

# Strong Invariants Are Hard

On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs

JULIAN MÜLLNER, TU Wien, Austria

MARCEL MOOSBRUGGER, TU Wien, Austria

LAURA KOVÁCS, TU Wien, Austria

We show that computing the strongest polynomial invariant for single-path loops with polynomial assignments is at least as hard as the SKOLEM problem, a famous problem whose decidability has been open for almost a century. While the strongest polynomial invariants are computable for *affine loops*, for polynomial loops the problem remained wide open. As an intermediate result of independent interest, we prove that reachability for discrete polynomial dynamical systems is SKOLEM-hard as well. Furthermore, we generalize the notion of invariant ideals and introduce *moment invariant ideals* for probabilistic programs. With this tool, we further show that the strongest polynomial moment invariant is (i) uncomputable, for probabilistic loops with branching statements, and (ii) SKOLEM-hard to compute for polynomial probabilistic loops without branching statements. Finally, we identify a class of probabilistic loops for which the strongest polynomial moment invariant is computable and provide an algorithm for it.

CCS Concepts: • **Theory of computation** → **Invariants; Probabilistic computation; Computability; Random walks and Markov chains.**

Additional Key Words and Phrases: Strongest algebraic invariant, Point-To-Point reachability, Skolem problem, Probabilistic programs

## 1 INTRODUCTION

Loop invariants describe valid program properties that hold before and after every loop iteration. Intuitively, invariants provide correctness information that may prevent programmers from introducing errors while making changes to the loop. As such, invariants are fundamental to formalizing program semantics as well as to automate the formal analysis and verification of programs. While automatically synthesizing loop invariants is, in general, an uncomputable problem, when considering only single-path loops with linear updates (linear loops), the strongest polynomial invariant is in fact computable [Hrushovski et al. 2018; Karr 1976; Kovács 2008; Müller-Olm and Seidl 2004a]. Yet, already for loops with “only” polynomial updates, computing the strongest invariant has been an open challenge since 2004 [Müller-Olm and Seidl 2004b]. In this paper, we bridge the gap between the computability result for linear loops and the uncomputability result for general loops by providing, to the best of our knowledge, the *first hardness result for computing the strongest polynomial invariant of polynomial loops*.

**Problem setting.** Let us motivate our hardness results using the two loops in Figure 1, showcasing that very small changes in loop arithmetic may significantly increase the difficulty of computing the strongest invariants. Figure 1a depicts an affine loop, that is, a loop where all updates are affine combinations of program variables. On the other hand, Figure 1b shows a polynomial loop whose updates are polynomials in program variables.

Authors’ addresses: Julian Müllner, TU Wien, Vienna, Austria, julian.muellner@tuwien.ac.at; Marcel Moosbrugger, TU Wien, Vienna, Austria, marcel.moosbrugger@tuwien.ac.at; Laura Kovács, TU Wien, Vienna, Austria, laura.kovacs@tuwien.ac.at.

```

 $[f \ u \ v \ w] \leftarrow [1 \ -1 \ 2 \ 0]$ 
while  $\star$  do
   $t \leftarrow 3t + 2u - 5w$ 
   $u \leftarrow u + 3w$ 
   $v \leftarrow 4u + 3v + w$ 
   $w \leftarrow t + u + 2v$ 
end while

```

(a) An *affine* loop from [Karimov et al. 2022].

```

 $\begin{bmatrix} x & y \end{bmatrix} \leftarrow \begin{bmatrix} x_0 & y_0 \end{bmatrix}$ 
while  $\star$  do
   $\begin{bmatrix} x \\ y \end{bmatrix} \leftarrow \begin{bmatrix} x + y \cdot \Delta_t \\ y + (y \cdot (1 - x^2) - x) \cdot \Delta_t \end{bmatrix}$ 
end while

```

(b) A *polynomial* loop, modelling the discrete-time Van der Pol oscillator [Dreossi et al. 2017] for some constant sampling time  $\Delta_t$ .

Fig. 1. Two examples of deterministic programs.

An affine (polynomial) **invariant** is a conjunction of affine (polynomial) equalities **holding before and after every loop iteration**. The computability of both the strongest affine and polynomial invariant has been studied extensively. For single-path affine loops, the seminal paper [Karr 1976] shows that the strongest affine invariant is computable, whereas [Kovács 2008] proves computability of the strongest polynomial invariant. Regarding single-path polynomial programs, for example the one in Figure 1b, [Müller-Olm and Seidl 2004a] gives an algorithm to compute all polynomial invariants of bounded degree.

Based on these results, the strongest polynomial invariant of Figure 1a is thus computable. Yet, the more general problem of computing the strongest polynomial invariant for *polynomial loops* without any restriction on the degree remained an open challenge since 2004 [Müller-Olm and Seidl 2004b]. In this paper, we address this challenge, which we coin as the **SPINV** problem and define below.

The **SPINV** Problem: Given a single-path loop with polynomial updates, compute the strongest polynomial invariant.

In Section 4, we prove that **SPINV** is *very hard*, essentially “defending” the state-of-the-art that so far failed to derive computational bounds on computing the strongest polynomial invariants of polynomial loops. The crux of our work is based on the **SKOLEM** problem, a prominent algebraic problem in the theory of linear recurrences [Everest et al. 2003; Tao 2008], which we briefly recall below and refer to Section 2.3 for details.

The **SKOLEM** Problem [Everest et al. 2003; Tao 2008]: Does a given linear recurrence sequence with constant coefficients have a zero?

The decidability of the **SKOLEM** problem has been open for almost a century, and its decidability would yield far-fetching consequences in number theory [Bilu et al. 2022; Lipton et al. 2022]. In Section 4, we show that **SPINV** is at least as hard as the **SKOLEM** problem, providing thus a computational lower bound showcasing the hardness of **SPINV**.

To the best of our knowledge, our results from Section 4 are the first lower bounds for **SPINV** and provide an answer to the open challenge posed by [Müller-Olm and Seidl 2004a]. While [Hrushovski et al. 2023] proved that the strongest polynomial invariant is uncomputable for multi-path polynomial programs, the computability of **SPINV** has been left open for future work. With our results proving that **SPINV** is **SKOLEM**-hard (Theorem 4.2), we show that the missing computability proof of **SPINV** is not surprising: solving **SPINV** is really hard.

**Connecting invariant synthesis and reachability.** A computational gap also exists in the realm of model-checking between affine and polynomial programs, similar to the computability of  $\text{SPInv}$ . Point-to-point reachability is arguably the simplest model-checking property; it asks whether a program can reach a given target state from a given initial state. For example, one may start the Van der Pol oscillator from Figure 1b in some initial configuration  $(x_0, y_0)$  and certify that it will eventually reach a certain target configuration  $(x_t, y_t)$ . Reachability, and even more involved model-checking properties, are known to be decidable for affine loops [Karimov et al. 2022]. However, the decidability or mere reachability of polynomial loops remains unknown without any existing non-trivial lower bounds. We refer to this reachability quest via the  $\text{P2P}$  problem.

The **Point-To-Point Reachability Problem ( $\text{P2P}$ )** : Given a single-path loop with polynomial updates, is a given target state reachable starting from a given initial state?

In Section 3, we resolve the lack of computational results on reachability in polynomial loops. In particular, we show that  $\text{P2P}$  is  $\text{SKOLEM-hard}$  (Theorem 3.3) as well. To the best of our knowledge, this yields the first non-trivial hardness result for  $\text{P2P}$ . In Section 4, we further show that  $\text{P2P}$  and  $\text{SPInv}$  are connected in the sense that  $\text{P2P}$  reduces to  $\text{SPInv}$ . That is,  $\text{SPInv}$  is at least as hard as  $\text{P2P}$ . Therefore, our reduction chain  $\text{SKOLEM} \leq \text{P2P} \leq \text{SPInv}$  implies that the decidability of  $\text{P2P}$  and/or  $\text{SPInv}$  would immediately solve the  $\text{SKOLEM}$  problem and longstanding conjectures in number theory.

**Beyond (non)deterministic loops and invariants.** In addition to computational limits within standard, (non)deterministic programs, we further establish computational (hardness) bounds in probabilistic loops. Probabilistic programs model stochastic processes and encode uncertainty information in standard control flow, used for example in cryptography [Barthe et al. 2012a], privacy [Barthe et al. 2012b], cyber-physical systems [Kofnov et al. 2022], and machine learning [Ghahramani 2015].

Because classical invariants, as in  $\text{SPInv}$ , do not account for probabilistic information, we provide a proper generalization of the strongest polynomial invariant for probabilistic loops in Section 5 (Lemma 5.5). With this generalization, we transfer the  $\text{SPInv}$  problem to the probabilistic setting. We hence consider the probabilistic version of  $\text{SPInv}$  as being the  $\text{Prob-SPInv}$  problem.

The **Prob-SPInv Problem**: Given a probabilistic loop with polynomial updates, compute the “probabilistic analog” of the strongest polynomial invariant.

In Section 5 we prove that  $\text{Prob-SPInv}$  inherits  $\text{SKOLEM-hardness}$  from its classical  $\text{SPInv}$  analog (Theorem 5.10). We also show that enriching the probabilistic program model with guards or branching statements renders the strongest polynomial (probabilistic) invariant uncomputable, even in the affine case (Theorems 5.8). We nevertheless provide a decision procedure when considering  $\text{Prob-SPInv}$  for a restricted class of polynomial loops: we define the class of *moment-computable* (polynomial) loops and show that  $\text{Prob-SPInv}$  is computable for such loops (Algorithm 1). Despite being restrictive, our moment-computable loops subsume affine loops with constant probabilistic choice. As such, Section 5 shows the limits of computability in deriving the strongest polynomial (probabilistic) invariants for probabilistic polynomial loops.

**Our contributions.** In conclusion, the main contributions of our work are as follows:

- In Section 3, we provide a reduction from  $\text{SKOLEM}$  to point-to-point reachability for polynomial loops, proving that  $\text{P2P}$  is  $\text{SKOLEM-hard}$  (Theorem 3.3).

- Section 4 gives a reduction from **P2P** to the problem of computing the strongest polynomial invariant of polynomial loops, establishing the connection between **P2P** and **SPINV**. As such, we prove that **SPINV** is **SKOLEM**-hard (Theorem 4.2).
- In Section 5, we generalize the concept of strongest polynomial invariants to the probabilistic setting (Lemma 5.5). We show that **PROB-SPINV** is **SKOLEM**-hard (Theorem 5.10) and uncomputable for general polynomial probabilistic programs (Theorem 5.8), but it becomes computable for moment-computable polynomial probabilistic programs (Algorithm 1).

## 2 PRELIMINARIES

Throughout the paper, we write  $\mathbb{N}$  for the natural numbers,  $\mathbb{Q}$  for the rationals,  $\mathbb{R}$  for the reals, and  $\overline{\mathbb{Q}}$  for the algebraic numbers. We denote by  $\mathbb{K}[x_1, \dots, x_k]$  the polynomial ring over  $k$  variables with coefficients in some field  $\mathbb{K}$ . Further, we use the symbol  $\mathbb{P}$  for probability measures and  $\mathbb{E}$  for the expected value operator.

### 2.1 Program Models

In accordance with [Hrushovski et al. 2023; Kovács and Varonka 2023], we consider **polynomial programs**  $\mathcal{P} = (Q, E, q_0)$  over  $k$  variables, where  $Q$  is a set of locations,  $q_0 \in Q$  is an initial location, and  $E \subseteq Q \times \mathbb{Q}[x_1, \dots, x_k] \times Q$  is a set of transitions. The vector of *variable valuations* is denoted as  $\vec{x} = (x_1, \dots, x_k)$ , where each transition  $(q, f, q') \in E$  maps a (program) configuration  $(q, \vec{x})$  to some configuration  $(q', f(\vec{x}))$ . A transition  $(q, f, q') \in E$  is *affine* if the function  $f$  is affine. In case all program transitions  $(q, f, q') \in E$  are affine, we say that the polynomial program  $\mathcal{P}$  is an *affine program*.

A **loop** is a program  $\mathcal{L} = (Q, E, q_0)$  with exactly two locations  $Q = \{q_0, q_1\}$ , such that the initial state  $q_0$  has exactly one outgoing transition to  $q_1$  and all outgoing transitions of  $q_1$  are self-loops, that is,  $E = \{(q_0, f_1, q_1), (q_1, f_2, q_1), \dots, (q_1, f_n, q_1)\}$ .

In a *guarded program*, each transition is additionally guarded by an equality/inequality predicate among variables of the state vector  $\vec{x}$ . If in some configuration the guard of an outgoing transition holds, we say that the transition is *enabled*, otherwise the transition is *disabled*.

**(Non)Deterministic programs.** If for any location  $q \in Q$  in a program  $\mathcal{P}$  there is exactly one outgoing transition  $(q, f, q')$ , then  $\mathcal{P}$  is **deterministic**; otherwise  $\mathcal{P}$  is *nondeterministic*. A deterministic guarded program may have multiple outgoing transitions from each location, but for any configuration, exactly one outgoing transition must be enabled. For a guarded nondeterministic program, we require that each configuration has at least one enabled outgoing transition. Deterministic, unguarded programs are called *single-path* programs.

