Inf. Comput. 312 (2026) 105487

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

On termination of polynomial programs with equality conditions

Yangjia Li [a,b], Mingshuai Chen [c,*], Liangran Zhao [d], Naijun Zhan [d,*], Hui Lu [e], Guohua Wu [f], Joost-Pieter Katoen [g]

[a] Key Laboratory of System Software (Chinese Academy of Sciences), Institute of Software, Chinese Academy of Sciences, Beijing, China
[b] University of Chinese Academy of Sciences, Beijing, China
[c] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[d] School of Computer Science & Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China
[e] Nanjing Audit University, Nanjing, China
[f] Nanyang Technological University, Singapore
[g] RWTH Aachen University, Aachen, Germany

ARTICLE INFO

Keywords:
Polynomial programs
Termination analysis
Hilbert ascending chains

ABSTRACT

We investigate the termination problem of a family of multi-path polynomial programs (MPPs) over an effective field $\mathbb{K}$, in which all assignments to program variables are polynomials, and test conditions of loops and conditional statements are polynomial equalities. We show that the set of non-terminating inputs (NTI) of such a program is algorithmically computable, which in turn yields the decidability of its termination on a given input – and that on a semi-algebraic set of inputs when $\mathbb{K}$ is $\mathbb{R}$. To the best of our knowledge, the considered family of MPPs is hitherto the largest fragment of nonlinear programs for which termination is decidable. We present an explicit recursive function, essentially of Ackermannian growth, to compute the maximal length of ascending chains of polynomial ideals under a control function, thereby providing a complete answer to the questions raised by Seidenberg [1]. This maximal length facilitates a precise complexity analysis of our algorithms for computing the NTI and deciding termination of MPPs. We further extend our approach to programs with polynomial guarded commands and show how an incomplete procedure for MPPs with inequality guards can be obtained. Finally, we show that our decidability result gives rise to a complete method for computing all polynomial equality invariants (of a fixed degree) of polynomial programs.

1. Introduction

Termination analysis [2,3] has been a long-standing, active field of research due to its great importance in program verification. However, the termination problem of programs is equivalent to the well-known halting problem [4], and hence is undecidable in general. A complete method for termination analysis for programs, even for linear or polynomial programs, is therefore impossible [5–8]. Consequently, a practical way is to provide sufficient conditions for termination and/or nontermination. The classical method for establishing termination of a program uses *ranking functions* that map the state space of the program to a well-founded domain, thus providing a sufficient condition for termination, see, e.g., [9–14]. In [15], Gupta et al. presented a sufficient condition for identifying

* Corresponding authors.
E-mail addresses: yangjia@ios.ac.cn (Y. Li), m.chen@zju.edu.cn (M. Chen), 2100013014@stu.pku.edu.cn (L. Zhao), njzhan@pku.edu.cn (N. Zhan), luhui@nau.edu.cn (H. Lu), guohua@ntu.edu.sg (G. Wu), katoen@cs.rwth-aachen.de (J.-P. Katoen).

https://doi.org/10.1016/j.ic.2026.105487
Received 6 June 2025; Received in revised form 1 June 2026; Accepted 3 June 2026
Available online 5 June 2026
0890-5401/© 2026 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

non-terminating inputs for linear programs; while in [16], Harris et al. investigated sufficient conditions for finding terminating and non-terminating inputs, as well as techniques for checking the two conditions in parallel.

In contrast, Tiwari [5] noticed that termination of a class of simple linear loops is related to the eigenvalues of the assignment matrix and proved that the termination problem of these linear programs on *all* inputs (i.e., *universal* termination) over the reals $\mathbb{R}$ is decidable. The same line of theory was further developed in [8,17–22]. In particular, Bradley et al. [7] studied the termination problem of a family of polynomial programs modelled as *multi-path polynomial programs* (MPPs) by using *finite difference trees* (FDTs). These MPP programs cover an expressive class of loops with multiple paths, polynomial loop guards and assignments, which enables practical code abstraction and analysis. Bradley et al. (ibid.) proved that the termination problem of MPPs is undecidable. A similar idea was employed in [23] for the termination analysis of polynomial programs. Moreover, Liu et al. [24] provide sufficient conditions for establishing (non)termination of a subclass of MPPs with equational loop guards. This is achieved by under/over-approximating the set of non-terminating inputs. As in [16], termination analysis can be conducted by checking these conditions in parallel. Until now, it remains an open problem whether the termination problem of MPPs with equality guards is decidable.

In this paper, we answer the question affirmatively: we show that the problem of termination (on a given input) of MPPs with equality guards is decidable. To the best of our knowledge, this family of (multi-path) polynomial programs should be the largest fragment of nonlinear programs with a decidable termination problem so far: Any extension of it by allowing inequalities or inequations ($\neq$) in the guards may render the termination problem undecidable. Note that program termination with inequality (resp. inequation) conditions is hard to decide even for simple linear loops, since it is equivalent to the Positivity (resp. Skolem) problem that remains open; see Ouaknine and Worrell's results in [25]. Our approach works as follows: Given an MPP P with ℓ paths, for any input $\boldsymbol{x} \in \mathbb{R}^d$, if in the first iteration $\boldsymbol{x}$ satisfies the loop guard, then one of the paths in the loop body will be nondeterministically selected and the corresponding assignment will be used to update the value of $\boldsymbol{x}$. This results in ℓ possible values of $\boldsymbol{x}$. Afterwards, the above procedure is repeated until no guard holds anymore. Thus the execution of an MPP on input $\boldsymbol{x} \in \mathbb{R}^d$ forms a tree. An input $\boldsymbol{x}$ is *non-terminating* if the execution tree on $\boldsymbol{x}$ has an infinite path. From an algebraic perspective, each of such paths forms an ascending chain of polynomial ideals, and an input $\boldsymbol{x}$ is non-terminating iff $\boldsymbol{x}$ is in the variety of the ascending chain. This can be determined in finitely many steps due to Hilbert's basis theorem [26] guaranteeing that such an ascending chain is always finite. The difficulty is that the execution tree may contain *infinitely many* paths of distinct lengths, whereas a uniform upper bound of them, say k, is required. We show how to obtain such a bound such that we only need to symbolically execute the loop *at most* k *steps* and check if the given input is a solution to the ideal formed by some path in the induced finite execution tree.

The main technique we propose to decide termination – including an explicit complexity analysis – of MPPs is to compute an upper bound on the length of the longest path in their execution tree. This is reduced to computing the maximal length of the Hilbert ascending chains under specific conditions. Briefly speaking, a Hilbert ascending chain is of the form $I_0 \subset I_1 \subset \cdots \subset I_n \subset \cdots$, where each I_n is an ideal of the polynomial ring (over program variables, in our setting). Seidenberg studied in [1,27] how to obtain an upper bound on these lengths under a *control function* f, such that for each n, $f(n)$ is an upper bound on the degrees of polynomials in some basis of I_n. Seidenberg (ibid.) proved the existence of such a bound in terms of f and the number d of indeterminates of the polynomial ring. Two questions were posed in [1]:

- Explicitness: *"[…] it would be desirable to bring to a more satisfactory expression […]"*;
- Complexity: *"Even for $d \geq 3$, where primitive recursiveness looks doubtful […]"*.

