begin{aligned}
\mathcal{I}_4 = \langle & p_0 s_0 - p_5 s_5 - r_0 q_0 + r_5 q_5, \\
& b_5 p_5 - b_0 p_0 + a_0 q_0 - a_5 q_5, \\
& b_5 r_5 - b_0 r_0 + a_0 s_0 - a_5 s_5, \\
& b_5(-p_5 s_0 + r_5 q_0) + b_0(p_5 s_5 - r_5 q_5) + a_5(-s_5 q_0 + s_0 q_5), \\
& b_5(-p_5 r_0 + p_0 r_5) + a_5(-p_0 s_5 + r_0 q_5) + a_0(p_5 s_5 - r_5 q_5), \\
& b_0 p_0(-p_5 s_5 + r_5 q_5) + b_5(p_5^2 s_5 - p_0 r_5 q_0 + p_5(r_0 q_0 - r_5 q_5)) + \\
& \quad a_5(p_0 s_5 q_0 + q_5(-p_5 s_5 - r_0 q_0 + r_5 q_5)) \rangle
\end{aligned}$$

$$\begin{aligned}
\mathcal{I}_6 = \langle & p_0 s_0 - p_7 s_7 - r_0 q_0 + r_7 q_7, \\
& b_7 p_7 - b_0 p_0 + a_0 q_0 - a_7 q_7, \\
& b_7 r_7 - b_0 r_0 + a_0 s_0 - a_7 s_7, \\
& b_7(-p_7 s_0 + r_7 q_0) + b_0(p_7 s_7 - r_7 q_7) + a_7(-s_7 q_0 + s_0 q_7), \\
& b_7(-p_7 r_0 + p_0 r_7) + a_7(-p_0 s_7 + r_0 q_7) + a_0(p_7 s_7 - r_7 q_7), \\
& b_0 p_0(-p_7 s_7 + r_7 q_7) + b_7(p_7^2 s_7 - p_0 r_7 q_0 + p_7(r_0 q_0 - r_7 q_7)) + \\
& \quad a_7(p_0 s_7 q_0 + q_7(-p_7 s_7 - r_0 q_0 + r_7 q_7)) \rangle
\end{aligned}$$

Note that we now reached the fixed point as $\mathcal{I}_6 = \mathcal{I}_4|_{\boldsymbol{v}^{(5)} \leftarrow \boldsymbol{v}^{(7)}}$.

Corollary 22 provides a bound on the number of iterations in Algorithm 1. Therefore, we know at which stage we have to reach the fixed point of the computation at the latest, viz. after computing $\mathcal{I}_{r \cdot m}$. This fact allows us to construct a new algorithm which computes the ideal $\mathcal{I}_{r \cdot m}$ directly instead of doing a fixed point computation. The benefit of Algorithm 2 is that we have

to perform only one Gröbner basis computation in the end, although the new algorithm might performs more iterations than Algorithm 1.

Algorithm 2 Invariant generation without fixed point computation

Input: Loop bodies $B_1, \ldots, B_r$ as described.
Output: The ideal of all polynomial invariants of $(B_1^*; B_2^*; \ldots; B_r^*)^*$.

1: Compute $I := I(\boldsymbol{\theta}, \boldsymbol{\zeta})$ as described above
2: $\mathcal{I}_{new} = \sum_{j=1}^{m} \langle v_j^{(1)} - v_i^{(0)} \rangle + I$
3: **FOR** $j = 1, \ldots, m$ **DO**
4: **FOR** $i = 1, \ldots, r$ **DO**
5: $\mathcal{I}_{new} \leftarrow (J_{i \cdot j} + \mathcal{I}_{new})$
6: **RETURN** $\mathcal{I}_{new} \cap \mathbb{K}[\boldsymbol{v}^{(m \cdot r+1)}, \boldsymbol{v}^{(0)}]$

The proof of termination of the invariant generation method of [10] assumes that the ideal of algebraic dependencies is prime. In general, this does not hold. Consider the following loop and its closed forms with exponential sequences 2^n and $(-2)^n$:

while ... **do**	
$x := 2x$	$x(n) = 2^n \cdot x(0)$
$y := -2y$	$y(n) = (-2)^n \cdot y(0)$
end while	

The ideal of algebraic dependencies among the before-mentioned exponential sequences is given by $\langle a^2 - b^2 \rangle$ which is obviously not prime. As a consequence, the termination proof of [10] is incorrect. This paper closes this gap by providing a new algorithm and a corresponding termination proof.