To capture the concept of a loop invariant, we consider the collecting semantics of  $\mathcal{P}$ , associating each location  $q \in Q$  with a set of vectors  $S_q$  that are reachable from the initial state  $(q_0, \vec{0})$ . More formally, the sets  $\{S_q \mid q \in Q\}$  are the least solution of the inclusion system

$$S_{q_0} \supseteq \{\vec{0}\} \quad \text{and} \quad S_{q'} \supseteq f(S_q) \quad \text{for all } (q, f, q') \in E.$$

**Definition 2.1 (Invariant).** A polynomial  $p \in \overline{\mathbb{Q}}[x_1, \dots, x_k]$  is an *invariant* with respect to program location  $q \in Q$ , if for all reachable configurations  $\vec{x} \in S_q$  the polynomial vanishes, that is  $p(\vec{x}) = 0$ . Moreover, for a loop  $\mathcal{L}$ , the polynomial  $p$  is an *invariant of  $\mathcal{L}$* , if  $p$  is an invariant with respect to the looping state  $q_1$ .

**Probabilistic programs.** In probabilistic programs, a probability  $pr$  is added to each program transition. That is,  $E \subseteq Q \times Q[x_1, \dots, x_k] \times (0, 1] \times Q$ , where we require that each location has countably many outgoing transitions and that their probabilities  $pr$  sum up to 1. Under the intended semantics, a transition  $(q, f, pr, q')$  then maps a configuration  $(q, \vec{x})$  to configuration  $(q', f(\vec{x}))$  with probability  $pr$ . Again, for guarded probabilistic programs, we require that each configuration has at least one enabled outgoing transition and that the probabilities of the enabled transition sum up to 1.

For probabilistic programs  $\mathcal{P}$ , we consider moment invariants over higher-order statistical moments of the probability distributions induced by  $\mathcal{P}$  (see Section 5). In this respect, it is necessary to count the number of executed transitions in the semantics of  $\mathcal{P}$ . Formally, the sets  $\{S_q^n \mid q \in Q, n \in \mathbb{N}_0\}$  are defined as

$$S_{q_0}^0 := \{\vec{0}\} \quad \text{and} \quad S_{q'}^{n+1} := f(S_q^n) \quad \text{for all } (q, f, pr, q') \in E \text{ and } n \in \mathbb{N}_0.$$

In addition, the probability of a configuration  $\vec{x}$  in location  $q$  after  $n$  iterations, in symbols  $\mathbb{P}(\vec{x} \mid S_q^n)$ , can be defined inductively: (i) in the initial state, the configuration  $\vec{0}$  after 0 executed transitions has probability 1; (ii) for any other state, the probability of reaching a specific configuration is defined by summing up the probabilities of all incoming paths. More formally, the probability  $\mathbb{P}(\vec{x} \mid S_q^n)$  is

$$\mathbb{P}(\vec{x} \mid S_q^0) := \begin{cases} 1 & q = q_0 \wedge \vec{x} = \vec{0} \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \mathbb{P}(\vec{x} \mid S_{q'}^{n+1}) := \sum_{(q, f, pr, q') \in E} \sum_{\vec{y} \in f^{-1}(\vec{x})} pr \cdot \mathbb{P}(\vec{y} \mid S_q^n).$$

We then define the  $n$ th higher-order statistical moment of a monomial  $M$  in program variables as the expected value of  $M$  after  $n$  loop iterations. Namely,

$$\bullet \quad \mathbb{E}[M_n] := \sum_{q \in Q} M(\vec{x}) \cdot \mathbb{P}(\vec{x} \mid S_q^n), \quad (1)$$

where  $M(\vec{x})$  evaluates the monomial  $M$  in a specific configuration  $\vec{x}$ .

**Universality of loops.** In this paper, we focus on polynomial loops. This is justified by the universality of loops [Hrushovski et al. 2023, Section 4], as every polynomial program can be transformed into a polynomial loop that preserves the collecting semantics. Intuitively, this is done by merging all program states into the looping state and by introducing additional variables that keep track of which state is actually active while invalidating infeasible traces. It is then possible to recover the sets  $S_q^{(n)}$  of the original program from the sets  $S_q^{(n)}$  of the loop.

## 2.2 Computational Algebraic Geometry & Strongest Invariants

We study polynomial invariants  $p(\vec{x})$  of polynomial programs; here,  $p(\vec{x})$  are multivariate polynomials in program variables  $\vec{x}$ . We therefore recap necessary terminology from algebraic geometry [Cox et al. 1997], to support us in reasoning whether  $p(\vec{x}) = 0$  is a loop invariant. In the following  $\mathbb{K}$  denotes a field, such as  $\mathbb{R}$ ,  $\mathbb{Q}$  or  $\overline{\mathbb{Q}}$ .

*Definition 2.2 (Ideal).* A subset of polynomials  $I \subseteq \mathbb{K}[x_1, \dots, x_k]$  is an *ideal* if (i)  $0 \in I$ ; (ii) for all  $x, y \in I$ :  $x + y \in I$ ; and (iii) for all  $x \in I$  and  $y \in \mathbb{K}[x_1, \dots, x_k]$ :  $xy \in I$ . For polynomials  $p_1, \dots, p_l \in \mathbb{K}[x_1, \dots, x_k]$  we denote by  $\langle p_1, \dots, p_l \rangle$  the set generated by these polynomials, that is

$$\langle p_1, \dots, p_l \rangle := \left\{ \sum_{i=1}^l q_i p_i \mid q_1, \dots, q_k \in \mathbb{K}[x_1, \dots, x_k] \right\}$$

The set  $I = \langle p_1, \dots, p_l \rangle$  is an ideal, with the polynomials  $p_1, \dots, p_l$  being a *basis* of  $I$ .

Of particular importance to our work is the set of all polynomial invariants of a program location. It is easy to check that this set forms an ideal.

*Definition 2.3 (Invariant Ideal).* Let  $\mathcal{P}$  be a program with location  $q$ . The set  $\mathcal{I}$  of all invariants with respect to the location  $q$  is called the *invariant ideal* of  $q$ . If  $\mathcal{P}$  is a loop and  $\mathcal{I}$  is the invariant ideal with respect to the looping state  $q_1$ , we call  $\mathcal{I}$  the invariant ideal of the loop  $\mathcal{P}$ <sup>1</sup>.

As the invariant ideal  $\mathcal{I}$  of a loop  $\mathcal{L}$  contains *all* polynomial invariants, a basis for  $\mathcal{I}$  is the strongest polynomial invariant of  $\mathcal{L}$ . This is further justified by the following key result, establishing that every ideal has a basis.

**THEOREM 2.4 (HILBERT’S BASIS THEOREM).** *Every ideal  $I \subseteq \mathbb{K}[x_1, \dots, x_k]$  has a basis. That is,  $I = \langle p_1, \dots, p_l \rangle$  for some  $p_1, \dots, p_l \in I$ .*

While an ideal  $I$  may have infinitely many bases, the work of [Buchberger 2006] proved that every ideal  $I$  has a unique (reduced) *Gröbner basis*, where uniqueness is guaranteed modulo some *monomial order*. A monomial order  $<$  is a total order on all monomials such that for all monomials  $m_1, m_2, m_3$ , if  $m_1 < m_2$  then  $m_1 m_3 < m_2 m_3$ . For instance, assume our polynomial ring is  $\mathbb{K}[x, y, z]$ , that is, over three variables  $x, y$ , and  $z$ . A total order  $z < y < x$  over variables can be extended to a lexicographic ordering on monomials, denoted also by  $<$  for simplicity. In this case, for example,  $xyz^3 < xy^2$  and  $y^2z < x$ . For a given monomial order, one can consider the leading term of a polynomial  $p$  which we denote by  $LT(p)$ . For a set of polynomials  $S$  we write  $LT(S)$  for the set of all leading terms of all polynomials.

*Definition 2.5 (Gröbner Basis).* Let  $I \subseteq \mathbb{K}[x_1, \dots, x_k]$  be an ideal and fix a monomial order. A basis  $G = \{g_1, \dots, g_k\}$  of  $I$  is a *Gröbner basis*, if  $\langle LT(g_1), \dots, LT(g_l) \rangle = \langle LT(I) \rangle$ . Further,  $G$  is a *reduced Gröbner basis* if every  $g_i$  has leading coefficient 1 and for all  $g, h \in G$  with  $g \neq h$ , no monomial in  $g$  is a multiple of  $LT(h)$ .

Gröbner bases provide the workhorses to compute and implement algebraic operations over (infinite) ideals, including ideal intersections/unions, variable eliminations, and polynomial memberships. Given *any* basis for an ideal  $I$ , a unique reduced Gröbner basis with respect to any monomial ordering  $<$  is computable using Buchberger’s algorithm [Buchberger 2006]. A central property of Gröbner basis computation is that repeated division of a polynomial  $p$  by elements of a Gröbner basis results in a unique remainder, regardless of the order in which the divisions are performed. Hence, to decide if a polynomial  $p$  is an element of an ideal  $I$ , that is deciding polynomial membership, it suffices to divide  $p$  by a Gröbner basis of  $I$  and check if the remainder is 0. Moreover, eliminating a variable  $y$  from an ideal  $I \subseteq \mathbb{K}[x, y]$  is performed by computing the Gröbner basis of the elimination ideal  $I \cap \mathbb{K}[x]$  only over  $x$ .

<sup>1</sup>Computing bases for invariant ideals is equivalent to computing the *Zariski closure* of the loop: the Zariski closure is the smallest algebraic set containing the set of reachable states [Hrushovski et al. 2018].

### 2.3 Recurrence Equations

Recurrence equations relate elements of a sequence to previous elements. There is a strong connection between recurrence equations and program loops: assignments in program loops relate values of program variables in the current iteration to the values in the next iteration. It is therefore handy to interpret a (polynomial) program loop as a recurrence. We briefly introduce linear and polynomial recurrence systems and refer to [Kauers and Paule 2011] for details.

We say that a sequence  $u(n) : \mathbb{N}_0 \rightarrow \mathbb{Q}$  is a *linear recurrence sequence (LRS)* of order  $k$ , if there are coefficients  $a_0, \dots, a_{k-1} \in \mathbb{Q}$ , where  $a_0 \neq 0$  and for all  $n \in \mathbb{N}_0$  we have

$$u(n+k) = a_{k-1}u(n+k-1) + \dots + a_1u(n+1) + a_0u(n) \quad (2)$$

The recurrence equation (2) is called a *linear recurrence equation*, with the coefficients  $a_0, \dots, a_{k-1}$  and the initial values  $u(0), \dots, u(k-1)$  uniquely specifying the sequence  $u(n)$ . Any LRS  $u(n)$  of order  $k$  as defined via (2) can be specified by a system of  $k$  linear recurrence sequences  $u_1(n), \dots, u_k(n)$ , such that each  $u_i(n)$  is of order 1 and, for all  $n \in \mathbb{N}_0$ , we have  $u(n) = u_1(n)$  and

$$\begin{aligned} u_1(n+1) &= \sum_{i=1}^k a_i^{(1)} u_i(n) = a_1^{(1)} u_1(n) + \dots + a_k^{(1)} u_k(n) \\ &\vdots \\ u_k(n+1) &= \sum_{i=1}^k a_i^{(k)} u_i(n) = a_1^{(k)} u_1(n) + \dots + a_k^{(k)} u_k(n) \end{aligned} \quad (3)$$

Again, the LRS  $u(n)$  is uniquely defined by the coefficients  $a_i^{(j)}$  and the initial values  $u_1(0), \dots, u_k(0)$ .

**Polynomial recursive sequences** are natural generalizations of linear recurrence sequences and allow not only linear combinations of sequence elements but also polynomial combinations [Cadilhac et al. 2020]. More formally, a sequence  $u(n)$  is *polynomial recursive*, if there exists  $k \in \mathbb{N}$  sequences  $u^1(n), \dots, u^k(n) : \mathbb{N}_0 \rightarrow \mathbb{Q}$  such that  $u(n) = u_1(n)$  and there are polynomials  $p_1, \dots, p_k \in \mathbb{Q}[u_1, \dots, u_k]$  such that, for all  $n \in \mathbb{N}_0$ , we have

$$\begin{aligned} u_1(n+1) &= p_1(u_1(n), \dots, u_k(n)) \\ &\vdots \\ u_k(n+1) &= p_k(u_1(n), \dots, u_k(n)) \end{aligned} \quad (4)$$

The sequence  $u(n)$  from (4) is uniquely defined by the polynomials  $p_1, \dots, p_k$  and the initial values  $u_1(0), \dots, u_k(0)$ . In contrast to linear recurrence sequences (2), polynomial recursive sequences (4) cannot be in general modeled using a single polynomial recurrence [Cadilhac et al. 2020]. Systems of recurrences are widely used to model the evolution of dynamical systems in discrete time.

We conclude this section by recalling the **SKOLEM** problem [Bilu et al. 2022; Lipton et al. 2022] related to linear recurrence sequences, whose decidability is an open question since the 1930s. We formally revise the definition from Section 1 as:

The **SKOLEM** Problem [Everest et al. 2003; Tao 2008]: Given an LRS  $u(n)$ ,  $n \in \mathbb{N}_0$ , does there exist some  $m \in \mathbb{N}_0$  such that  $u(m) = 0$ ?