The first question asks for a more explicit expression of the bound, since the original one is too complex to compute. The second question is on the complexity of the length, which was conjectured not to be primitive recursive. Our techniques answer both questions in a constructive way.

Contributions. The main contributions of this work are as follows.

- We construct a Hilbert ascending chain of the maximal length, and express its length (i.e., the optimal upper bound) as a recursive function $L(d, f)$ w.r.t. the control function f and the number of indeterminates d. As a direct consequence of the expression, the maximal length is shown Ackermannian, i.e., not primitive recursive. This answers the complexity conjectured by Seidenberg in [1].
- By leveraging the result on the maximal length of Hilbert ascending chains, we prove decidability of the termination problem of multi-path polynomial programs with equality conditions. Algorithms for computing the set of non-terminating inputs (and thus deciding termination for a single input) are proposed and demonstrated on a collection of examples from the literature. An explicit complexity analysis of the decision algorithms is presented in terms of the function $L(d, f)$. The decidability result is further extended to programs with polynomial guarded commands, and is applied to obtaining an incomplete procedure for MPPs with inequality guards.
- We investigate the problem of discovering loop invariants for polynomial programs, which is another fundamental problem in program verification. As an application of the decidability result on termination, a constraint-solving algorithm is proposed to compute, for every positive integer k, all polynomial equalities of degree k that are invariants of the program. This yields, to the best of our knowledge, the first complete method for computing all degree-k polynomial invariants.

We provide a detailed discussion on the relation between our results and related work in Section 9.

Paper structure. Section 2 gives an overview of our approach through an example. Section 3 recaps preliminaries on computational algebraic geometry. Section 4 is dedicated to explicitly computing the length of a strictly Hilbert ascending chain of polynomial

ideals in the worst case. In Section 5, we introduce the model of MPPs with equality guards. In Section 6, we show the decidability of termination for such MPPs and propose decision algorithms to compute the set of non-terminating inputs. Section 7 extends the method to more general polynomial programs and the related invariant generation problem. Section 8 reports our experimental results. After the discussion of related work in Section 9, we conclude the paper in Section 10.

2. An illustrative example

We use the following example to give a bird's-eye view of our approach.

Example 1 (`overview`)**.**

$(x, y) := (x_0, y_0)$;
WHILE $(x + y = 0)$ DO
  IF ? THEN $(x, y) := (y^2, 2x + y)$;
  ELSE $(x, y) := (2x^2 + y - 1, x + 2y + 1)$;
END WHILE

Here "?" indicates that the condition takes a nondeterministic Boolean value, and thus in each iteration one of the two assignments is *nondeterministically* selected. Our interest is to decide whether for any initial value (x_0, y_0) in the set[1] $V = \{(x, y) \mid x^2 + y = 0\}$, the program terminates.

For simplicity, the polynomial of the loop guard is denoted as $G(x, y) \triangleq x + y$, and the two polynomial vectors of the assignments as $\mathbf{A}_1(x, y) = (y^2, 2x + y)$ and $\mathbf{A}_2(x, y) = (2x^2 + y - 1, x + 2y + 1)$. Our approach is to compute the set D of all possible initial values of (x_0, y_0) on which the program may not terminate (a.k.a., diverge). Thus, the termination problem of the program on the input set V can be solved by checking whether $V \cap D = \emptyset$. The procedure is outlined as follows.

1. Let D_0 be solutions of $G(x, y) = x + y = 0$. $(x_0, y_0) \in D_0$ thus means the loop body is executed at least once.
2. Denote respectively by $D_{(1)}$ and $D_{(2)}$ the solution set of equations:

$$\begin{cases} G(x, y) = x + y = 0 \\ G(\mathbf{A}_1(x, y)) = y^2 + (2x + y) = 0 \end{cases} \quad \text{and} \quad \begin{cases} G(x, y) = x + y = 0 \\ G(\mathbf{A}_2(x, y)) = (2x^2 + y - 1) + (x + 2y + 1) = 0 \end{cases}$$

   Analogously, $(x_0, y_0) \in D_{(i)}$ $(i \in \{1, 2\})$ indicates that the loop body is executed at least twice by correspondingly choosing $\mathbf{A}_i$ to be the assignment in the first iteration. Hence, $D_1 \triangleq D_{(1)} \cup D_{(2)}$ collects all inputs for which there exists an execution path of length at least 2. It is easy to calculate that $D_{(1)} = \{(0, 0), (-1, 1)\}$, $D_{(2)} = \{(0, 0), (1, -1)\}$, and thus $D_1 = \{(0, 0), (1, -1), (-1, 1)\}$.
3. Similarly, the solution set $D_{(ij)}$ $(i, j \in \{1, 2\})$ of equation

$$\begin{cases} G(x, y) = 0 \\ G(\mathbf{A}_i(x, y)) = 0 \\ G(\mathbf{A}_j(\mathbf{A}_i(x, y))) = 0 \end{cases}$$

   is the set of inputs on which the third iteration is achievable by successively choosing $\mathbf{A}_i$ and $\mathbf{A}_j$ in the first and second iterations. $D_2 \triangleq D_{(11)} \cup D_{(12)} \cup D_{(21)} \cup D_{(22)}$ is the set of inputs that admit at least three iterations. By simple calculation, we obtain $D_{(11)} = \{(0, 0)\}$, $D_{(12)} = \{(0, 0), (-1, 1)\}$, $D_{(21)} = \{(0, 0), (1, -1)\}$, $D_{(22)} = \{(1, -1)\}$, and $D_2 = \{(0, 0), (1, -1), (-1, 1)\}$.
4. Observe that $D_1 = D_2$. Our results in this paper guarantee that $D = D_1 = \{(0, 0), (1, -1), (-1, 1)\}$, i.e., D_1 is the set of inputs which make the program diverging.
5. Note that $V \cap D = \{(1, -1)\} \neq \emptyset$, yielding that the program is diverging on input $(x_0, y_0) = (1, -1)$.

The upper bound given in Section 4 on the length of polynomial ascending chains (i.e., Hilbert ascending chains of polynomial ideals) guarantees that we only need to symbolically execute the loop for finitely many steps and check whether the given input is a solution to the ideal formed by some path in the induced finite execution tree.