4 Implementation and Experiments

We implemented our method in the Mathematica package ALIGATOR[1]. ALIGATOR is open source and available at:

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

Comparison of generated invariants. Based on the examples in Figure 1 we show that our technique can infer invariants which cannot be found by other state-of-the-art approaches. Our observations indicate that our method is superior to existing approaches if the loop under consideration has some *mathematical meaning* like division or factorization algorithms as depicted in Figure 1, whereas the approach of [7] has advantages when it comes to programs with complex flow.

[1] ALIGATOR requires the Mathematica packages Hyper [14], Dependencies [6] and FastZeil [13], where the latter two are part of the compilation package ErgoSum [15].

The techniques of [2] and [7] were implemented in tools called FASTIND[2] and DUET[3] respectively. Unlike ALIGATOR and FASTIND, DUET is not a pure inference engine for polynomial invariants, instead it tries to prove user-specified safety assertions. In order to check which invariants can be generated by DUET, we therefore asserted the invariants computed by ALIGATOR and checked if DUET can prove them.

```
while a ≠ b do
  if a > b then
    a := a − b
    p := p − q
    r := r − s
  else
    b := b − a
    q := q − p
    s := s − r
  end if
end while
```

(a)

```
while r ≠ 0 do
  if r > 0 then
    r := r − v
    v := v + 2
  else
    r := r + u
    u := u + 2
  end if
end while
```

(b)

```
while d ≥ E do
  if P < a + b then
    b := b/2
    d := d/2
  else
    a := a + b
    y := y + d/2
    b := b/2
    d := d/2
  end if
end while
```

(c)

Fig. 1: Three examples: (a) Extended Euclidean algorithm, (b) a variant of Fermat's factorization algorithm and (c) Wensley's algorithm for real division.

Let us consider the loop depicted in Figure 1a. Since we treat conditional branches as inner loops, we have that the invariants for this loop are the same as for the loop in Example 23. By instantiating the generated invariants with the following initial values on the left we get the following polynomial invariants on the right:

$$\begin{aligned}
a_0 &\mapsto x & 1 + qr - ps && (I_1)\\
b_0 &\mapsto y & bp - aq - y && (I_2)\\
p_0 &\mapsto 1 & br - as + x && (I_3)\\
q_0 &\mapsto 0 & -bp + aq - qry + psy && (I_4)\\
r_0 &\mapsto 0 & br - as - qrx + psx && (I_5)\\
s_0 &\mapsto 1 & (qr - ps)(-bp + aq + y) && (I_6)
\end{aligned}$$

Note that (I_4)-(I_6) are just linear combinations of (I_1)-(I_3). However, FASTIND was able to infer (I_1)-(I_3), whereas DUET was only able to prove (I_2), (I_5) and (I_6).

Other examples where ALIGATOR is superior in terms of the number of inferred invariants are given by the loops in Figures 1b and 1c. For Fermat's

[2] Available at `http://www.irisa.fr/celtique/ext/polyinv/`
[3] Available at `https://github.com/zkincaid/duet`

algorithm (Figure 1b) and the following initial values, ALIGATOR found one invariant, which was also found by FASTIND. However, DUET was not able to prove it.

$$\begin{aligned} &u_0 \mapsto 2R+1 \\ &v_0 \mapsto 1 \\ &r_0 \mapsto RR - N \end{aligned} \qquad -4N - 4r - 2u + u^2 + 2v - v^2 \qquad (I_7)$$

In case of Wensley's algorithm (Figure 1c) ALIGATOR was able to identify the following three invariants. FASTIND inferred the first two invariants, whereas DUET could not prove any of them.

$$\begin{aligned} &a_0 \mapsto 0 \\ &b_0 \mapsto Q/2 \\ &d_0 \mapsto 1 \\ &y_0 \mapsto 0 \end{aligned} \qquad \begin{aligned} &2b - dQ && (I_8) \\ &ad - 2by && (I_9) \\ &a - Qy && (I_{10}) \end{aligned}$$

Benchmarks and Evaluation. For the experimental evaluation of our approach, we used the following set of examples: (i) 18 programs taken from [2]; (ii) 4 new programs of extended P-solvable loops that were created by us. All examples are available at the repository of ALIGATOR.

Our experiments were performed on a machine with a 2.9 GHz Intel Core i5 and 16 GB LPDDR3 RAM; for each example, a timeout of 300 seconds was set. When using ALIGATOR, the Gröbner basis of the invariant ideal computed by ALIGATOR was non-empty for each example; that is, for each example we were able to find non-trivial invariants.

We evaluated ALIGATOR against FASTIND. As DUET is not a pure inference engine for polynomial invariants, we did not include it in the following evaluation. When compared to [2], we note that we do not fix the degree of the polynomial invariants to be generated. Moreover, our method is complete. That is, whenever ALIGATOR terminates, the basis of the polynomial invariant ideal is inferred; any other polynomial invariant is a linear combination of the basis polynomials.