In the upcoming sections, we show that the **SKOLEM** problem is reducible to the decidability of three fundamental problems in programming languages, namely **P2P**, **SPINV** and **PROB-SPINV** from



Section 1. As such, we prove that the **SKOLEM** problem gives us intrinsically hard computational lower bounds for **P2P**, **SPINV**, and **PROB-SPINV**.

### 3 HARDNESS OF REACHABILITY IN POLYNOMIAL PROGRAMS

We first address the computational limitations of reachability analysis within polynomial programs. It is decidable whether a loop with *affine* assignments reaches a target state from a given initial state [Kannan and Lipton 1980]. Additionally, even problems generalizing reachability are known to be decidable for linear loops, such as various model-checking problems [Karimov et al. 2022]. However, reachability for loops with polynomial assignments, or equivalently discrete-time polynomial dynamical systems, has been an open challenge. In this section, we address this reachability challenge via our **P2P** problem, showing that reachability in polynomial program loops is at least as hard as the **SKOLEM** problem (Theorem 3.3). To this end, let us revisit and formally define our **P2P** problem from Section 1, as follows.

The **Point-To-Point Reachability Problem (P2P)**: Given a system of  $k$  polynomial recursive sequences  $u_1(n), \dots, u_k(n), n \in \mathbb{N}_0$  and a target vector  $\vec{t} = (t_1, \dots, t_k)$ , does there exist some  $m \in \mathbb{N}_0$  such that for all  $1 \leq i \leq k$ , it holds that  $u_i(m) = t_i$ ?

To the best of our knowledge, nothing is known about the hardness of **P2P** for polynomial recursive sequences<sup>2</sup>, and hence for loops with arbitrary polynomial assignments, apart from the trivial lower bounds provided by the linear/affine cases [Kannan and Lipton 1980; Karimov et al. 2022].

In the sequel, in Theorem 3.3 we prove that the **P2P** problem for polynomial recursive sequences is *at least as hard* as **SKOLEM**. Doing so, we show that solving **SKOLEM** can be solved by *reducing* it to inputs for **P2P**, written in symbols as **SKOLEM**  $\leq$  **P2P**. We thus establish a computational lower bound for **P2P** in the sense that providing a decision procedure for **P2P** for polynomial recursive sequences would prove the decidability of the long-lasting open decision problem given by **SKOLEM**.

Even for a fixed target vector in **P2P**

**Our reduction for **SKOLEM**  $\leq$  **P2P****. In a nutshell, we fix an arbitrary **SKOLEM** instance, that is, a linear recurrence sequence  $u(n)$  of order  $k$ . We say that the instance  $u(n)$  is *positive*, if there exists some  $m \in \mathbb{N}_0$  such that  $u(m) = 0$ , otherwise we call the instance *negative*. Our reduction **SKOLEM**  $\leq$  **P2P** constructs an instance of **P2P** that reaches the all-zero vector  $\vec{0}$  if and only if the **SKOLEM** instance is positive. Hence, a decision procedure for **P2P** would directly lead to a decision procedure for **SKOLEM**.

Following (2), let our **SKOLEM** instance of order  $k$  to be the LRS  $u(n) : \mathbb{N}_0 \rightarrow \mathbb{Q}$  specified by coefficients  $a_0, \dots, a_{k-1} \in \mathbb{Q}$  such that  $a_0 \neq 0$  and, for all  $n \in \mathbb{N}_0$ , we have

$$u(n+k) = a_{k-1} \cdot u(n+k-1) + \dots + a_1 \cdot u(n+1) + a_0 \cdot u(n) = \sum_{i=0}^{k-1} a_i \cdot u(n+i). \quad (5)$$

From our **SKOLEM** instance (5), we construct a system of  $k$  polynomial recursive sequences  $x_0, \dots, x_{k-1}$ , as given in (4). Namely, the initial sequence values are defined inductively as

$$\boxed{x_0(0) := u(0)} \quad \boxed{x_i(0) := u(i) \cdot \prod_{\ell=0}^{i-1} x_\ell(0) \quad (1 \leq i < k)}$$

<sup>2</sup>For linear systems, the Point-To-Point Reachability problem (**P2P**) is also referred to as the *Orbit problem* in [Kannan and Lipton 1980].

The running product of a LRS is Polyrec, even SIMPLE polynec



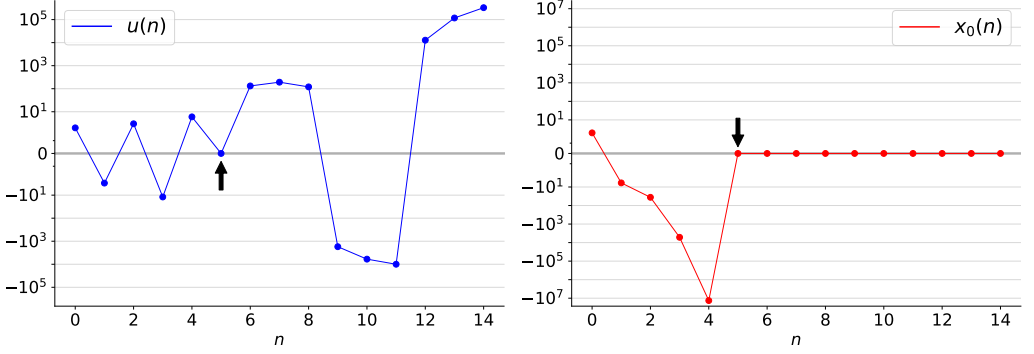


Fig. 2. The first 15 sequence elements of  $u(n)$  and  $x_0(n)$  in Example 3.1.

With the initial values defined, the sequences  $x_0, \dots, x_{k-1}$  are uniquely defined via the following system of recurrence equations:

$$\boxed{x_i(n+1) := x_{i+1}(n) \quad (1 \leq i < k-1)} \quad \boxed{x_{k-1}(n+1) := \sum_{i=0}^{k-1} a_i \cdot x_i(n) \cdot \prod_{\ell=i}^{k-1} x_\ell(n)} \quad (6)$$

Intuitively, the  $x_i$  sequences are a “non-linear variant” of the **SKOLEM** instance  $u(n)$  such that, **once any  $x_i$  reaches 0,  $x_i$  remains 0 forever. The target vector for our P2P instance is therefore  $\vec{t} = \vec{0}$ .**

Let us illustrate the main idea of our construction with the following example.

*Example 3.1.* Assume our **SKOLEM** instance from (5) is given by the recurrence  $u(n+3) = 2u(n+2) - 2u(n+1) - 12u(n)$  and the initial values  $u(0) = 2, u(1) = -3, u(2) = 3$ . Following our reduction (6), we construct a system of polynomial recursive sequences  $x_i(n)$ :

$$\begin{aligned} x_0(0) &= u(0) = 2 & x_0(n+1) &= x_1(n) \\ x_1(0) &= u(1)x_0(0) = -6 & x_1(n+1) &= x_2(n) \\ x_2(0) &= u(2)x_0(0)x_1(0) = -36 & x_2(n+1) &= 2x_2(n)^2 - 2x_1(n)^2x_2(n) - 12x_0(n)^2x_1(n)x_2(n) \end{aligned}$$

The first few sequence elements of  $u(n)$  and  $x_0(n)$  are shown in Figure 2 and illustrate the key property of our reduction:

- (i)  $x_0(n)$  is non-zero as long as  $u(n)$  is non-zero, which we prove in Lemma 3.2;
- (ii) if there is an  $N$  such that  $u(N) = 0$ , it holds that for all  $n \geq N$  :  $x_0(n) = 0$ . The other sequences  $x_1$  and  $x_2$  in the system are “shifted” variants of  $x_0$ . Hence, the constructed sequences all eventually reach the all-zero configuration and remain there. In Theorem 3.3, we prove that this is the case if and only if the **SKOLEM** instance  $u(n)$  is positive.

**Correctness of **SKOLEM**  $\leq$  P2P.** To prove the correctness of our reduction **SKOLEM**  $\leq$  P2P and to assert the properties (i)-(ii) of Example 3.1 among  $u(n)$  and  $x_i(n)$ , we introduce  $k$  auxiliary variables  $s_0, \dots, s_{k-1}$  defined as

$$\boxed{s_i(0) := \begin{cases} 1 & (i = 0) \\ \prod_{\ell=0}^{i-1} x_\ell(0) & (1 \leq i < k) \end{cases}} \quad \boxed{s_i(n+1) := \begin{cases} s_{i+1}(n) & (i \neq k-1) \\ s_{k-1}(n) \cdot x_{k-1}(n) & (i = k-1) \end{cases}}$$

Using these auxiliary sequences  $s_i(n)$ , we next prove two central properties of our **P2P** instance.

LEMMA 3.2. *For the system of polynomial recursive sequences in (6), it holds that  $\forall n \geq 0$  and  $0 \leq i < k$*

$$x_i(n) = s_i(n) \cdot u(n+i), \text{ and} \quad (7)$$

$$s_i(n) = \prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^{i-1} x_\ell(n). \quad (8)$$

PROOF. We prove the two properties by well-founded induction on the lexicographic order  $(n, i)$ , where  $n \geq 0$  and  $0 \leq i < k$ . Here,  $(n, i) \leq (n', i')$  if and only if  $n < n'$  or  $n = n' \wedge i < i'$ . The order has the unique least element  $(0, 0)$ .

*Base case:*  $n = 0$ . If  $i = 0$ , then properties (7) and (8) hold by definition of  $s_0(0) := 1 = \prod_{\ell=0}^{-1} x_0(\ell) \cdot \prod_{\ell=0}^{-1} x_\ell(0)$  and  $x_0(0) := u(0) = s_0(0) \cdot u(0)$ . Also, if  $0 < i < k$ , then properties (7) and (8) are trivially satisfied by the definition of the initial values:  $s_i(0) := \prod_{\ell=0}^{i-1} x_\ell(0)$  and  $x_i(0) := u(i) \cdot \prod_{\ell=0}^{i-1} x_\ell(0) = u(i) \cdot s_i(0)$ .

*Induction step – Case 1:*  $n > 0 \wedge 0 \leq i < k-1$ . By the lexicographical ordering, it holds that  $(n, i+1) < (n+1, i)$ . Hence, we can assume that properties (7) and (8) hold for  $(n, i+1)$ . Thus, we have the induction hypothesis

$$x_{i+1}(n) = s_{i+1}(n) \cdot u(n+i+1), \text{ and} \quad (9)$$

$$s_{i+1}(n) = \prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^i x_\ell(n). \quad (10)$$

To prove property (7) for  $(n+1, i)$  means to show that

$$x_i(n+1) = s_i(n+1) \cdot u(n+i+1).$$

The sequences  $x_i$  and  $s_i$  are defined by  $x_i(n+1) = x_{i+1}(n)$  and  $s_i(n+1) = s_{i+1}(n)$  and hence property (7) follows from the induction hypothesis (9).

To prove property (8) for  $(n+1, i)$  means to show that

$$s_i(n+1) = \prod_{\ell=0}^n x_0(\ell) \cdot \prod_{\ell=0}^{i-1} x_\ell(n+1).$$

We prove the equation by using the induction hypothesis (10), the definitions  $x_i(n+1) = x_{i+1}(n)$  and  $s_i(n+1) = s_{i+1}(n)$ , and index manipulation:

$$\begin{aligned} s_i(n+1) &= s_{i+1}(n) = \prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^i x_\ell(n) \\ &= \prod_{\ell=0}^{n-1} x_0(\ell) \cdot x_0(n) \cdot \prod_{\ell=0}^{i-1} x_{\ell+1}(n) \\ &= \prod_{\ell=0}^n x_0(\ell) \cdot \prod_{\ell=0}^{i-1} x_\ell(n+1) \end{aligned}$$

*Induction step – Case 2:*  $n > 0$  and  $i = k-1$ . We show that property (7) holds for  $(n+1, k-1)$  by proving it to be equivalent to the definition of  $x_{k-1}(n+1)$ . To do so, we first instantiate property (7)

and replace both  $s_{k-1}(n+1)$  and  $u(n+k)$  by their defining recurrence:

$$\begin{aligned} x_{k-1}(n+1) &= s_{k-1}(n+1) \cdot u(n+k) \\ &= s_{k-1}(n) \cdot x_{k-1}(n) \cdot \left( \sum_{i=0}^{k-1} a_i \cdot u(n+i) \right) \end{aligned}$$

Next, we rearrange and apply the induction hypothesis (8) for  $(n, k-1)$  and  $(n, i)$  and obtain:

$$\begin{aligned} x_{k-1}(n+1) &= x_{k-1}(n) \cdot \left( \sum_{i=0}^{k-1} a_i \cdot u(n+i) \cdot s_{k-1}(n) \right) \\ &= x_{k-1}(n) \cdot \left( \sum_{i=0}^{k-1} a_i \cdot u(n+i) \cdot \underbrace{\prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^{k-2} x_\ell(n)}_{s_{k-1}(n) \text{ by I.H. (8)}} \right) \\ &= x_{k-1}(n) \cdot \left( \sum_{i=0}^{k-1} a_i \cdot u(n+i) \cdot \underbrace{\prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^{i-1} x_\ell(n)}_{=s_i(n) \text{ by I.H. (8)}} \cdot \prod_{\ell=i}^{k-2} x_\ell(n) \right) \\ &= x_{k-1}(n) \cdot \left( \sum_{i=0}^{k-1} a_i \cdot u(n+i) \cdot s_i(n) \cdot \prod_{\ell=i}^{k-2} x_\ell(n) \right) \\ &= \sum_{i=0}^{k-1} a_i \cdot u(n+i) \cdot s_i(n) \cdot \prod_{\ell=i}^{k-1} x_\ell(n) \end{aligned}$$

Now, we can apply the induction hypothesis (7) to replace  $u(n+i) \cdot s_i(n)$  by  $x_i(n)$  and arrive at the relation:

$$x_{k-1}(n+1) = \sum_{i=0}^{k-1} a_i \cdot x_i(n) \cdot \prod_{\ell=i}^{k-1} x_\ell(n)$$

However, this is exactly the defining recurrence equation from (6). Hence, property (8) necessarily holds for  $(n, k-1)$ .