3. Preliminaries

3.1. Polynomial rings and ideals

A monomial of $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ is of the form $\boldsymbol{x}^{\boldsymbol{\alpha}} \triangleq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$, where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_d) \in \mathbb{N}^d$ is a vector of natural numbers, and $\deg(\boldsymbol{x}^{\boldsymbol{\alpha}}) \triangleq \sum_{i=1}^{d} \alpha_i$ is called the *degree* of $\boldsymbol{x}^{\alpha}$. A monomial $m_1 = \boldsymbol{x}^{\boldsymbol{\alpha}}$ is a *factor* of a monomial $m_2 = \boldsymbol{x}^{\boldsymbol{\beta}}$ (or equivalently, m_2 is a *multiple* of m_1), denoted by $m_1 | m_2$, if $\alpha_i \leq \beta_i$ for $i = 1, 2, \ldots, d$. A polynomial f of $\boldsymbol{x}$ is a linear combination of a finite number of monomials over a field $\mathbb{K}$, i.e., $f = \sum_{i=1}^{m} \lambda_i \boldsymbol{x}^{\boldsymbol{\alpha}_i}$, where m is the number of distinct monomials of f and $\lambda_i \in \mathbb{K}$ is the nonzero coefficient of $\boldsymbol{x}^{\boldsymbol{\alpha}_i}$,

[1] The initial set V can be arbitrary, since our approach computes the whole set of non-terminating inputs.

for each i. In this article, $\mathbb{K}$ can be an arbitrary field, including the field $\mathbb{Q}$ of rational numbers, $\overline{\mathbb{Q}}$ of algebraic complex numbers, $\mathbb{R}$ of real numbers, $\mathbb{C}$ of complex numbers, the field of rational functions, and even a finite field. The degree of $f(\boldsymbol{x})$ is defined as $\deg(f) \triangleq \max\{\deg(\boldsymbol{x}^{\boldsymbol{\alpha}_i}) \mid i = 1, 2, \ldots, m\}$. Denote by $\mathcal{M}[\boldsymbol{x}]$ the set of monomials of $\boldsymbol{x}$ and $\mathbb{K}[\boldsymbol{x}]$ the polynomial ring of $\boldsymbol{x}$ over $\mathbb{K}$. The degree of a finite set $X \subseteq \mathbb{K}[\boldsymbol{x}]$ is defined as $\deg(X) \triangleq \max\{\deg(f) \mid f \in X\}$.

We introduce the *lexicographic order* over monomials: $\boldsymbol{x}^{\boldsymbol{\alpha}} \prec \boldsymbol{x}^{\boldsymbol{\beta}}$ if and only if there exists $1 \le i \le d$ such that $\alpha_i < \beta_i$ and $\alpha_j = \beta_j$ for all $1 \le j < i$. For every polynomial $f \in \mathbb{K}[\boldsymbol{x}]$ we write its leading monomial (i.e., the greatest monomial under $\prec$) as $\mathrm{lm}(f)$. For any $n \in \mathbb{N}$, a set of monomials M is called *n-compressed*, if for any $m \in M$ with $\deg(m) = n$, $\{m' \mid \deg(m') = n, m' \succ m\} \subseteq M$.

Definition 1 (Polynomial ideal)**.**

1. A nonempty subset $I \subseteq \mathbb{K}[\boldsymbol{x}]$ is called an *ideal* if $f, g \in I \implies f + g \in I$, and $f \in I, h \in \mathbb{K}[\boldsymbol{x}] \implies fh \in I$.
2. Let $P = \{f_1, f_2, \ldots, f_n\}$ be a nonempty subset of $\mathbb{K}[\boldsymbol{x}]$. The *ideal generated by* P is defined as $\langle P\rangle \triangleq \left\{ \sum_{i=1}^{n} f_i h_i \mid f_i \in P, h_i \in \mathbb{K}[\boldsymbol{x}] \right\}$. P is called a *basis* of $\langle P\rangle$.
3. The *product* of two ideals I and J is defined as
$$I \times J \triangleq \langle \{fg \mid f \in I, g \in J\}\rangle.$$

For a finite set $P = \{f_1, f_2, \ldots, f_n\}$, we simply write $\langle P\rangle$ as $\langle f_1, f_2, \ldots, f_n\rangle$. Given two polynomial sets P and Q, we define $P \cdot Q \triangleq \{fg \mid f \in P, g \in Q\}$. Obviously, $\langle P\rangle \times \langle Q\rangle = \langle P \cdot Q\rangle$.

Hilbert's basis theorem [26, Chapter 7] implies that the polynomial ring $\mathbb{K}[\boldsymbol{x}]$ is a Noetherian ring, that is,

Theorem 1 (Ascending Chain Condition)**.** *For any ascending chain of ideals $I_1 \subseteq I_2 \subseteq \cdots \subseteq I_n \subseteq \cdots$ of $\mathbb{K}[\boldsymbol{x}]$, there exists $N \in \mathbb{N}$ such that $I_n = I_N$ for all $n \ge N$.*

An equivalent statement of Theorem 1 is that every ideal $I \subseteq \mathbb{K}[\boldsymbol{x}]$ is *finitely-generated*, that is, $I = \langle f_1, f_2, \ldots, f_n\rangle$ for some $f_1, f_2, \ldots, f_n \in \mathbb{K}[\boldsymbol{x}]$. Thus, we can define the degree of an ideal I as $\mathrm{gdeg}(I) = \min\{\deg(P) \mid P \text{ is a basis of } I\}$.

Note that an ideal may have different bases. However, using Buchberger's algorithm [28] under a fixed monomial ordering, a *Gröbner basis* $\mathrm{GB}(I)$ of I can be computed from any other basis of I. We simply write $\mathrm{GB}(\langle P\rangle)$ as $\mathrm{GB}(P)$ for any basis P. An important property of such Gröbner basis is that the remainder of any polynomial f on division by $\mathrm{GB}(P)$ – written as $\mathrm{Rem}(f, \mathrm{GB}(P))$, i.e., the remainder obtained by the multivariate polynomial division with respect to the fixed monomial order – satisfies

$$f \in \langle P\rangle \iff \mathrm{Rem}(f, \mathrm{GB}(P)) = 0.$$

Therefore, the membership problem of polynomial ideals over an effective field (i.e., fields with computable arithmetic) $\mathbb{K}$ is decidable by constructing the Gröbner basis. Note that the problem of constructing Gröbner bases is EXPSPACE-complete and can be solved in time of doubly exponential in d when $\mathbb{K} = \mathbb{Q}$ [29,30].

3.2. Algebraic sets and varieties

A polynomial $f \in \mathbb{K}[\boldsymbol{x}]$ can be regarded as a function from the affine space $\mathbb{K}^d$ to $\mathbb{K}$. The set of zeros of a polynomial set $P \subseteq \mathbb{K}[\boldsymbol{x}]$ can be defined as $Z(P) \triangleq \{\boldsymbol{x} \in \mathbb{K}^d \mid \forall f \in P :\ f(\boldsymbol{x}) = 0\}$. Clearly, $Z(P) = Z(\langle P\rangle)$.

Definition 2 (Algebraic Set and Variety)**.** A set $X \subseteq \mathbb{K}^d$ is

1. *algebraic*, if there exists $P \subseteq \mathbb{K}[\boldsymbol{x}]$ with $X = Z(P)$, and P is called a set of *generating polynomials of* X;
2. *reducible*, if it has two proper algebraic subsets X_1 and X_2 such that $X = X_1 \cup X_2$; otherwise *irreducible*;
3. a *variety*, if it is a nonempty irreducible algebraic set.