Table 1a summarizes our experimental results on single-path loops, whereas Table 1b reports on the results from multi-path programs. The first column of each table lists the name of the benchmark. The second and third columns of Table 1a report, on the timing results of ALIGATOR and FASTIND, respectively. In Table 1b, the second column lists the number of branches (paths) of the multi-path loop, whereas the third column gives the number of variables used in the program. The fourth column reports on the number of iterations until the fixed point is reached by ALIGATOR, and hence terminates. The fifth and sixth columns, labeled AL1 and AL2, show the performance of ALIGATOR when using Algorithm 1 or Algorithm 2, respectively. The last column of Table 1b lists the results obtained by FASTIND. In both tables, timeouts are denoted by

[4] Testing the Maple implementation was not possible due to constraints regarding the Maple version.

Table 1: Experimental evaluation of ALIGATOR.

(a)

Single-path	ALIGATOR	FASTIND
`cohencu`	0.072	0.043
`freire1`	0.016	0.041
`freire2`	0.062	0.048
`petter1`	0.015	0.040
`petter2`	0.026	0.042
`petter3`	0.035	0.051
`petter4`	0.042	0.104
`petter5`	0.053	0.261
`petter20`	48.290	9.816
`petter22`	247.820	9.882
`petter23`	*TO*	9.853

(b)

Multi-path	$\#b$	$\#v$	$\#i$	AL1	AL2	FASTIND
`divbin`	2	3	2	0.134	45.948	0.045
`euclidex`	2	6	3	0.433	*TO*	0.049
`fermat`	2	3	2	0.045	0.060	0.043
`knuth`	4	5	2	55.791	*TO*	1.025
`lcm`	2	4	3	0.051	87.752	0.043
`mannadiv`	2	3	2	0.022	0.025	0.048
`wensley`	2	4	2	0.124	41.851	*err*
`extpsolv2`	2	3	2	0.192	*TO*	*err*
`extpsolv3`	3	3	2	0.295	*TO*	*err*
`extpsolv4`	4	3	2	0.365	*TO*	*err*
`extpsolv10`	10	3	2	0.951	*TO*	*err*

$\#b, \#v$ ... number of branches, variables
$\#i$ ... number of iterations until fixed point reached
AL1 ... ALIGATOR with Algorithm 1 (timeout $300s$)
AL2 ... ALIGATOR with Algorithm 2 (timeout $100s$)
FASTIND ... OCaml version of the tool in [2][4]
TO, *err* ... timeout, error

TO, whereas errors, due to the fact that the tool cannot be evaluated on the respective example, are given as *err*.

The results reported in Tables 1a and 1b show the efficiency of ALIGATOR: in 14 out of 18 examples, ALIGATOR performed significantly better than FASTIND. For the examples `petter20`, `petter22` and `petter23`, the time-consuming part in ALIGATOR comes from recurrence solving (computing the closed form of the recurrence), and not from the Gröbner basis computation. We intend to improve this part of ALIGATOR in the future. The examples `extpsolv2`, `extpsolv3`, `extpsolv4` and `extpsolv10` are extended P-solvable loops with respectively 2, 3, 4, and 10 nested conditional branches. The polynomial arithmetic of these examples is not supported by FASTIND. The results of ALIGATOR on these examples indicate that extended P-solvable loops do not increase the complexity of computing the invariant ideal.

We also compared the performance of ALIGATOR with Algorithm 1 against Algorithm 2. As shown in columns 5 and 6 of Table 1b, Algorithm 2 is not as efficient as Algorithm 1, even though Algorithm 2 uses only a single Gröbner basis computation. We conjecture that this is due to the increased number of variables in the polynomial system which influences the Gröbner basis computation. We therefore conclude that several small Gröbner basis computations (with fewer variables) perform better than a single large one.

5 Conclusions

We proposed a new algorithm for computing the ideal of all polynomial invariants for the class of extended P-solvable multi-path loops. The new approach computes the invariant ideal for a non-deterministic program $(L_1; \ldots; L_r)^*$ where the L_i are single-path loops. As a consequence, the proposed method can handle loops containing (i) an arbitrary nesting of conditionals, as these conditional branches can be transformed into a sequence of single-path loops by introducing flags, and (ii) one level of nested single-path loops.

Our method computes the ideals $\mathcal{I}_1, \mathcal{I}_2, \ldots$ until a fixed point is reached where $\mathcal{I}_i$ denotes the invariant ideal of $(L_1; \ldots; L_r)^i$. This fixed point is then a basis for the ideal containing all polynomial invariants for the extended P-solvable loop. We showed that this fixed point computation is guaranteed to terminate which implies the completeness of our method. Furthermore, we gave a bound on the number of iterations we have to perform to reach the fixed point. The proven bound is given by m iterations where m is the number of loop variables.