To prove property (8) for  $(n+1, k-1)$  we use the defining equation of  $s_{k-1}(n+1)$  and the induction hypothesis for  $(n, k-1)$ :

$$\begin{aligned} s_{k-1}(n+1) &= s_{k-1}(n) \cdot x_{k-1}(n) = x_{k-1}(n) \cdot \prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^{k-2} x_\ell(n) = \prod_{\ell=0}^{n-1} x_0(\ell) \cdot \prod_{\ell=0}^{k-1} x_\ell(n) \\ &= \prod_{\ell=0}^{n-1} x_0(\ell) \cdot x_0(n) \cdot \prod_{\ell=0}^{k-2} x_{\ell+1}(n) = \prod_{\ell=0}^n x_0(\ell) \cdot \prod_{\ell=0}^{k-2} x_\ell(n+1) \end{aligned}$$

As we have covered all possible cases, we conclude the proof.  $\square$

Lemma 3.2 establishes two central properties of our reduction. We now use these properties to show that **P2P** is at least as hard as **SKOLEM**.

**THEOREM 3.3 (HARDNESS OF P2P).** *P2P is SKOLEM-hard. That is,  $SKOLEM \leq P2P$ .*

**PROOF.** We show that our polynomial recursive system constructed in (6) reaches the all-zero vector from the initial value if and only if the original SKOLEM instance is positive.

( $\Rightarrow$ ) : Assume the SKOLEM instance is positive, then there is some smallest  $N \in \mathbb{N}_0$  such that  $u(N) = 0$ . Property (7) of Lemma 3.2 implies

$$x_0(N) = s_0(N) \cdot u(N) = 0.$$

Using this equation and property (8) of Lemma 3.2, we deduce that for all  $n > N$ , each  $s_i(n)$  contains  $x_0(N)$  as a factor and hence  $s_i(n) = 0$ . Additionally, as  $x_i(n) = s_i(n) \cdot u(n+i)$  by property (7), we conclude that for all  $n > N$  also  $x_i(n) = 0$ . Hence, the polynomial recursive system reaches the all-zero vector.

( $\Leftarrow$ ) Assume that the SKOLEM instance is negative, meaning that the linear recurrence sequence  $u(n)$  does not have a 0. In particular,  $u(i) \neq 0$  for all  $0 \leq i < k$ . Therefore, by definition of the polynomial recursive system (6),  $x_i(0) \neq 0$  for all  $0 \leq i < k$ . Towards a contradiction, assume that the polynomial recursive system still reaches the all-zero vector. Hence, there is a smallest  $N \in \mathbb{N}_0$  such that  $x_i(N) = 0$  for all  $0 \leq i < k$ . In particular,  $x_0(N) = 0$ . Moreover,  $x_0$  is the last sequence to reach 0, because of the recurrence equation  $x_i(n+1) = x_{i+1}(n)$  for  $0 \leq i < k$ . Therefore,  $N$  is also the smallest number such that  $x_0(N) = 0$ . By property (7) of Lemma 3.2, we have

$$x_0(N) = s_0(N) \cdot u(N) = 0.$$

However,  $s_0(N)$  must be non-zero, because

$$s_0(N) = \prod_{\ell=0}^{N-1} x_0(\ell),$$

by property (8) of Lemma 3.2, and the fact that  $N$  is the smallest number such that  $x_0(N) = 0$ . Then we necessarily have  $u(N) = 0$ , yielding a contradiction.  $\square$

Theorem 3.3 shows that P2P for polynomial recursive sequences is at least as hard as the SKOLEM problem. Thus, reachability and model-checking of loops with polynomial assignments is SKOLEM-hard. A decision procedure establishing decidability for P2P would lead to major breakthroughs in number theory [Lipton et al. 2022], as by Theorem 3.3 this would imply decidability of the SKOLEM problem.

#### 4 HARDNESS OF COMPUTING THE STRONGEST POLYNOMIAL INVARIANT

This section goes beyond reachability analysis and focuses on inferring the strongest polynomial invariants of polynomial loops. As such, we turn our attention to solving the SPINV problem of Section 1, which is formally defined as given below.

The SPINV Problem: Given an unguarded, deterministic loop with polynomial updates, compute a basis of its polynomial invariant ideal.

We prove that finding the strongest polynomial invariant for deterministic loops with polynomial updates, that is, solving SPINV, is at least as hard as P2P (Theorem 4.2). Hence,  $P2P \leq SPINV$ .

Then, by the  $SKOLEM \leq P2P$  hardness result of Theorem 3.3, we conclude the SKOLEM-hardness of SPINV, that is  $SKOLEM \leq P2P \leq SPINV$ . To the best of our knowledge, our Theorem 3.3 together

with Theorem 4.2 provide the first computational lower bound on **SPInv**, when focusing on loops with arbitrary polynomial updates (see Table 1).

**Our reduction for  $\mathbf{P2P} \leq \mathbf{SPInv}$ .** We fix an arbitrary **P2P** instance of order  $k$ , given by a system of polynomial recursive sequences  $u_1, \dots, u_k : \mathbb{N}_0 \rightarrow \mathbb{Q}$  and a target vector  $\vec{t} = (t_1, \dots, t_k) \in \mathbb{Q}^k$ . This **P2P** instance is positive if and only if there exists an  $N \in \mathbb{N}_0$  such that  $(u_1(N), \dots, u_k(N)) = \vec{t}$ . For reducing **P2P** to **SPInv**, we construct the following deterministic loop with polynomial updates over  $k+2$  variables:

Deterministic loop = Polyrec

$$\begin{aligned}
 & [f \ g \ x_1 \ \dots \ x_k] \leftarrow [1 \ 0 \ u_1(0) \ \dots \ u_k(0)] \\
 & \textbf{while } \star \textbf{ do} \\
 & \quad \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ f \\ g \end{bmatrix} \leftarrow \begin{bmatrix} p_1(x_1, \dots, x_k) \\ \vdots \\ p_k(x_1, \dots, x_k) \\ f \cdot ((x_1 - t_1)^2 + \dots + (x_k - t_k)^2) \\ g + 1 \end{bmatrix} \\
 & \textbf{end while}
 \end{aligned} \tag{11}$$

The polynomial recursive sequences  $u_1, \dots, u_k$  are fully determined by their initial values and the polynomials  $p_1, \dots, p_k \in \mathbb{Q}[u_1, \dots, u_k]$  defining the respective recurrence equations  $u_i(n+1) = p_i(u_1(n), \dots, u_k(n))$ . Hence, by the construction of the **SPInv** instance (11), every program variable  $x_i$  models the sequence  $u_i$ . As such, for any number of loop iterations  $n \in \mathbb{N}_0$ , we have  $x_i(n) = u_i(n)$ . Moreover, the variable  $g$  models the loop counter  $n$ , meaning  $g(n) = n$  for all  $n \in \mathbb{N}_0$ . The motivation behind using the program variable  $f$  is that  **$f$  becomes 0 as soon as all sequences  $u_i$  reach their target  $t_i$ ; moreover,  $f$  remains 0 afterward.** More precisely, for  $n \in \mathbb{N}_0$ ,  $f(n) = 0$  if and only if there is some  $N \leq n$  such that  $x_1(N) = t_1 \wedge \dots \wedge x_k(N) = t_k$ . Hence, the sequence  $f$  has a 0 value, and subsequently, all its values are 0, if and only if the original instance of **P2P** is positive.

Let us illustrate the main idea of our  $\mathbf{P2P} \leq \mathbf{SPInv}$  reduction via the following example.

*Example 4.1.* Consider the recursive sequences  $x(n+1) = x(n) + 2$  and  $y(n+1) = y(n) + 3$ , with initial values  $x(0) = y(0) = 0$ . It is easy to see that the system  $S = (x(n), y(n))$  reaches the target  $\vec{t}_1 = (4, 6)$  but does not reach the target  $\vec{t}_2 = (5, 7)$ . Following are the two **SPInv** instances produced by our reduction for the **P2P** instances  $(S, \vec{t}_1)$  and  $(S, \vec{t}_2)$ .

**SPInv instance for  $(S, \vec{t}_1)$ :**

$$\begin{aligned}
 & [f \ g \ x \ y] \leftarrow [1 \ 0 \ 0 \ 0] \\
 & \textbf{while } \star \textbf{ do} \\
 & \quad \begin{bmatrix} x \\ y \\ f \\ g \end{bmatrix} \leftarrow \begin{bmatrix} x + 2 \\ y + 3 \\ f \cdot ((x - 4)^2 + (y - 6)^2) \\ g + 1 \end{bmatrix} \\
 & \textbf{end while}
 \end{aligned}$$

Invariant ideal:  $\langle x - 2g, y - 3g, g(g - 1)f \rangle$

**SPInv instance for  $(S, \vec{t}_2)$ :**

$$\begin{aligned}
 & [f \ g \ x \ y] \leftarrow [1 \ 0 \ 0 \ 0] \\
 & \textbf{while } \star \textbf{ do} \\
 & \quad \begin{bmatrix} x \\ y \\ f \\ g \end{bmatrix} \leftarrow \begin{bmatrix} x + 2 \\ y + 3 \\ f \cdot ((x - 5)^2 + (y - 7)^2) \\ g + 1 \end{bmatrix} \\
 & \textbf{end while}
 \end{aligned}$$

Invariant ideal:  $\langle x - 2g, y - 3g \rangle$

The invariant ideals for both instances are given in terms of Gröbner bases with respect to the lexicographic order for the variable order  $g < f < y < x$ .

For the instance with the reachable target  $\vec{t}_1$ , we have  $f(n) = 0$  for  $n \geq 2$ . Hence,  $g(g-1)f$  is a polynomial invariant and must be in the invariant ideal of this **SPInv** instance; in fact,  $g(g-1)f$  is not only in the invariant ideal but even a basis element for the Gröbner basis with the chosen order. However,  $g(g-1)f$  is not in the ideal of the **SPInv** instance with the unreachable target  $\vec{t}_2$ . These two **SPInv** instances illustrate thus how a basis of the invariant ideal can be used to decide **P2P**.

While, for simplicity, our recursive sequences  $x(n)$  and  $y(n)$  are linear, our approach to reducing **P2P** to **SPInv** also applies to polynomial recursive sequences. In Theorem 4.2, we show that a polynomial such as  $g(g-1)f$  is an element of the basis of the invariant ideal (with respect to a specific monomial order) if and only if the original **P2P** instance is positive.

**Correctness of  $\mathbf{P2P} \leq \mathbf{SPInv}$ .** To show that it is decidable whether  $f(n)$  has a 0 given a basis of the invariant ideal, we employ Gröbner bases and an argument introduced in [Kauers 2005] for recursive sequences defined by rational functions, adjusted to our setting using recursive sequences defined by polynomials.

*→ not just an ideal membership query.*

**THEOREM 4.2 (HARDNESS OF **SPInv**).** ***SPInv** is at least as hard as **P2P**. That is,  $\mathbf{P2P} \leq \mathbf{SPInv}$ .*

**PROOF.** Assume we are given an oracle for **SPInv**, computing a basis  $B$  of the polynomial invariant ideal  $\mathcal{I} = \langle B \rangle$  of our loop (11). We show that given such a basis  $B$ , it is decidable whether  $f(n)$  has a root, which is equivalent to the fixed **P2P** instance being positive.

Note that by the construction of the loop (11), if  $f(N) = 0$  for some  $N \in \mathbb{N}_0$ , then  $\forall n \geq N : f(n) = 0$ . Moreover, such an  $N$  exists if and only if the **P2P** instance is positive. This is true if and only if there exists an  $N \in \mathbb{N}_0$  such that the sequence

$$n \mapsto f(n) \cdot n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot (n-N+1)$$

is 0 for all  $n \in \mathbb{N}_0$ . Consequently, the polynomial invariant ideal  $\mathcal{I}$  contains a polynomial

$$P := f \cdot g \cdot (g-1) \dots \cdot (g-N+1) \tag{12}$$

for some  $N \in \mathbb{N}_0$  only if the **P2P** instance (11) is positive. It is left to show that, given a basis  $B$  of  $\mathcal{I}$ , it is decidable whether  $\mathcal{I}$  contains a polynomial (12). Using Buchberger's algorithm [Buchberger 2006],  $B$  can be transformed into a Gröbner basis with respect to any monomial order. We choose a total order among program variables such that  $g < f < x_1, \dots, x_k$ . Without loss of generality, we assume that  $B$  is a Gröbner basis with respect to the lexicographic order extending the variable order.