The following properties on algebraic sets and varieties hold: the union of two algebraic sets is algebraic, i.e., $Z(P) \cup Z(Q) = Z(P \cdot Q)$; the intersection of any family of algebraic sets is algebraic, i.e., $\bigcap_\alpha Z(P_\alpha) = Z(\bigcup P_\alpha)$; and suppose $X_1, X_2, \ldots, X_n$ are algebraic sets and X is a variety, then

$$X \subseteq X_1 \cup \cdots \cup X_n \implies \exists k \in \{1, \ldots, n\} :\ X \subseteq X_k.$$

An algebraic set is usually represented by a (not necessarily unique) set of generating polynomials. By defining

$$I(X) \triangleq \{f \in \mathbb{K}[\boldsymbol{x}] \mid \forall \boldsymbol{x} \in X :\ f(\boldsymbol{x}) = 0\}$$

for any $X \subseteq \mathbb{K}^d$, one can verify that $I(Z(P))$ is the maximal set that generates $Z(P)$. Hence, any algebraic set X can be identified by the ideal $I(X)$. The membership $f \in I(Z(P))$ for any polynomial f and any finite set $P = \{f_1, \ldots, f_n\}$ of polynomials is equivalent to the unsatisfiability of $f_1(\boldsymbol{x}) = 0 \wedge f_2(\boldsymbol{x}) = 0 \wedge \cdots \wedge f_n(\boldsymbol{x}) = 0 \wedge f(\boldsymbol{x}) \ne 0$, which is decidable for $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$, and $f, f_1, \ldots, f_n \in \mathbb{Q}[\boldsymbol{x}]$ due to Tarski's approach [31]. We resort to the construction of Gröbner bases for effective fields beyond $\mathbb{R}$ or $\mathbb{C}$ (see Section 6.2), such that our results on both the decidability of termination and the complexity of deciding termination (see Proposition 1) still apply.

Additionally, considering the fact that $X_1 \subset X_2 \iff I(X_1) \supset I(X_2)$ for any two algebraic sets X_1 and X_2, it follows immediately from Theorem 1 that

Theorem 2 (Descending Chain Condition)**.** *For any descending chain of algebraic sets $X_1 \supseteq X_2 \supseteq \cdots \supseteq X_n \supseteq \cdots$ of $\mathbb{K}^d$, there is $N \in \mathbb{N}$ with $X_n = X_N$ for all $n \ge N$.*

3.3. Monomial ideals and Hilbert's functions

An ideal I is called monomial if it can be generated by a set of monomials. A monomial ideal always has a finite basis $\{m_1, m_2, \ldots, m_s\}$ of monomials (by Dickson's Lemma), and any monomial $m \in \langle m_1, m_2, \ldots, m_s\rangle$ should be a multiple of some m_i.

Definition 3 (Hilbert's Function)**.** For a monomial ideal $I \subseteq \mathbb{K}[\boldsymbol{x}]$, *Hilbert's function* $H_I : \mathbb{N} \to \mathbb{N}$ is defined as

$$H_I(n) \triangleq \dim_{\mathbb{K}} \mathbb{K}_n[\boldsymbol{x}]/I_n = \dim_{\mathbb{K}} \mathbb{K}_n[\boldsymbol{x}] - \dim_{\mathbb{K}} I_n,$$

where $\mathbb{K}_n[\boldsymbol{x}]$ is the set of homogeneous polynomials of degree n (i.e., polynomials in which all monomials have the same degree n) and $I_n = I \cap \mathbb{K}_n[\boldsymbol{x}]$, and both are linear spaces over $\mathbb{K}$; $\dim_{\mathbb{K}}$ denotes the dimension of the linear spaces over $\mathbb{K}$.

Note that $\dim_{\mathbb{K}} \mathbb{K}_n[\boldsymbol{x}] = (n)_{d-1}$ is the number of monomials of degree n, where

$$(n)_{d-1} \triangleq \binom{n+d-1}{d-1} = \frac{(n+d-1)!}{n!\,(d-1)!}.$$

$H_I(n) = 0$ means that I contains all monomials of degree at least n.

We use Macaulay's theorem [32] to estimate the value of Hilbert's function H_I. To this end, we define a function $\mathrm{Inc}_k : \mathbb{N} \to \mathbb{N}$ for natural number $k \geq 1$ below. When k is given, any number $n \in \mathbb{N}$ can be uniquely decomposed as

$$n = (n_1)_k + (n_2)_{k-1} + \cdots + (n_r)_{k-r+1} \triangleq (n_1, n_2, \ldots, n_r)_k,$$

where $0 \leq r \leq k$ and $n_1 \geq n_2 \geq \cdots \geq n_r \geq 0$. In fact, $0 = ()_k$ with $r = 0$; and for $n > 0$, $n_1, n_2, \ldots, n_r$ are successively determined by

$$\begin{aligned} &(n_1)_k \leq n < (n_1+1)_k \\ &(n_2)_{k-1} \leq n - (n_1)_k < (n_2+1)_{k-1} \\ &\qquad \cdots \end{aligned}$$

until $n = (n_1, n_2, \ldots, n_r)_k$ for some $r \geq 1$. Now we define

$$\mathrm{Inc}_k((n_1, \ldots, n_r)_k) = (n_1, \ldots, n_r)_{k+1}.$$

For instance, $\mathrm{Inc}_k(0) = 0$, and $\mathrm{Inc}_3(11) = \mathrm{Inc}_3((2,0)_3) = (2,0)_4 = 16$ (note that $11 = (2,0)_3$).

Theorem 3 (Macaulay's Theorem [32])**.** *For any monomial ideal $I \subseteq \mathbb{K}[\boldsymbol{x}]$,*

$$H_I(n+1) \leq \mathrm{Inc}_n(H_I(n)) \text{ for all } n \geq 1.$$

Moreover, $H_I(n+1) = \mathrm{Inc}_n(H_I(n))$ if $\mathrm{gdeg}(I) \leq n$ and a monomial basis of I is n-compressed.

4. Upper bound on the length of polynomial ascending chains with a control function

In this section, we investigate the maximal length of ascending chains of polynomial ideals with a control function. To this end, we first define the concepts of *f-bounded chains* and *f-generating sequences*.

Consider a general field $\mathbb{K}$. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ be a vector of d variables and $\mathbb{K}[\boldsymbol{x}]$ the polynomial ring of $\boldsymbol{x}$ over $\mathbb{K}$.

Definition 4 (f-Bounded Chain of Polynomial Ideals)**.** For any monotonically increasing function $f : \mathbb{N} \to \mathbb{N}$ (namely, $f(x) \leq f(y)$ for all $x \leq y$), an ascending chain $I_1 \subseteq I_2 \subseteq \cdots \subseteq I_n$ of polynomial ideals of $\mathbb{K}[\boldsymbol{x}]$ is called *f-bounded*, if $\mathrm{gdeg}(I_i) \leq f(i)$ for all $i \geq 1$. We denote by $L(d, f)$ the maximal length of all strictly ascending chains of $\mathbb{K}[\boldsymbol{x}]$ that are f-bounded.

Remark 1.