We showed that our method can generate invariants which cannot be inferred by other state-of-the-art techniques. In addition, we showcased the efficiency of our approach by comparing our Mathematica package ALIGATOR with state-of-the-art tools in invariant generation.

Future research directions include the incorporation of the loop condition into our method. So far we operate on an abstraction of the loop where we ignore the loop condition and treat the loop as a non-deterministic program. By doing so we might loose valuable information about the control flow of the program. By employing $\Pi\Sigma^*$-theory [19] it might be possible to extend our work also to loops containing arbitrary nesting of inner loops, which reflects another focus for further research.

Acknowledgments. We want to thank the anonymous reviewers for their helpful comments and remarks.

References

1. Buchberger, B.: An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. J. Symbolic Computation 41(3-4), 475–511 (2006)
2. Cachera, D., Jensen, T.P., Jobin, A., Kirchner, F.: Inference of Polynomial Invariants for Imperative Programs: A Farewell to Gröbner Bases. In: Miné, A., Schmidt, D. (eds.) Static Analysis - 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7460, pp. 58–74. Springer (2012)
3. Farzan, A., Kincaid, Z.: Compositional recurrence analysis. In: Proc. of FMCAD. pp. 57–64. FMCAD Inc, Austin, TX (2015)
4. Humenberger, A., Jaroschek, M., Kovács, L.: Automated Generation of Non-Linear Loop Invariants Utilizing Hypergeometric Sequences. In: Proceedings of the 2017

ACM on International Symposium on Symbolic and Algebraic Computation. pp. 221–228. ISSAC '17, ACM, New York, NY, USA (2017)
5. Kauers, M., Paule, P.: The Concrete Tetrahedron. Text and Monographs in Symbolic Computation, Springer Wien, 1st edn. (2011)
6. Kauers, M., Zimmermann, B.: Computing the algebraic relations of C-finite sequences and multisequences. Journal of Symbolic Computation 43(11), 787 – 803 (2008)
7. Kincaid, Z., Cyphert, J., Breck, J., Reps, T.: Non-Linear Reasoning For Invariant Synthesis. In: POPL (2018), to appear
8. Kovács, L.: Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema. Ph.D. thesis, RISC, Johannes Kepler University Linz (October 2007)
9. Kovács, L.: Reasoning Algebraically About P-Solvable Loops. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. pp. 249–264 (2008)
10. Kovács, L.: A Complete Invariant Generation Approach for P-solvable Loops. In: Perspectives of Systems Informatics, 7th International Andrei Ershov Memorial Conference, PSI 2009, Novosibirsk, Russia, June 15-19, 2009. Revised Papers. pp. 242–256 (2009)
11. Müller-Olm, M., Seidl, H.: A Note on Karr's Algorithm. In: Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings. pp. 1016–1028 (2004)
12. de Oliveira, S., Bensalem, S., Prevosto, V.: Polynomial invariants by linear algebra. In: Artho, C., Legay, A., Peled, D. (eds.) Proc. of ATVA. pp. 479–494. Springer (2016)
13. Paule, P., Schorn, M.: A Mathematica Version of Zeilbergers Algorithm for Proving Binomial Coefficient Identities. Journal of Symbolic Computation 20, 673 – 698 (1995)
14. Petkovšek, M.: Mathematic package hyper (1998), `http://www.fmf.uni-lj.si/~petkovsek/`
15. Research Institute for Symbolic Computation.: Mathematic Package ErgoSum (2016), `http://www.risc.jku.at/research/combinat/software/ergosum/`
16. Rodriguez-Carbonell, E., Kapur, D.: Automatic Generation of Polynomial Invariants of Bounded Degree using Abstract Interpretation. J. Science of Computer Programming 64(1), 54–75 (2007)
17. Rodríguez-Carbonell, E., Kapur, D.: Generating all polynomial invariants in simple loops. Journal of Symbolic Computation 42(4), 443 – 476 (2007)
18. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using gröbner bases. In: Proc. of POPL. pp. 318–329. ACM, New York, NY, USA (2004)
19. Schneider, C.: Summation theory ii: Characterizations of $r\pi\sigma$-extensions and algorithmic aspects. J. Symb. Comput. 80(3), 616–664 (2017), arXiv:1603.04285 [cs.SC]
20. Sharma, R., Gupta, S., Hariharan, B., Aiken, A., Liang, P., Nori, A.V.: A Data Driven Approach for Algebraic Loop Invariants. In: Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7792, pp. 574–592. Springer (2013)