In what follows, we argue that if a polynomial  $P$  as in (12) is an element of  $\mathcal{I}$ , then  $P$  must be an element of the basis  $B$ . As the leading term of  $P$  is  $g^N \cdot f$ , there must be some polynomial  $Q$  in  $B$  with a leading term that divides  $g^N \cdot f$ . By the choice of the lexicographic order, this polynomial must be of the form  $Q = Q_1(g) \cdot f - Q_2(g)$ , since if any other term would occur in  $Q$ , it would necessarily be in the leading term. As both  $P \in \mathcal{I}$  and  $Q \in \mathcal{I}$ , it holds that

$$P \cdot Q_1 - g \cdot (g-1) \dots \cdot (g-N+1) \cdot Q \in \mathcal{I}.$$

By expanding  $P$  and  $Q$ , we see that the above polynomial is equivalent to

$$Q_2 \cdot g \cdot (g-1) \dots \cdot (g-N+1).$$

As this polynomial is in the ideal  $\mathcal{I}$ , it follows that for all  $n \in \mathbb{N}_0$ :

$$Q_2(n) \cdot n \cdot (n-1) \dots \cdot (n-N+1) = 0.$$

However, this implies that  $Q_2(n)$  has infinitely many zeros, a property that is unique to the zero polynomial. Therefore, we conclude that  $Q_2 \equiv 0$ . Hence, if the original P2P instance is positive, there necessarily exists a basis polynomial of the form  $Q_1(g) \cdot f$ .

We show that this basis polynomial  $Q_1(g) \cdot f$  actually has the form (12): choose the basis polynomial of the form  $Q_1(g) \cdot f$  such that  $Q_1$  has minimal degree. Note that  $Q_1(g) \cdot f$  must divide  $P$ . Assume  $Q_1(g)$  is not of the form  $g \cdot (g-1) \dots (g-N+1)$ . Then, at least one factor  $(g-m)$  is not a factor of  $Q_1$ , or equivalently  $Q_1(m) \neq 0$ . Then, necessarily  $f(m) = 0$  and  $g \cdot (g-1) \dots (g-m+1) \cdot f$  must be in the ideal  $I$ , contradicting the minimality of the degree of  $Q_1$ .

Therefore, we conclude that the P2P instance is positive if and only if the Gröbner basis contains a polynomial of the form (12). As the basis  $B$  is finite, this property can be checked by enumeration of the basis elements of  $B$ . Hence, given an oracle for SPINV, we can decide if the P2P instance is positive or negative.  $\square$

Theorem 4.2 shows that SPINV is at least as hard as the P2P problem. Together with Theorem 3.3, we conclude that SPINV is SKOLEM-hard.

**An improved direct reduction from SKOLEM to SPINV.** Theorem 4.2 together with Theorem 3.3 yields the chain of reductions

$$\text{SKOLEM} \leq \text{P2P} \leq \text{SPINV}.$$

Within these reductions, a SKOLEM instance of order  $k$  yields a P2P instance with  $k$  sequences, which in turn reduces to a SPINV instance over  $k+2$  variables.

We conclude this section by noting that, if the linear recurrence sequence of the SKOLEM-instance is an *integer sequence*, then a reduction directly from SKOLEM to SPINV can be established by using only  $k+1$  variables. A slight modification of SKOLEM  $\leq$  P2P reduction of Section 3 results in a reduction from SKOLEM instances of order  $k$  directly to SPINV instances with  $k+1$  variables. Any system of polynomial recursive sequences can be encoded in a loop with polynomial updates. Hence, the instance produced by the SKOLEM  $\leq$  P2P reduction can be interpreted as a loop. It is sufficient to modify the resulting loop in the following way:

$$\begin{array}{c} x_{k-1} \leftarrow \sum_{i=0}^{k-1} a_i \cdot x_i \cdot \prod_{\ell=i}^{k-1} x_\ell \\ s_{k-1} \leftarrow x_{k-1} \cdot s_{k-1} \end{array} \quad \rightarrow \quad \begin{array}{c} x_{k-1} \leftarrow \sum_{i=0}^{k-1} a_i \cdot x_i \cdot \prod_{\ell=i}^{k-1} 2 \cdot x_\ell \\ s_{k-1} \leftarrow 2 \cdot x_{k-1} \cdot s_{k-1} \end{array}$$

As in the reduction in Section 3, the equation  $u_0(n) = \frac{x_0(n)}{s_0(n)}$  still holds and the resulting loop reaches the all-zero configuration if and only if the original SKOLEM-instance is positive (the integer sequence has a 0). Additionally, the resulting loop has infinitely many *different* configurations if and only if the SKOLEM instance is positive, as the additional factor in the updates forces a strict increase in  $|s_{k-1}|$ . Assuming a solution to SPINV for the constructed loop, that is a basis of the polynomial invariant ideal, it is decidable whether the number of reachable program locations (and its algebraic closure) is finite or not [Cox et al. 1997]. Therefore, an oracle for SPINV implies the decidability of SKOLEM for *integer sequences*, while the chain of reductions SKOLEM  $\leq$  P2P  $\leq$  SPINV is also valid for rational sequences.

**Summary of computability results in polynomial (non)deterministic loops.** We conclude this section by overviewing our computability results in Table 1, focusing on the strongest polynomial invariants of (non)deterministic loops and in relation to the state-of-the-art.



Program Model			Strongest Affine Invariant		Strongest Polynomial Invariant	
Det.	Unguarded	Affine	✓	[Karr 1976]	✓	[Kovács 2008]
		Poly.	✓	[Müller-Olm and Seidl 2004a]	SKOLEM-hard	Theorems 3.3 & 4.2
	Guarded ( $=, <$ )	Affine	✗ (Halting Problem)			
		Poly.				
Nondet.	Unguarded	Affine	✓	[Karr 1976]	✓	[Hrushovski et al. 2023]
		Poly.	✓	[Müller-Olm and Seidl 2004a]	✗	[Hrushovski et al. 2023]
	Guarded ( $=, <$ )	Affine	✗ [Müller-Olm and Seidl 2004b]			
		Poly.				

Table 1. Summary of computability results for strongest invariants of *nonprobabilistic* polynomial loops, including our own results (Theorems 3.3 & 4.2). With '✓' we denote decidable problems, while '✗' denotes undecidable problems.

## 5 STRONGEST INVARIANT FOR PROBABILISTIC LOOPS

In this section, we finally go beyond (non-)deterministic programs and address computational challenges in probabilistic programming, in particular loops. Unlike the programming models of Section 3–4, probabilistic loops follow different transitions with different probabilities.

Recall that the standard definition of an invariant  $I$ , as given in Definition 2.1, demands that  $I$  holds in *every reachable* configuration and location. As such, when using Definition 2.1 to define an invariant  $I$  of a probabilistic loop, the information provided by the probabilities of reaching a configuration within the respective loop is omitted in  $I$ . However, Definition 2.1 captures an invariant  $I$  of a probabilistic loop when every probabilistic loop transition is replaced by a nondeterministic transition.

Nevertheless, for incorporating probability-based information in loop invariants, Definition 2.1 needs to be revised to consider expected values and higher (statistical) moments describing the value distributions of probabilistic loop variables [Kozen 1983; McIver and Morgan 2005]. Therefore, in Definition 5.2 we introduce *polynomial moment invariants* to reason about value distributions of probabilistic loops. We do so by utilizing higher moments of the probability distributions induced by the value distributions of loop variables during the execution (Section 5.1). We prove that polynomial moment invariants generalize classical invariants (Lemma 5.5) and show that the strongest moment invariants up to moment order  $\ell$  are computable for the class of so-called moment-computable polynomial loops (Section 5.2). In this respect, in Algorithm 1 we give a complete procedure for computing the strongest moment invariants of moment-computable polynomial loops. When considering *arbitrary* polynomial probabilistic loops, we prove that the strongest moment invariants are (i) not computable for guarded probabilistic loops (Section 5.3) and (ii) SKOLEM-hard to compute for unguarded probabilistic loops (Section 5.4).

### 5.1 Polynomial Moment Invariants

Higher moments capture expected values of monomials over loop variables, for example,  $\mathbb{E}[x^2]$  and  $\mathbb{E}[xy]$  respectively yield the second-order moment of  $x$  and a second-order mixed moment. Such higher moments are necessary to characterize, and potentially recover, the value distribution of probabilistic loop variables, allowing us to reason about statistical properties, such as variance or skewness, over probabilistic value distributions.

When reasoning about moments of probabilistic program variables, note that in general neither  $\mathbb{E}[x^\ell] = \mathbb{E}[x]^\ell$  nor  $\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y]$  hold, due to potential dependencies among the (random) loop variables  $x$  and  $y$ . Therefore, describing all polynomial invariants among all higher moments

*infinitely many variables*

by finitely many polynomials is futile. A natural restriction and the one we undertake in this paper is to consider polynomials over finitely many moments, which we do as follows.

*Definition 5.1 (Moments of Bounded Degree).* Let  $\ell$  be a positive integer. Then the *set of program variable moments of order at most  $\ell$*  is given by

$$\mathbb{E}^{\leq \ell} := \{ \mathbb{E}[x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_k^{\alpha_k}] \mid \alpha_1 + \dots + \alpha_k \leq \ell \}.$$

While Definition 5.1 uses a bound  $\ell$  to define the set of moments of bounded degree, our subsequent results apply to any *finite* set of moments of program variables.

Recall that Section 2.1 defines the semantics  $S_q^n$  of a probabilistic loop with respect to the location  $q \in Q$  and the number of executed transitions  $n \geq 0$ . The set  $S_q^n$  in combination with the probability of each configuration allows us to define the moments of program variables after  $n$  transitions. Further, for a monomial  $M$  in program variables, we defined  $\mathbb{E}[M_n]$  in (1) to be the expected value of  $M$  after  $n$  transitions. For example,  $\mathbb{E}[x_n]$  denotes the expected value of the program variable  $x$  after  $n$  transitions. With this, we define the set of polynomial invariants among moments of program variables, as follows.

*Definition 5.2 (Moment Invariant Ideal).* Let  $\mathbb{E}^{\leq \ell}$  be the set of program variable moments of order less than or equal to  $\ell$  and  $k = |\mathbb{E}^{\leq \ell}|$ . The *moment invariant ideal  $\mathbb{I}^{\leq \ell}$*  is defined as

$$\bullet \quad \mathbb{I}^{\leq \ell} = \left\{ p(M_1, \dots, M_k) \in \overline{\mathbb{Q}}[\mathbb{E}^{\leq \ell}] \mid p(M_{1,n}, \dots, M_{k,n}) = 0 \text{ for all } n \in \mathbb{N}_0 \right\}.$$

We refer to elements of  $\mathbb{I}^{\leq \ell}$  as *polynomial moment invariants*.

For example, using Definition 5.2, a polynomial  $p(\mathbb{E}[x], \mathbb{E}[y])$  in the expected values of the variables  $x$  and  $y$  is a *polynomial moment invariant*, if  $p(\mathbb{E}[x_n], \mathbb{E}[y_n]) = 0$  for all number of transitions  $n \in \mathbb{N}_0$ . Note that, although  $\mathbb{E}^{\leq \ell}$  is a finite set, the moment invariant ideal  $\mathbb{I}^{\leq \ell}$  is, in general, an infinite set.

*Example 5.3.* Consider two asymmetric random walks  $x_n$  and  $y_n$  that both start at the origin. Both random walks increase or decrease with probability  $1/2$ , respectively. The random walk  $x_n$  either decreases by 2 or increases by 1, while  $y_n$  behaves conversely, which means  $y_n$  either decreases by 1 or increases by 2. Following is a probabilistic loop encoding this process together with the moment invariant ideal  $\mathbb{I}^{\leq 2}$ . The loop is given as program code. The intended meaning of the expression  $e_1[pr]e_2$  is that it evaluates to  $e_1$  with probability  $pr$  and to  $e_2$  with probability  $1-pr$ .

```

[ x  y ] ← [ 0  0 ]
while ★ do
  [ x ] ← [ x + 2 [1/2] x - 1 ]
  [ y ] ← [ y + 1 [1/2] y - 2 ]
end while

```

Basis of the moment invariant ideal  $\mathbb{I}^{\leq 2}$ :

$$\begin{aligned}
& \mathbb{E}[x^2] - \mathbb{E}[y^2] \\
& 9 \cdot \mathbb{E}[x] - 2 \cdot \mathbb{E}[xy] - 2 \cdot \mathbb{E}[y^2] \\
& \mathbb{E}[xy]^2 + 2 \cdot \mathbb{E}[xy] \cdot \mathbb{E}[y^2] + {}^{81/4} \cdot \mathbb{E}[xy] + \mathbb{E}[y^2]^2 \\
& 2 \cdot \mathbb{E}[xy] + 9 \cdot \mathbb{E}[y] + 2 \cdot \mathbb{E}[y^2]
\end{aligned}$$

This ideal  $\mathbb{I}^{\leq 2}$  contains all algebraic relations that hold among  $\mathbb{E}[x_n]$ ,  $\mathbb{E}[y_n]$ ,  $\mathbb{E}[x_n^2]$ ,  $\mathbb{E}[y_n^2]$  and  $\mathbb{E}[(xy)_n]$  after all number of iterations  $n \in \mathbb{N}_0$ . The ideal provides information about the stochastic process encoded by the loop. For instance, using the basis, it can be automatically checked that

$\mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y]$  is an element of  $\mathbb{I}^{\leq 2}$ . Hence,  $\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y]$  is an invariant, witnessing  $x$  and  $y$  being uncorrelated.