1. The condition of f-boundedness is necessary to define the maximal length, as the length of a chain with unbounded degrees could be arbitrarily large. For instance, the length of $\langle x^n\rangle \subset \langle x^{n-1}\rangle \subset \cdots \subset \langle 1\rangle$ may be arbitrarily large if n is unbounded.
2. For ease of discussion, the function f is assumed to be increasing without loss of generality. In fact, for a general f, consider the increasing function $F : \mathbb{N} \to \mathbb{N}$ with $F(n) \triangleq \max\{f(1), f(2), \ldots, f(n)\}$. Then an f-bounded chain is always an F-bounded chain since $f(n) \leq F(n)$ for all n. Hence, $L(d, f) \leq L(d, F)$ and $L(d, F)$ is an upper bound on the lengths of the chains.

We aim to compute $L(d, f)$ in terms of the number d of variables and function f. To this end, we consider ascending chains of a special form.

Definition 5 (f-Generating Sequence of Monomials)**.** For a function $f : \mathbb{N} \to \mathbb{N}$, a finite sequence of monomials $m_1, m_2, \ldots, m_n \in \mathcal{M}[\boldsymbol{x}]$ is *f-generating*, if $\deg(m_i) \leq f(i)$ and $m_{i+1} \notin \langle m_1, \ldots, m_i\rangle$ for all $i \geq 1$.

Then an f-generating monomial sequence $m_1, \ldots, m_n$ generates a strictly ascending chain $I_1 \subset \cdots \subset I_n$ of monomial ideals satisfying $\mathrm{gdeg}(I_i) \leq f(i)$, where $I_i \triangleq \langle m_1, \ldots, m_i\rangle$. By considering the sequence of leading monomials of polynomials in an ascending chain, Moreno-Socías proved in [33] that in order to compute $L(d, f)$, it suffices to consider the ascending chains that are generated by f-generating monomial sequences. Namely,[2]

[2] Proposition 1 is valid, as it only states that $L(d, f)$ is the largest number of monomials, yet without addressing how to construct $L(d, f)$, which is essentially the flawed part in [33].

Proposition 1 ([33, Lemma 3.1]). *$L(d, f)$ is exactly the largest number of monomials of an f-generating sequence in $\mathcal{M}[\boldsymbol{x}]$.*

Proposition 1 suggests that the length $L(d, f)$ is *independent* of the choice of the field $\mathbb{K}$, as it suffices to consider monomials instead of polynomials. In the rest of this section, we construct the longest chain in the form of an f-generating sequence. We start with a special case where the degrees of polynomial ideals are not just bounded but completely determined by a function f. Then we reduce the general case to this special one.

Our approach to investigating the maximal length of polynomial ascending chains with a control function consists of three steps:

(i) Reduce computing the bound on f-bounded polynomial ideal chains to computing the bound on f-generating sequences of monomials (cf. Proposition 1), which has been obtained by the results in [33].
(ii) Construct the longest f-generating sequence of monomials whose degrees are precisely determined (not just upper-bounded) by f, leveraging Hilbert's function and Macaulay's theorem (cf. Lemma 1).
(iii) Prove that the bound on f-generating sequences of monomials is identical to the length of the sequence of f-bounded polynomial ideals over an extended space (cf. Theorem 4).

Remark 2. The use of Hilbert's function and Macaulay's theorem is inspired by Moreno-Socías's proof tactics in [33]. There are, however, crucial differences in the underlying techniques for constructing the longest f-generating sequence of monomials: we perform the construction in a necessarily extended space of monomials $\mathcal{M}[\boldsymbol{x}, x_{d+1}]$ (Step (iii), with an auxiliary variable x_{d+1}); in contrast, the construction in [33] works directly on the original monomial space $\mathcal{M}[\boldsymbol{x}]$. We will show in Example 4 that the constructed f-generating sequence of monomials in [33] is in fact not the longest (revealing that Proposition 4.3 in [33] is wrong). It will also be shown that such a spuriously "maximal" length may lead to unsoundness when being applied to termination analysis of MPPs.

4.1. The longest chain of specified degrees

Consider a special type of f-generating sequences $m_1, \dots, m_n$ such that

$$\forall i \in \{1, \dots, n\}: \ \deg(m_i) = f(i). \tag{1}$$

And in particular, we inductively construct a sequence $seq[\boldsymbol{x}; f] \triangleq (\hat{m}_1, \dots, \hat{m}_N)$ of monomials in $\mathcal{M}[\boldsymbol{x}]$ for a control function f as follows: Initially define $\hat{m}_1 = x_1^{f(1)}$; suppose $\hat{m}_1, \dots, \hat{m}_n$ are defined for some $n \geq 1$, then let

$$\hat{m}_{n+1} \triangleq \max\{m \mid \deg(m) = f(n+1), m \notin \langle \hat{m}_1, \dots, \hat{m}_n \rangle\} \tag{2}$$

until $\{m \mid \deg(m) = f(N+1)\} \subseteq \langle \hat{m}_1, \dots, \hat{m}_N \rangle$ holds for some N. Here max is taken with respect to the lexicographical ordering over monomials.

Obviously, $seq[\boldsymbol{x}; f]$ is an f-generating sequence that satisfies (1). It follows from (2) that the corresponding ideal $\hat{I}_n \triangleq \langle \hat{m}_1, \hat{m}_2, \dots, \hat{m}_n \rangle$ is $f(n)$-compressed for every $n = 1, 2, \dots, N$, and $H_{\hat{I}_N}(f(N+1)) = 0$ holds by the definition of Hilbert's function.

Proposition 2. $x_1^{f(1)} = \hat{m}_1 \succ \hat{m}_2 \succ \cdots \succ \hat{m}_N = x_d^{f(N)}$.

Proof. We first prove that $\hat{m}_n \succ \hat{m}_{n+1}$ for any $n \in \{1, 2, \dots, N-1\}$. Let

$$m = \max\{m' \mid \deg(m') = f(n) \text{ and } m' \text{ is a monomial factor of } \hat{m}_{n+1}\}.$$

Such m exists since $\deg(\hat{m}_{n+1}) = f(n+1) \geq f(n)$. $\hat{m}_{n+1} \notin \hat{I}_n \triangleq \langle \hat{m}_1, \hat{m}_2, \dots, \hat{m}_n \rangle$ implies that $m \notin \hat{I}_n$, which further implies that $\hat{m}_n \succ m$ since $\hat{I}_n$ is $f(n)$-compressed and $\deg(m) = \deg(\hat{m}_n) = f(n)$. $\hat{m}_n \succ \hat{m}_{n+1}$ is immediately obtained from the maximality of m.

To prove $\hat{m}_N = x_d^{f(N)}$, we note that

$$x_d^{f(N+1)} \in \{m \mid \deg(m) = f(N+1)\} \subseteq \langle \hat{m}_1, \dots, \hat{m}_N \rangle,$$

namely, $x_d^{f(N+1)}$ is a monomial multiple of some $\hat{m}_n$, $n \in \{1, 2, \dots, N\}$. Thus, $\hat{m}_n = x_d^{f(n)} \preceq \hat{m}_N$. But for $n < N$, we just proved that $\hat{m}_n \succ \hat{m}_N$. The only possibility is that $n = N$ and $\hat{m}_N = x_d^{f(N)}$. □

Example 2. For $d = 3$ and $f(n) = 2 \times 3^{n-1}$, we have

$$seq[x, y, z; f] = \{\hat{m}_1, \hat{m}_2, \dots, \hat{m}_{4382}\} = \{x^2, xy^5, xy^4z^{13}, xy^3z^{50}, xy^2z^{159}, xyz^{484}, xz^{1457}, y^{4374}, y^{4373}z^{8749}, \dots, y^0z^{2\cdot 3^{4381}}\}.$$

Then, $N = 4382$ in this case.

Now we show that the sequence $seq[\boldsymbol{x}; f]$ has the largest number of monomials among all f-generating sequences satisfying (1).

Lemma 1. *If $m_1, m_2, \dots, m_n$ is an f-generating sequence that satisfies (1), then $n \leq N$.*

Proof. We prove by contradiction. Suppose $n \geq N+1$. Let $I_i = \langle m_1, \dots, m_i \rangle$ for all $i \in \{1, 2, \dots, n\}$.

For simplicity, we define $H_i(j) \triangleq H_{I_i}(j)$ and $\hat{H}_i(j) \triangleq H_{\hat{I}_i}(j)$, for all $i, j \geq 1$. Observe that

$$H_i(j) \begin{cases} = H_{i-1}(j), & j < f(i); \\ = H_{i-1}(j) - 1, & j = f(i); \\ \leq \mathrm{Inc}_{j-1} H_i(j-1), & j > f(i). \end{cases} \tag{3}$$

The first two equalities follow from the definition of I_i, and the third inequality from Theorem 3. Similarly, we have

$$\hat{H}_i(j)\begin{cases} = \hat{H}_{i-1}(j), & j < f(i); \\ = \hat{H}_{i-1}(j) - 1, & j = f(i); \\ = \mathrm{Inc}_{j-1}\, \hat{H}_i(j-1), & j > f(i). \end{cases} \tag{4}$$

Here, the third one becomes an equality since $\hat{I}_i$ is $f(i)$-compressed and $\mathrm{gdeg}(\hat{I}_i) = f(i) \le j-1$, and thus the conditions for the equality in Theorem 3 are satisfied. On the other hand, we observe that $H_1(f(1)) = \hat{H}_1(f(1)) = (f(1))_{d-1} - 1$. It then can be proved inductively from (3) and (4) that

$$\begin{aligned} &H_i(j) = H_{i-1}(j) \le \hat{H}_{i-1}(j) = \hat{H}_i(j), && j < f(i); \\ &H_i(j) = H_{i-1}(j) - 1 \le \hat{H}_{i-1}(j) - 1 = \hat{H}_i(j), && j = f(i); \\ &H_i(j) \le \mathrm{Inc}_{j-1}\, H_i(j-1) \le \mathrm{Inc}_{j-1}\, \hat{H}_i(j-1) = \hat{H}_i(j), && j > f(i). \end{aligned}$$

Here, the fact that $r \le t \implies \mathrm{Inc}_j(r) \le \mathrm{Inc}_j(t)$ is applied. Hence we have proved that $H_i(j) \le \hat{H}_i(j)$ for all $i \ge 1$ and $j \ge 1$. Then $H_N(f(N+1)) \le \hat{H}_N(f(N+1)) = 0$. We have $m_{N+1} \in \{m \mid \deg(m) = f(N+1)\} \subseteq \langle \hat{m}_1, \dots, \hat{m}_n \rangle$, which contradicts (2). □

We consider next how to compute the maximal length N from the sequence $seq[\boldsymbol{x}; f]$. To this end, we define a function $\Omega(d, f, t)$ to calculate the locations k of monomials of the form $\hat{m}_k = x_1^a x_d^b$, i.e., the monomials consisting of only x_1 and x_d.

Definition 6.

Given $d, t \in \mathbb{N}$ and an increasing function $f : \mathbb{N} \to \mathbb{N}$, a function $\Omega(d, f, t)$ is recursively defined for $d \ge 1$ and $t \le f(1)$ as:

1. $\Omega(1, f, t) = 1$ and $\Omega(d, f, 0) = 1$, for all $d \ge 1$, f and $t \in \{0, 1, \dots, f(1)\}$.
2. For $d > 1$ and $1 \le t \le f(1)$,

$$\Omega(d, f, t) = \Omega(d, f, t-1) + \Omega(d-1, f_{\Omega(d,f,t-1),t-f(1)}, f_{\Omega(d,f,t-1),t-f(1)}(1)).$$

Here $f_{m,r}$ is a function $f_{m,r}(n) \triangleq f(m+n) + r$.

Proposition 3.

Let $k = \Omega(d, f, t)$ with $d \ge 2$ and $t \le f(1)$, then the kth monomial $\hat{m}_k$ of the chain $seq[\boldsymbol{x}; f]$ constructed in (2) is exactly

$$\hat{m}_k = x_1^{f(1)-t} x_d^{f(k)-f(1)+t}.$$

In particular, for $t = f(1)$, the monomial $\hat{m}_N = x_d^{f(N)}$ is at the end of the chain and thus the length of the chain is $N = \Omega(d, f, f(1))$.

Proof. We prove by induction on d. For $d = 2$, the kth monomial is $\hat{m}_k = x_1^{f(1)-k+1} x_2^{f(k)-f(1)+k-1}$ according to the construction of (2). On the other hand, $\Omega(d, f, t) = t + 1$ from Definition 6. Let $k = t + 1$, the result holds.

Suppose the result is true for $d-1$, then we prove it for $d \ge 3$. From Proposition 2, the exponents of variable x_1 are (not strictly) decreasing in the sequence $seq[\boldsymbol{x}; f]$. Hence, this sequence can be divided into many segments such that for monomials in each segment, the exponents of x_1 are the same. Consider an arbitraty one of such segments:

$$\hat{m}_{i+1} \triangleq x_1^{\alpha} m_{i+1}, \hat{m}_{i+2} \triangleq x_1^{\alpha} m_{i+2}, \dots, \hat{m}_j \triangleq x_1^{\alpha} m_j,$$

where α denotes the exponent of x_1 and $m_{i+1}, m_{i+2}, \dots, m_j$ are monomials in $\mathcal{M}[x_2, \dots, x_d]$. We claim that the sequence $m_{i+1}, m_{i+2}, \dots, m_j$ is exactly $seq[x_2, \dots, x_d; f_{i,-\alpha}]$ (and for ease of reading, we postpone its argument to the end of this proof). Then, from Proposition 2, the first monomial is of the form $m_{i+1} = x_2^{f(i+1)-\alpha}$ and the last monomial is $m_j = x_d^{f(j)-\alpha}$; by induction hypothesis, the length of this sequence is

$$j - i = \Omega(d-1, f_{i,-\alpha}, f_{i,-\alpha}(1)).$$

Now we apply this result to all segments of the sequence $seq[\boldsymbol{x}; f]$: The first segment is a single monomial $\hat{m}_1 = x_1^{f(1)}$ with the exponent $\alpha = f(1)$ of x_1; the second segment is $\hat{m}_2, \hat{m}_3, \dots, \hat{m}_{i_1}$ with exponent $\alpha = f(1) - 1$ of x_1; and generally, the segment with exponent $\alpha = f(1) - t$ is represented as