Moment invariant ideals of Definition 5.2 generalize the notion of classical invariant ideals of Definition 2.3 for nonprobabilistic loops. For a program variable  $x$  of a nonprobabilistic loop, the expected value of  $x$  after  $n$  transitions is just the value of  $x$  after  $n$  iterations, that is  $\mathbb{E}[x_n] = x_n$ . Furthermore,  $\mathbb{E}[x_n \cdot y_n] = x_n \cdot y_n$  for all program variables  $x$  and  $y$ . Hence, a moment invariant such as  $\mathbb{E}[x^2]^3 - \mathbb{E}[y]\mathbb{E}[y^2]$  corresponds to the classical invariant  $x^6 - y^3$ . To formalize this observation, we introduce a function  $\psi$  mapping invariants involving moments to classical invariants.

*Definition 5.4 (From Moment Invariants to Invariants).* Let  $\mathcal{P}$  be a program with variables  $x_1, \dots, x_k$ . We define the natural ring homomorphism  $\psi: \overline{\mathbb{Q}}[\mathbb{E}^{\leq \ell}] \rightarrow \overline{\mathbb{Q}}[x_1, \dots, x_k]$  extending  $\psi(\mathbb{E}[M]) := M$ . That means, for all  $p, q \in \overline{\mathbb{Q}}[\mathbb{E}^{\leq \ell}]$  and  $c \in \overline{\mathbb{Q}}$  the function  $\psi$  satisfies the properties (i)  $\psi(p + q) = \psi(p) + \psi(q)$ ; (ii)  $\psi(p \cdot q) = \psi(p) \cdot \psi(q)$ ; and (iii)  $\psi(c \cdot p) = c \cdot \psi(p)$ .

The function  $\psi$  maps polynomials over moments to polynomials over program variables, for example,  $\psi(\mathbb{E}[x^2]^3 - \mathbb{E}[y]\mathbb{E}[y^2]) = \psi(\mathbb{E}[x^2])^3 - \psi(\mathbb{E}[y])\psi(\mathbb{E}[y^2]) = x^6 - y^3$ . If  $p$  is a polynomial moment invariant of a *probabilistic* program,  $\psi(p)$  is in general *not* a classical invariant. However, for nonprobabilistic programs,  $\psi(p)$  is necessarily an invariant for every moment invariant  $p$ , as we show in the next lemma.

**LEMMA 5.5 (MOMENT INVARIANT IDEAL GENERALIZATION).** *Let  $\mathcal{L}$  be a nonprobabilistic loop. Let  $\mathcal{I}$  be the classical invariant ideal and  $\mathbb{I}^{\leq \ell}$  the moment invariant ideal of order  $\ell$ . Then,  $\mathbb{I}^{\leq \ell}$  and  $\mathcal{I}$  are identical under  $\psi$ , that is*

$$\psi(\mathbb{I}^{\leq \ell}) := \{\psi(p) \mid p \in \mathbb{I}^{\leq \ell}\} = \mathcal{I}$$

**PROOF.** We show that  $\psi(\mathbb{I}^{\leq \ell}) \subseteq \mathcal{I}$ . The reasoning for  $\mathcal{I} \subseteq \psi(\mathbb{I}^{\leq \ell})$  is analogous.

Let  $q \in \psi(\mathbb{I}^{\leq \ell})$ . Then, there is a  $p(\mathbb{E}(M^{(1)}), \dots, \mathbb{E}(M^{(m)})) \in \mathbb{I}^{\leq \ell}$  for some monomials in program variables  $M^{(i)}$  such that  $\psi(p) = p(M^{(1)}, \dots, M^{(m)}) = q$ . The polynomial  $p$  in moments of program variables is an invariant because it is an element of  $\mathbb{I}^{\leq \ell}$ . Moreover, because the loop  $\mathcal{L}$  is nonprobabilistic, we have  $\mathbb{E}(M_n) = M_n$  for all number of transitions  $n \in \mathbb{N}_0$  and all monomials  $M$  in program variables<sup>3</sup>. Hence,  $q = p(M^{(1)}, \dots, M^{(m)})$  necessarily is a classical invariant as in Definition 2.1 and therefore  $q \in \mathcal{I}$ .  $\square$

Lemma 5.5 hence proves that Definition 5.2 generalizes the notion of invariant ideals of nonprobabilistic loops.

## 5.2 Computability of Moment Invariant Ideals

We next consider a special class of probabilistic loops, called moment-computable polynomial loops. For such loops, we prove that the bases for moment invariant ideals  $\mathbb{I}^{\leq \ell}$  are computable for any order  $\ell$ . Moreover, in Algorithm 1 we give a decision procedure computing moment invariant ideals of moment-computable polynomial loops.

Let us recall the notion of *moment-computable loops* [Moosbrugger et al. 2022], which we adjusted to our setting of polynomial probabilistic loops.

<sup>3</sup>If the loop contains nondeterministic choice, this property holds with respect to every scheduler resolving nondeterminism. For readability and simplicity, we omit the treatment of schedulers and refer to [Barthe et al. 2020] for details on schedulers.

**Algorithm 1** Computing moment invariant ideals**Input:** A moment-computable polynomial loop  $\mathcal{L}$  and an order  $\ell \in \mathbb{N}$ **Output:** A basis  $B$  for the moment invariant ideal  $\mathbb{I}^{\leq \ell}$ 

▷ Closed forms of moments as exponential polynomials

 $C \leftarrow \text{compute\_closed\_forms}(\mathcal{L}, \mathbb{E}^{\leq \ell})$ ▷ A basis for the ideal of all algebraic relations among sequences in  $C$  $B \leftarrow \text{compute\_algebraic\_relations}(C)$ **return**  $B$ i.e.  $\mathbb{E}(M_n)$  is LRS

*Definition 5.6 (Moment-Computable Polynomial Loops).* A polynomial probabilistic loop  $\mathcal{L}$  is *moment-computable* if, for any monomial  $M$  in loop variables of  $\mathcal{L}$ , we have that  $\mathbb{E}(M_n)$  exists and is computable as  $\mathbb{E}(M_n) = f(n)$ , where  $f(n)$  is an exponential polynomial in  $n$ , describing sums of polynomials multiplied by exponential terms in  $n$ . That is,  $f(n) = \sum_{i=0}^k p_i(n) \cdot \lambda^n$  where all  $p_i \in \mathbb{Q}[n]$  are polynomials and  $\lambda \in \overline{\mathbb{Q}}$ . *is this class decidable?*

As stated in [Kauers and Paule 2011], we note that any LRS (2) has an exponential polynomial as closed form. As proven in [Moosbrugger et al. 2022], when considering loops with affine assignments, probabilistic choice with constant probabilities, and drawing from probability distributions with constant parameters and existing moments, all moments of program variables follow linear recurrence sequences. Moreover, one may also consider polynomial (and not just affine) loop updates such that non-linear dependencies among variables are acyclic. If-statements can also be supported if the loop guards contain only program variables with a finite domain. Under such structural considerations, the resulting probabilistic loops are moment-computable loops [Moosbrugger et al. 2022]: expected values  $\mathbb{E}(M_n)$  for monomials  $M$  over loop variables are exponential polynomials in  $n$ . Furthermore, a basis for the polynomial relations among exponential polynomials is computable [Kauers and Zimmermann 2008]. We thus obtain a decision procedure computing the bases of moment invariant ideals of moment-computable polynomial loops, as given in Algorithm 1 and discussed next.

→ LRS  
are  
closed  
for +, ·

The procedure `compute_closed_form( $\mathcal{L}, S$ )` in Algorithm 1 takes as inputs a moment-computable polynomial loop  $\mathcal{L}$  and a set  $S$  of moments of loop variables and computes exponential polynomial closed forms of the moments in  $S$ ; here, we adjust results of [Moosbrugger et al. 2022] to implement `compute_closed_form( $\mathcal{L}, S$ )`. Further, `compute_algebraic_relations( $C$ )` in Algorithm 1 denotes a procedure that takes a set  $C$  of exponential polynomial closed forms as input and computes a basis for all algebraic relations among them; in our work, we use [Kauers and Zimmermann 2008] to implement `compute_algebraic_relations( $C$ )`. Soundness of Algorithm 1 follows from the soundness arguments of [Kauers and Zimmermann 2008; Moosbrugger et al. 2022]. We implemented Algorithm 1 in our tool called Polar<sup>4</sup>, allowing us to automatically derive the strongest polynomial moment invariants of moment-computable polynomial loops.

*Example 5.7.* Using Algorithm 1 for the probabilistic loop of Example 5.3, we compute a basis for the moment invariant ideal  $\mathbb{I}^{\leq 2}$  in approximately 0.4 seconds and for  $\mathbb{I}^{\leq 3}$  in roughly 0.8 seconds, on a machine with a 2.6GHz Intel i7 processor and 32GB of RAM.

<sup>4</sup><https://github.com/probing-lab/polar>

### 5.3 Hardness for Guarded Probabilistic Loops

As Algorithm 1 provides a decision procedure for moment-computable polynomial loops, a natural question is whether the moment invariant ideals remain computable if we relax

- (C1) the restrictions on the guards,
  - (C2) the structural requirements on the polynomial assignments
- of moment-computable polynomial loops.

We first focus on (C1), that is, lifting the restriction on guards and show that in this case a basis for the moment invariant ideal of any order becomes uncomputable (Theorem 5.8).

We recall the seminal result of [Müller-Olm and Seidl 2004b] proving that the strongest polynomial invariant for *nonprobabilistic* loops with affine updates, nondeterministic choice, and guarded transitions is uncomputable. Interestingly, nondeterministic choice can be replaced by uniform probabilistic choice, allowing us to also establish the uncomputability of the strongest polynomial moment invariants, which means a basis for the ideal  $\mathbb{I}^{\leq \ell}$ , for any order  $\ell$ .

**THEOREM 5.8 (UNCOMPUTABILITY OF MOMENT INVARIANT IDEAL).** *For the class of guarded probabilistic loops with affine updates, a basis for the moment invariant ideal  $\mathbb{I}^{\leq \ell}$  is uncomputable for any order  $\ell$ .*

**PROOF.** The proof is by reduction from Post’s correspondence problem (PCP), which is undecidable [Post 1946]. A PCP instance consists of a finite alphabet  $\Sigma$  and a finite set of tuples  $\{(x_i, y_i) \mid 1 \leq i \leq N, x_i, y_i \in \Sigma^*\}$ . A solution is a sequence of indices  $(i_k)$ ,  $1 \leq k \leq K$  where  $i_k \in \{1, \dots, N\}$  and the concatenations of the substrings indexed by the sequence are identical, written in symbols as

$$x_{i_1} \cdot x_{i_2} \cdot \dots \cdot x_{i_K} = y_{i_1} \cdot y_{i_2} \cdot \dots \cdot y_{i_K}$$

Note that the tuple elements may be of different lengths. Moreover, any instance of the PCP over a finite alphabet  $\Sigma$  can be equivalently represented over the alphabet  $\{0, 1\}$  by a binary encoding.

Now, given an instance of the (binary) PCP, we construct the guarded probabilistic loop with affine updates shown in Figure 3. We encode the binary strings as integers and denote a transition with probability  $pr$ , guard  $g$  and updates  $f$  as  $[pr] : g : \vec{x} \leftarrow f(\vec{x})$ .

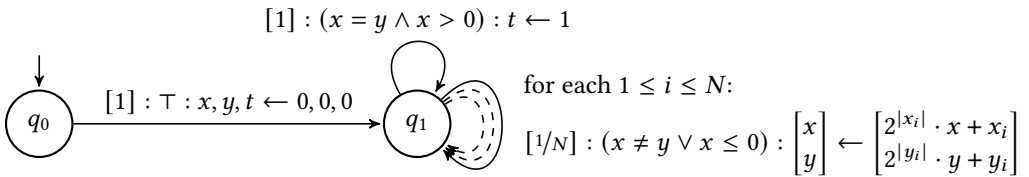


Fig. 3. A guarded probabilistic loop with affine updates simulating the PCP.

The idea is to pick a pair of integer-encoded strings uniformly at random and append them to the string built so far. This is done by left-shifting the existing bits of the string (by multiplying by a power of 2) and adding the randomly selected string.

If the PCP instance does not have a solution, we have  $t = 0$  after every transition. Hence,  $\mathbb{E}[t] = 0$  must be an invariant. Therefore,  $\mathbb{E}[t]$  is necessarily an element of  $\mathbb{I}^{\leq \ell}$  for any order  $\ell$ .

If the PCP instance does have a solution  $(i_k), 1 \leq k \leq K$ , then after exactly  $n = K + 2$  transitions it holds that  $\mathbb{P}(x_n = y_n) \geq (\frac{1}{N})^K$ , as this is the probability of choosing the correct sequence uniformly at random. Because  $t$  is an indicator variable,  $\mathbb{E}[t_n] = \mathbb{P}(t_n = 1) = \mathbb{P}(x_n = y_n) \geq (\frac{1}{N})^K > 0$ . Hence,  $\mathbb{E}[t_n] \neq 0$  after  $n$  transitions and  $\mathbb{E}[t]$  cannot be an element of  $\mathbb{I}^{\leq \ell}$  for any order  $\ell$ .