$$\hat{m}_{i_{t-1}+1}, \hat{m}_{i_{t-1}+2}, \dots, \hat{m}_{i_t},$$

for $t = 1, 2, \dots, f(1)$. Here i_0 is defined as 1 and $i_{f(1)} = N$. Then, we have $\hat{m}_{i_t} = x_1^{f(1)-t} x_d^{f(i_t)-f(1)+t}$ and

$$i_t - i_{t-1} = \Omega(d-1, f_{i_{t-1},t-f(1)}, f_{i_{t-1},t-f(1)}(1)).$$

It further follows that $i_t = \Omega(d, f, t)$ by induction on t: For $t = 0$, $i_0 = 1 = \Omega(d, f, 0)$; if $i_{t-1} = \Omega(d, f, t-1)$, then, from Clause 2 of Definition 6,

$$i_t = i_{t-1} + \Omega(d-1, f_{i_{t-1},t-f(1)}, f_{i_{t-1},t-f(1)}(1)) = \Omega(d, f, t).$$

The result of Proposition 3 is achieved by letting $k = i_t$.

At last, we prove the claim of $seq[x_2, \dots, x_d; f_{i,-\alpha}] = (m_{i+1}, m_{i+2}, \dots, m_j)$. From the definition, we actually prove that

1. $m_{i+1} = x_2^{f(i+1)-\alpha}$;

2. $m_{k+1} = \max\{m \mid \deg(m) = f(k+1) - \alpha, m \notin \langle m_{i+1}, \dots, m_k\rangle\}$, for $k = i+1, i+2, \dots, j-1$; and
3. $\{m \mid \deg(m) = f(j+1) - \alpha\} \subseteq \langle m_{i+1}, \dots, m_j\rangle$.

For $i = 0$, we have $\alpha = f(1)$ from $\hat{m}_1 = x_1^{f(1)}$, and thus $m_{i+1} = 1$. The first result holds. For $i > 0$, the first result follows from the construction (2) with $n = i+1$. Similarly, the second result follows from the construction (2) with $n = k$, noting that the exponent of x_1 is greater than α in $\hat{m}_1, \hat{m}_2, \dots, \hat{m}_i$ and thus $\hat{m}_{n+1} \notin \langle \hat{m}_1, \dots, \hat{m}_n\rangle$ is equivalent to $m_{n+1} \notin \langle m_{i+1}, \dots, m_n\rangle$. For $j = N$, the third result follows immediately from the condition

$$\{x_1^\alpha \cdot m \mid \deg(m) = f(N+1) - \alpha\} \subseteq \{m' \in \mathcal{M}[\boldsymbol{x}] \mid \deg(m') = f(N+1)\} \subseteq \langle \hat{m}_1, \dots, \hat{m}_N\rangle.$$

To prove the third result for $j < N$, consider a monomial $m \in \mathcal{M}[x_2, \dots, x_d]$ with $\deg(m) = f(j+1) - \alpha$, then $x_1^\alpha m \succ \hat{m}_{j+1}$ as the exponent of x_1 is smaller than α in $\hat{m}_{j+1}$. Since $\deg(x_1^\alpha m) = f(j+1)$ and $\langle \hat{m}_1, \hat{m}_2, \dots, \hat{m}_{j+1}\rangle$ is $f(j+1)$-compressed, we have $x_1^\alpha m \in \langle \hat{m}_1, \hat{m}_2, \dots, \hat{m}_{j+1}\rangle$ and consequently $m \in \langle m_{i+1}, \dots, m_j\rangle$. □

Example 3. Let $f(n) = 2 \times 3^{n-1}$, then $\Omega(3, f, 2) = 4382$ is exactly the number of monomials in Example 2.

4.2. Reduction of the general case

Now we drop the restriction in (1) and consider the length of a general f-generating sequence of monomials $m_1, \dots, m_n$ in $\mathcal{M}[\boldsymbol{x}]$. The idea is to reduce this general case to the special case. To this end, we introduce a new variable x_{d+1} (for which the lexicographic order adheres to $x_1 \succ \cdots \succ x_d \succ x_{d+1}$), and construct for each m_i $(i = 1, \dots, n)$ a monomial $\tilde{m}_i \in \mathcal{M}[\boldsymbol{x}, x_{d+1}]$ such that $\tilde{m}_i = m_i x_{d+1}^{c_i}$, where $c_i = f(i) - \deg(m_i) \in \mathbb{N}$. Hence $\deg(\tilde{m}_i) = f(i)$, and thus (1) is satisfied by $\tilde{m}_1, \dots, \tilde{m}_n$. Furthermore,

Proposition 4. *$\tilde{m}_1, \dots, \tilde{m}_n$ is an f-generating sequence of $\mathbb{K}[\boldsymbol{x}, x_{d+1}]$.*

Proof. It suffices to prove that for every $i = 1, 2, \dots, n-1$, $\tilde{m}_{i+1} \notin \langle \tilde{m}_1, \dots, \tilde{m}_i\rangle$. In fact, if this is not the case then $\tilde{m}_{i+1}$ should be a multiple of some $\tilde{m}_j$, with $j \le i$. It implies that m_{i+1} is a multiple of m_j, which contradicts $m_{i+1} \notin \{m_1, \dots, m_i\}$. □

It then follows immediately from Lemma 1 and Proposition 3 that $n \le \Omega(d+1, f, f(1))$. Since $m_1, \dots, m_n$ is arbitrarily chosen, we obtain $L(d, f) \le \Omega(d+1, f, f(1))$ by Proposition 1. Conversely, we also show that $\Omega(d+1, f, f(1)) \le L(d, f)$. Consider the sequence $seq[\boldsymbol{x}, x_d; f] = (\hat{m}_1, \dots, \hat{m}_N)$, where $N = \Omega(d+1, f, f(1))$. Replacing x_{d+1} by 1 in each $\hat{m}_i$ yields a sequence $m'_1, \dots, m'_N$ of monomials in $\mathbb{K}[\boldsymbol{x}]$, which is still an f-generating sequence:

Proposition 5. *$m'_1, \dots, m'_N$ is an f-generating sequence.*

Proof. Observe that $\deg(m'_i) \le \deg(\hat{m}_i) = f(i)$ for all i. It remains to prove that for all i, $m'_i \notin \langle m'_1, \dots, m'_{i-1}\rangle$.

From Proposition 2, $\hat{m}_i \succ \hat{m}_{i+1}$ for all i, which implies that $m'_i \succeq m'_{i+1}$. Note that $m'_i \neq m'_{i+1}$; otherwise, $\hat{m}_i \triangleq m'_i x_{d+1}^\alpha \succ m'_{i+1} x_{d+1}^\beta \triangleq \hat{m}_{i+1}$ implies that $\alpha > \beta$, and thus $f(i) = \deg(m'_i) + \alpha > \deg(m'_{i+1}) + \beta = f(i+1)$, which contradicts the monotonicity of f. Therefore, $m'_i \succ m'_{i+1}$. The result follows immediately. □

$\Omega(d+1, f, f(1)) = N \le L(d, f)$ follows immediately from this result and Proposition 1. Therefore, we obtain the following fact.