Consequently, for all orders  $\ell$ , the PCP instance has a solution if and only if  $\mathbb{E}[t]$  is an element of  $\mathbb{I}^{\leq \ell}$ . However, given a basis, checking for ideal membership is decidable (cf. Section 2.2). Hence, a basis for the moment invariant ideal  $\mathbb{I}^{\leq \ell}$  must be uncomputable for any order  $\ell$ .  $\square$

Note that the PCP reduction within the proof of Theorem 5.8 requires only affine updates and affine invariants. Therefore, allowing loop guards renders even the problem of finding the strongest affine invariant for a finite set of moments uncomputable for probabilistic loops with affine updates.

#### 5.4 Hardness for Unguarded Polynomial Probabilistic Loops

In this section we address challenge (C2), that is, study computational lower bounds for computing a basis of moment invariant ideals for probabilistic loops that lack guards and nondeterminism, but feature arbitrary polynomial updates. We show that addressing (C2) boils down to solving the **PROB-SPINV** problem of Section 1, which in turn we prove to be **SKOLEM**-hard (Theorem 5.10). As such, computing the moment invariant ideals of probabilistic loops with arbitrary polynomial updates as stated in (C2) is **SKOLEM**-hard.

We restrict our attention to moment invariant ideals of order 1. Intuitively, a basis for  $\mathbb{I}^{\leq 1}$  is easier to compute than  $\mathbb{I}^{\leq \ell}$  for  $\ell > 1$ . A formal justification in this respect is given by the following lemma.

**LEMMA 5.9 (MOMENT INVARIANT IDEAL OF ORDER 1).** *Given a basis for the moment invariant ideal  $\mathbb{I}^{\leq \ell}$  for any order  $\ell \in \mathbb{N}$ , a basis for  $\mathbb{I}^{\leq 1}$  is computable.*

**PROOF.** The moment invariant ideal  $\mathbb{I}^{\leq \ell}$  is an ideal in the polynomial ring with variables  $\mathbb{E}^{\leq \ell}$ . Moreover,  $\mathbb{E}^{\leq 1} \subseteq \mathbb{E}^{\leq \ell}$ . Hence,  $\mathbb{I}^{\leq 1} = \mathbb{I}^{\leq \ell} \cap \overline{\mathbb{Q}}[\mathbb{E}^{\leq 1}]$ , meaning  $\mathbb{I}^{\leq 1}$  is an elimination ideal of  $\mathbb{I}^{\leq \ell}$ . Given a basis for a polynomial ideal, bases for elimination ideals are computable [Cox et al. 1997].  $\square$

Using Lemma 5.9, we translate challenge (C2) into the **PROB-SPINV** problem of Section 1, formally defined as follows.

The **PROB-SPINV** Problem: Given an unguarded, probabilistic loop with polynomial updates and without nondeterministic choice, compute a basis of the moment invariant ideal of order 1.

Recall that computing a basis for the classical invariant ideal for nonprobabilistic programs with arbitrary polynomial updates, that is, deciding **SPINV**, is **SKOLEM**-hard (Theorem 3.3 and Theorem 4.2). We next show that **SPINV** reduces to **PROB-SPINV**, thus implying **SKOLEM**-hardness of **PROB-SPINV** as a direct consequence of Lemma 5.5.

**THEOREM 5.10 (HARDNESS OF **PROB-SPINV**).** ***PROB-SPINV** is at least as hard as **SPINV**, in symbols  $\mathbf{SPINV} \leq \mathbf{PROB-SPINV}$ .*

**PROOF.** Assume  $\mathcal{L}$  is an instance of **SPINV**. That is,  $\mathcal{L}$  is a deterministic loop with polynomial updates. Let  $x_1, \dots, x_k$  be the program variables and  $\mathcal{I}$  the classical invariant ideal of  $\mathcal{L}$ . Note that  $\mathcal{L}$  is also an instance of **PROB-SPINV** and assume  $B$  is a basis for the moment invariant ideal  $\mathbb{I}^{\leq 1}$ .



From Lemma 5.5 we know that  $\psi(\mathbb{I}^{\leq 1}) = \mathcal{I}$ . For order 1, the function  $\psi$  is a ring isomorphism between the polynomial rings  $\overline{\mathbb{Q}}[x_1, \dots, x_k]$  and  $\overline{\mathbb{Q}}[\mathbb{E}[x_1], \dots, \mathbb{E}[x_k]]$ . Hence, the set  $\{\psi(b) \mid b \in B\}$  is a basis for  $\mathcal{I}$ . Therefore, given a basis for  $\mathbb{I}^{\leq 1}$ , a basis for  $\mathcal{I}$  is computable.  $\square$

Theorem 5.10 shows that **PROB-SPINV** is at least as hard as the **SPINV** problem. Together with Theorem 3.3 and Theorem 4.2, we conclude the following chain of reductions:

$$\text{SKOLEM} \leq \text{P2P} \leq \text{SPINV} \leq \text{PROB-SPINV}.$$

**On attempting to prove uncomputability of **PROB-SPINV** – A remaining open challenge.** While Theorem 5.10 asserts that **PROB-SPINV** is **SKOLEM**-hard, it could be that **PROB-SPINV** is uncomputable.

Recall that for proving the uncomputability of moment invariant ideals for guarded probabilistic programs in Theorem 5.8, we replaced nondeterministic choice with probabilistic choice. The “nondeterministic version” of **PROB-SPINV** refers to computing the strongest polynomial invariant for nondeterministic polynomial programs, which has been recently established as uncomputable [Hrushovski et al. 2023]. Therefore, it is natural to consider transferring the uncomputability results of [Hrushovski et al. 2023] to **PROB-SPINV** by replacing nondeterministic choice with probabilistic choice. However, such a generalization of [Hrushovski et al. 2023] to the probabilistic setting poses considerable problems and ultimately fails to establish the potential uncomputability of **PROB-SPINV**, for the reasons discussed next.

The proof in [Hrushovski et al. 2023] reduces the Boundedness problem for Reset Vector Addition System with State (VASS) to the problem of finding the strongest polynomial invariant for nondeterministic polynomial programs. A Reset VASS is a nondeterministic program where any transition may increment, decrement, or reset a vector of unbounded, non-negative variables. Importantly, a transition can *only be executed if no zero-valued variable is decremented*. The *Boundedness Problem for Reset VASS* asks, given a Reset VASS and a specific program location, whether the set of reachable program configurations is finite. The Boundedness Problem for Reset VASS is undecidable [Dufourd et al. 1998] and therefore instrumental in the reduction of [Hrushovski et al. 2023].

Namely, in the reduction of [Hrushovski et al. 2023] to prove uncomputability of the strongest polynomial invariant for nondeterministic polynomial programs, an arbitrary Reset VASS  $\mathcal{V}$  with  $n$  variables  $a_1, \dots, a_n$  is simulated by a nondeterministic polynomial program  $\mathcal{P}$  with  $n+1$  variables  $b_0, \dots, b_n$ . Note that the programming model is purely nondeterministic, that is, without equality guards, since introducing guards would render the problem immediately undecidable [Müller-Olm and Seidl 2004b]. To avoid zero-testing the variables before executing a transition, the crucial point in the reduction of [Hrushovski et al. 2023] is to map invalid traces to the vector  $\vec{0}$  and faithfully simulate valid executions. By properties of the reduction, it holds that the configuration  $(b_0, \dots, b_n)$  is reachable in  $\mathcal{P}$ , if and only if there exists a corresponding configuration  $1/b_0 \cdot (b_1, \dots, b_n)$  in  $\mathcal{V}$ . Essential to the reduction of [Hrushovski et al. 2023] is, that even though there may be multiple configurations in  $\mathcal{P}$  for each configuration in  $\mathcal{V}$ , all these configurations are only scaled by the factor  $b_0$  and hence collinear. By collinearity, the variety of the invariant ideal can be covered by a finite set of lines if and only if the set of reachable VASS configurations is finite. Testing this property is decidable, and hence finding the invariant ideal must be undecidable.

Transferring the reduction of [Hrushovski et al. 2023] to the probabilistic setting of **PROB-SPINV** by replacing nondeterministic choice with probabilistic choice poses the following problem: in the nondeterministic setting, any path is independent of all other paths. However, this does not



hold in the probabilistic setting of **PROB-SPIINV**. The expected value operator  $\mathbb{E}[x_n]$  aggregates all possible valuations of  $x$  in iteration  $n$  across all possible paths through the program. Specifically, the expected value is a linear combination of the possible configurations of  $\mathcal{V}$ , which is not necessarily limited to a collection of lines but may span a higher-dimensional subspace. This is the step where a reduction similar to [Hrushovski et al. 2023] fails for the **PROB-SPIINV** problem of probabilistic programs.

It is however worth noting how well-suited the Boundedness Problem for Reset VASS is for proving the undecidability of problems for unguarded programs. A Reset VASS is not powerful enough to determine if a variable is zero, yet the Boundedness Problem is still undecidable. The vast majority of other undecidable problems that may be used in a reduction are formulated in terms of counter-machines, Turing machines, or other automata that rely on explicitly determining if a given variable is zero, hindering a straightforward simulation as unguarded programs. Therefore, we conjecture that any attempt towards proving (un)computability of **PROB-SPIINV** would require a new methodology, unrelated to [Hrushovski et al. 2023]. We leave this task as an open challenge for future work.

### 5.5 Summary of Computability Results for Probabilistic Polynomial Loop Invariants

We finally conclude this section by summarizing our computability results on the strongest polynomial (moment) invariants of probabilistic loops. We overview our results in Table 2.

Program Model			Strongest Affine Invariant	Strongest Polynomial Invariant
Prob.	Unguarded & Guarded (finite)	Affine	✓ Algorithm 1	✓ Algorithm 1
		Poly.	?	<b>SKOLEM</b> -hard Theorem 5.10
	Guarded ( $=, <$ )	Affine	✗ Theorem 5.8	
		Poly.		

Table 2. Our computability results for strongest polynomial (moment) invariants of polynomial *probabilistic* loops. The symbol ‘✓’ denotes computable problems, ‘?’ shows open problems, and ‘✗’ marks uncomputable problems.

## 6 RELATED WORK

We discuss our work in relation to the state-of-the-art in computing strongest (probabilistic) invariants and analyzing point-to-point reachability.

**Strongest Invariants.** Algebraic invariants were first considered for unguarded deterministic programs with affine updates [Karr 1976]. Here, a basis for both the ideal of affine invariants and for the ideal of polynomial invariants is computable [Karr 1976; Kovács 2008].

For unguarded deterministic programs with polynomial updates, all invariants of *bounded degree* are computable [Müller-Olm and Seidl 2004a], while the more general task of computing a basis for the ideal of *all* polynomial invariants, that is solving our **SPIINV** problem, was stated as an open problem. In Section 4 we proved that **SPIINV** is at least as hard as **SKOLEM** and **P2P**. Strengthening these results by proving computability for **SPIINV** would result in major breakthroughs in number theory [Bilu et al. 2022; Lipton et al. 2022].

For guarded deterministic programs, the strongest affine invariant is uncomputable, even for programs with only affine updates. This is a direct consequence of the fact that this model is sufficient to encode Turing machines and allows us to encode the Halting problem [Hopcroft and

Ullman 1969]. Nevertheless, there exists a multitude of incomplete methods capable of extracting useful invariants even for non-linear programs, for example, based on abstract domains [Kincaid et al. 2018], over-approximation in combination with recurrences [Farzan and Kincaid 2015; Kincaid et al. 2019] or using consequence finding in tractable logical theories of non-linear arithmetic [Kincaid et al. 2023].

For nondeterministic programs with affine updates, a basis for the invariant ideal is computable [Karr 1976]. Furthermore, the set of invariants of bounded degree is computable for nondeterministic programs with polynomial updates, while bases for the ideal of all invariants are uncomputable [Hrushovski et al. 2023; Müller-Olm and Seidl 2004a]. Additionally, even a single transition guarded by an equality or inequality predicate renders the problem uncomputable, already for affine updates [Müller-Olm and Seidl 2004b].

**Point-To-Point Reachability.** The Point-To-Point reachability problem formalized by our **P2P** problem appears in various areas dealing with discrete systems, such as dynamical systems, discrete mathematics, and program analysis. For linear dynamical systems, **P2P** is known as the *Orbit problem* [Chonev et al. 2013], with a significant amount of work on analyzing and proving decidability of **P2P** for linear systems [Baier et al. 2021; Chonev et al. 2013, 2015; Kannan and Lipton 1980]. In contrast, for polynomial systems, the **P2P** problem remained open regarding decidability or computational lower bounds. Existing techniques in this respect resorted to approximate techniques [Dang and Testylier 2012; Dreossi et al. 2017]. Contrarily to these works, in Section 3 we rigorously proved that **P2P** for polynomial systems is at least as hard as the **SKOLEM** problem. The **P2P** problem is essentially undecidable already for affine systems that additionally include nondeterministic choice [Finkel et al. 2013; Ko et al. 2018].

**Probabilistic Invariants.** Invariants for probabilistic loops can be defined in various incomparable ways, depending on the context and use case. Dijkstra’s weakest-precondition calculus for classical programs was generalized to the weakest-preexpectation (wp) calculus in the seminal works [Kozen 1983, 1985; McIver and Morgan 2005]. In the wp-calculus, the semantics of a loop can be described as the least fixed point of the *characteristic* function of the loop in the lattice of so-called *expectations* [Kaminski et al. 2019]. Invariants are expectations that over- or under-approximate this fixed point and are called super- or sub-invariants, respectively. One line of research is to synthesize such invariants using templates and constraint-solving methods [Batz et al. 2023a, 2021; Gretz et al. 2013]. A calculus, analogous to the wp-calculus, has been introduced for expected runtime analysis [Kaminski et al. 2018] and amortized expected runtime analysis [Batz et al. 2023b]. The work of [Chatterjee et al. 2017] introduces the notion of *stochastic invariants*, that is, expressions that are violated with bounded probability. Other notions of probabilistic invariants involve martingale theory [Barthe et al. 2016] or utilize bounds on the expected value of program variable expressions [Chakarov and Sankaranarayanan 2014]. The techniques presented in [Bartocci et al. 2019; Moosbrugger et al. 2022] compute closed forms for moments of program variables parameterized by the loop counter.

The different notions of probabilistic invariants, in general, do not form ideals or are relative to some other expression. Furthermore, the existing procedures to compute invariants are heuristics-driven and hence incomplete. Contrarily to these, our *polynomial moment invariants* presented in Section 5 form ideals and relate all variables. Moreover, our Algorithm 1 computes a basis for *all* moment invariants and is complete for the class of moment-computable polynomial loops. Going beyond such loops, we showed that **PROB-SPINV** is **SKOLEM**-hard and/or uncomputable (Theorem 5.10 and Theorem 5.8).

## 7 CONCLUSION

We prove that computing the strongest polynomial invariant for single-path loops with polynomial assignments (**SPINV**) is at least as hard as the **SKOLEM** problem, a famous problem whose decidability has been open for almost a century. As such, we provide the first non-trivial lower bound for computing the strongest polynomial invariant for deterministic polynomial loops, a quest introduced in [Müller-Olm and Seidl 2004b]. As an intermediate result, we show that point-to-point reachability in deterministic polynomial loops (**P2P**), or equivalently in discrete-time polynomial dynamical systems, is **SKOLEM**-hard. Further, we devise a reduction from **P2P** to **SPINV**. We generalize the notion of invariant ideals from classical programs to the probabilistic setting, by introducing *moment invariant ideals* and addressing the **PROB-SPINV** problem. We show that the strongest polynomial moment invariant, and hence **PROB-SPINV**, is (i) computable for the class of *moment-computable* probabilistic loops, but becomes (ii) uncomputable for probabilistic loops with branching statements and (iii) **SKOLEM**-hard for polynomial probabilistic loops without branching statements. Going beyond **SKOLEM**-hardness of **PROB-SPINV** and **SPINV** are open challenges we aim to further study.

## ACKNOWLEDGMENTS

This research was supported by the Vienna Science and Technology Fund WWTF 10.47379/ICT19018 grant ProInG, the European Research Council Consolidator Grant ARTIST 101002685, the Austrian Science Fund (FWF) project W1255-N23, and the SecInt Doctoral College funded by TU Wien. We thank Manuel Kauers for providing details on sequences and algebraic relations and Toghrul Karimov for inspiring us to consider the orbit problem. We thank the McGill Bellairs Research Institute for hosting the Bellairs 2023 workshop, whose fruitful discussions influenced parts of this work.

## REFERENCES

- Christel Baier, Florian Funke, Simon Jantsch, Toghrul Karimov, Engel Lefauchaux, Florian Luca, Joël Ouaknine, David Purser, Markus A. Whiteland, and James Worrell. 2021. The Orbit Problem for Parametric Linear Dynamical Systems. In *Proc. of CONCUR*. <https://doi.org/10.4230/LIPIcs.CONCUR.2021.28>
- Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016. Synthesizing Probabilistic Invariants via Doob’s Decomposition. In *Proc. of CAV*. [https://doi.org/10.1007/978-3-319-41528-4\\_3](https://doi.org/10.1007/978-3-319-41528-4_3)
- Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2012a. Probabilistic Relational Hoare Logics for Computer-Aided Security Proofs. In *Proc. of MPC*. <https://doi.org/10.1007/978-3-642-31113-0>
- Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. 2020. *Foundations of Probabilistic Programming*. Cambridge University Press. <https://doi.org/10.1017/9781108770750>
- Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012b. Probabilistic Relational Reasoning for Differential Privacy. In *Proc. of POPL*. <https://doi.org/10.1145/2103656.2103670>
- Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. 2019. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops. In *Proc. of ATVA*. [https://doi.org/10.1007/978-3-030-31784-3\\_15](https://doi.org/10.1007/978-3-030-31784-3_15)
- Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023a. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In *Proc. of TACAS*. [https://doi.org/10.1007/978-3-031-30820-8\\_25](https://doi.org/10.1007/978-3-031-30820-8_25)
- Kevin Batz, Mingshuai Chen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Philipp Schröder. 2021. Latticed k-Induction with an Application to Probabilistic Programs. In *Proc. of CAV*. [https://doi.org/10.1007/978-3-030-81688-9\\_25](https://doi.org/10.1007/978-3-030-81688-9_25)
- Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. 2023b. A Calculus for Amortized Expected Runtimes. *Proc. ACM Program. Lang.* POPL (2023). <https://doi.org/10.1145/3571260>
- Yuri Bilu, Florian Luca, Joris Nieuwveld, Joël Ouaknine, David Purser, and James Worrell. 2022. **Skolem Meets Schanuel**. In *Proc. of MFCS*. <https://doi.org/10.4230/LIPIcs.MFCS.2022.20>
- Bruno Buchberger. 2006. Bruno Buchberger’s PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *J. Symb. Comput.* (2006). <https://doi.org/10.1016/j.jsc.2005.09.007>
- Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michal Pilipczuk, and Géraud Sénizergues. 2020. On Polynomial Recursive Sequences. In *Proc. of ICALP*. <https://doi.org/10.4230/LIPIcs.ICALP.2020.117>

- Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *Proc. of SAS*. [https://doi.org/10.1007/978-3-319-10936-7\\_6](https://doi.org/10.1007/978-3-319-10936-7_6)
- Krishnendu Chatterjee, Petr Novotný, and Dord Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proc. of POPL*. <https://doi.org/10.1145/3009837.3009873>
- Ventsislav Chonev, Joël Ouaknine, and James Worrell. 2013. The orbit problem in higher dimensions. In *Proc. of STOC*. <https://doi.org/10.1145/2488608.2488728>
- Ventsislav Chonev, Joël Ouaknine, and James Worrell. 2015. The Polyhedron-Hitting Problem. In *Proc. of SODA*. <https://doi.org/10.1137/1.9781611973730.64>
- David A. Cox, John Little, and Donal O'Shea. 1997. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra*. <https://doi.org/10.1137/1035171>
- Thao Dang and Romain Testylier. 2012. Reachability Analysis for Polynomial Dynamical Systems Using the Bernstein Expansion. *Reliab. Comput.* (2012).
- Tommaso Dreossi, Thao Dang, and Carla Piazza. 2017. Reachability computation for polynomial dynamical systems. *Formal Methods Syst. Des.* (2017).
- Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. 1998. Reset Nets Between Decidability and Undecidability. In *Proc. of ICALP*. <https://doi.org/10.1007/BFb0055044>
- Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. 2003. *Recurrence Sequences*. American Mathematical Society. ISBN 978-0-8218-3387-2.
- Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *FMCAD*. <https://doi.org/10.1109/FMCAD.2015.7542253>
- Alain Finkel, Stefan Göller, and Christoph Haase. 2013. Reachability in Register Machines with Polynomial Updates. In *Proc. of MFCS*. [https://doi.org/10.1007/978-3-642-40313-2\\_37](https://doi.org/10.1007/978-3-642-40313-2_37)
- Zoubin Ghahramani. 2015. Probabilistic Machine Learning and Artificial Intelligence. *Nature* (2015). <https://doi.org/10.1038/nature14541>
- Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2013. Prinsys - On a Quest for Probabilistic Loop Invariants. In *Proc. of QEST*. [https://doi.org/10.1007/978-3-642-40196-1\\_17](https://doi.org/10.1007/978-3-642-40196-1_17)
- John E. Hopcroft and Jeffrey D. Ullman. 1969. *Formal languages and their relation to automata*.
- Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2018. Polynomial Invariants for Affine Programs. In *Proc. of LICS*. <https://doi.org/10.1145/3209108.3209142>
- Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2023. On Strongest Algebraic Program Invariants. (2023). [https://people.mpi-sws.org/~joel/publications/strongest\\_algebraic\\_invariants23abs.html](https://people.mpi-sws.org/~joel/publications/strongest_algebraic_invariants23abs.html) To appear in Journal of the ACM.
- Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2019. On the hardness of analyzing probabilistic programs. *Acta Inform.* (2019). <https://doi.org/10.1007/s00236-018-0321-1>
- Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* (2018). <https://doi.org/10.1145/3208102>
- Ravindran Kannan and Richard J. Lipton. 1980. The Orbit Problem is Decidable. In *Proc. of STOC*. <https://doi.org/10.1145/800141.804673>
- Toghrul Karimov, Engel Lefauchaux, Joël Ouaknine, David Purser, Anton Varonka, Markus A. Whiteland, and James Worrell. 2022. What's decidable about linear loops? *Proc. ACM Program. Lang.* POPL (2022). <https://doi.org/10.1145/3498727>
- Michael Karr. 1976. Affine Relationships Among Variables of a Program. *Acta Inform.* (1976). <https://doi.org/10.1007/BF00268497>
- Manuel Kauers. 2005. *Algorithms for Nonlinear Higher Order Difference Equations*. Ph. D. Dissertation. RISC, Johannes Kepler University, Linz.
- Manuel Kauers and Peter Paule. 2011. *The Concrete Tetrahedron - Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. Springer. <https://doi.org/10.1007/978-3-7091-0445-3>
- Manuel Kauers and Burkhard Zimmermann. 2008. Computing the algebraic relations of C-finite sequences and multisequences. *J. Symb. Comput.* (2008).
- Zachary Kincaid, Jason Breck, John Cyphert, and Thomas W. Reps. 2019. Closed forms for numerical loops. *Proc. ACM Program. Lang.* POPL (2019). <https://doi.org/10.1145/3290368>
- Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.* POPL (2018). <https://doi.org/10.1145/3158142>
- Zachary Kincaid, Nicolas Koh, and Shaowei Zhu. 2023. When Less Is More: Consequence-Finding in a Weak Theory of Arithmetic. *Proc. ACM Program. Lang.* POPL (2023). <https://doi.org/10.1145/3571237>
- Sang-Ki Ko, Reino Niskanen, and Igor Potapov. 2018. Reachability Problems in Nondeterministic Polynomial Maps on the Integers. In *Proc. of DLT*. [https://doi.org/10.1007/978-3-319-98654-8\\_38](https://doi.org/10.1007/978-3-319-98654-8_38)

- Andrey Kofnov, Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Efstathia Bura. 2022. Moment-Based Invariants for Probabilistic Loops with Non-polynomial Assignments. In *Proc. of QEST*. [https://doi.org/10.1007/978-3-031-16336-4\\_1](https://doi.org/10.1007/978-3-031-16336-4_1)
- Laura Kovács. 2008. Reasoning Algebraically About P-Solvable Loops. In *Proc. of TACAS*. [https://doi.org/10.1007/978-3-540-78800-3\\_18](https://doi.org/10.1007/978-3-540-78800-3_18)
- Laura Kovács and Anton Varonka. 2023. What Else is Undecidable About Loops?. In *Proc. of RAMiCS*. [https://doi.org/10.1007/978-3-031-28083-2\\_11](https://doi.org/10.1007/978-3-031-28083-2_11)
- Dexter Kozen. 1983. A Probabilistic PDL. In *Proc. of STOC*. <https://doi.org/10.1145/800061.808758>
- Dexter Kozen. 1985. A Probabilistic PDL. *J. Comput. Syst. Sci.* (1985). [https://doi.org/10.1016/0022-0000\(85\)90012-1](https://doi.org/10.1016/0022-0000(85)90012-1)
- Richard Lipton, Florian Luca, Joris Nieuwveld, Joël Ouaknine, David Purser, and James Worrell. 2022. **On the Skolem Problem and the Skolem Conjecture**. In *Proc. of LICS*. <https://doi.org/10.1145/3531130.3533328>
- Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. <https://doi.org/10.1007/b138392>
- Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Laura Kovács. 2022. **This is the moment for probabilistic loops**. *Proc. ACM Program. Lang.* OOPSLA2 (2022). <https://doi.org/10.1145/3563341>
- Markus Müller-Olm and Helmut Seidl. 2004a. Computing polynomial program invariants. *Inf. Process. Lett.* (2004). <https://doi.org/10.1016/j.ipl.2004.05.004>
- Markus Müller-Olm and Helmut Seidl. 2004b. **A Note on Karr's Algorithm**. In *Proc. of ICALP*. [https://doi.org/10.1007/978-3-540-27836-8\\_85](https://doi.org/10.1007/978-3-540-27836-8_85)
- Emil L. Post. 1946. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.* (1946).
- Terrence Tao. 2008. *Structure and Randomness*. American Mathematical Society. ISBN 0-8218-4695-7.