Theorem 4. $L(d, f) = \Omega(d+1, f, f(1))$.

4.3. Mistake in Moreno-Socías's approach

The following example shows that the monomial ascending chain constructed by Moreno-Socías in [33] is *not* the longest:

Example 4. For $d = 2, f(i) = 1 + i$, the monomial set of the longest monomial ascending chain is

$$\{m_1, m_2, \dots, m_{11}\} = \{x^2, xy^2, xy, x, y^6, y^5, y^4, y^3, y^2, y, 1\},$$

and $L(2, f) = 11$.

However, according to Moreno-Socías's approach [33], the monomial set in Example 4 is

$$\{x^2, xy^2, y^4, y^3, xy, y^2, x, y, 1\}, \tag{5}$$

and thus $L(2, f) = 9$, which is evidently smaller (than 11) and can be proven not to be the maximal length of all the ascending chains (by applying Definition 5). We remark that such a spurious bound on the length may lead to crucial consequences when being applied in termination analysis of MPPs: a non-terminating MPP can be erroneously concluded as being terminating because of not reaching a sufficient depth in its execution tree (which is captured by the maximal length of the ascending chains). Thus termination analysis based on this bound may give unsound results.

Next, we pinpoint the specific technical flaw in [33] that leads to the erroneous result as demonstrated by Example 4. À la Proposition 4.3 in [33], the strategy to construct the longest monomial ascending chain is as follows: Every new monomial to be added is chosen among those of the maximum possible degree, by taking the least available one per the lexicographic order. This strategy, however, relies on a condition that is not always valid: Given *any* simple monomial chain $(m_1, \dots, m_i)$, suppose we employ the above strategy to generate monomials μ_i with length L. We define, as per Proposition 4.3 in [33], $i_0 = 0$, $i_r = L$, and

$$\{i_1, \dots, i_{r-1}\} = \{i \mid 0 < i < L, \deg(\mu_i) \neq \deg(\mu_{i+1})\}$$

with $i_1 < \cdots < i_{r-1}$. Moreover, for any $0 < j < r$, let $\alpha = i_{j-1}$, $\beta = i_j$, and

$$\bar{\delta} = \deg(\mu_{\alpha+1}) = \cdots = \deg(\mu_\beta), \quad \underline{\delta} = \min\{\deg(m_{\alpha+1}), \ldots, \deg(m_\beta)\}.$$

Then the condition erroneously asserts that

$$\underline{\delta} \le \deg(m_k) \le \bar{\delta} \quad \text{for} \quad \alpha < k \le \beta.$$

This condition holds for the so-constructed chain (5) where $\{i_0, i_1, \ldots, i_7\} = \{0, 1, 2, 3, 4, 6, 8, 9\}$. Consider, for instance, $\alpha = i_4 = 4$ and $\beta = i_5 = 6$, then $\bar{\delta} = \deg(xy) = \deg(y^2) = 2$, and indeed we have $\deg(m_5) = 2 \le \bar{\delta}$ and $\deg(m_6) = 2 \le \bar{\delta}$. Nonetheless, for the genuinely longest chain $\{m_1, m_2, \ldots, m_{11}\}$ as given in Example 4, we have $\deg(m_5) = 6 \not\le \bar{\delta}$ and $\deg(m_6) = 5 \not\le \bar{\delta}$. In fact, one could verify that Proposition 4.3 in [33] yields the correct result in the special case of $\forall i \in \{1, \ldots, n\}: \ \deg(m_i) = f(i)$, as identified in Section 4.1.

4.4. Complexity of $L(d, f)$

To show that $L(d, f)$ is not primitive recursive, we introduce the Ackermann function:

Definition 7 (Ackermann Operator). The *Ackermann operator* $\Uparrow^d$ for natural numbers d, x, y is defined by

$$x \Uparrow^0 y = 1 + xy$$
$$x \Uparrow^d 0 = 1 \quad d > 0$$
$$x \Uparrow^d y = x \Uparrow^{d-1} (x \Uparrow^d (y-1)) \quad d > 0, y > 0$$

For instance, one can easily compute that $x \Uparrow^d 1 = 1 + x$ and $x \Uparrow^1 y = 1 + x + x^2 + \cdots + x^y$. A basic property of Ackermann function is that it is not primitive recursive. Now we use it to discuss the complexity of $L(d, f)$.

Proposition 6. *For two positive integers a, b, and $f(n) = a(n-1) + b$ with $n \ge 1$,*

$$L(d, f) = \frac{1}{a}\big((a+1) \Uparrow^{d-1} (b+1)\big) - \frac{b+2}{a}.$$

Proof. We prove that for $d \ge 2$ and $0 \le t \le b$,

$$\Omega(d, f, t) = \frac{1}{a}\big((a+1) \Uparrow^{d-2} (t+1)\big) - \frac{t+2}{a}, \tag{6}$$

then Proposition 6 follows immediately from Theorem 4 by setting $t = b = f(1)$. The proof of (6) is achieved by induction on d and t as follows.

For $t = 0$ and $d \ge 2$, $\Omega(d, f, 0) = 1$, and $(a+1) \Uparrow^{d-2} (t+1) = (a+1) \Uparrow^{d-2} 1 = a + 2$; (6) holds. For $d = 2$ and $0 \le t \le b$, we have $\Omega(2, f, t) = 1 + t$ since $\Omega(1, f, t) = \Omega(2, f, 0) = 1$ and $\Omega(2, f, t) = \Omega(2, f, t-1) + 1$ from Definition 6. Noting that $(a+1) \Uparrow^0 (t+1) = 1 + (a+1)(t+1)$, (6) is obtained.

Suppose (6) holds for $d - 1$ and for $(d, t-1)$, we now prove it for (d, t) from the recursive relation in Definition 6:

$$\Omega(d, f, t) = \Omega(d, f, t-1) + \Omega(d-1, f_{\Omega(d,f,t-1),t-f(1)}, f_{\Omega(d,f,t-1),t-f(1)}(1)).$$

Specifically, by induction hypothesis on $(d, t-1)$,

$$\Omega(d, f, t-1) = \frac{1}{a}\big((a+1) \Uparrow^{d-2} t\big) - \frac{t+1}{a}.$$

We write $m = \Omega(d, f, t-1)$, then $am + t + 1 = (a+1) \Uparrow^{d-2} t$. We further write $r = t - f(1) = t - b$, then, by induction hypothesis on $d - 1$, we have

$$\Omega(d-1, f_{m,r}, f_{m,r}(1)) = \frac{1}{a}\big((a+1) \Uparrow^{d-3} (am + b + r + 1)\big) - m - \frac{b + r + 2}{a},$$

noting that $f_{m,r}(n) = a(n + m - 1) + b + r = a(n-1) + am + b + r$ and thus $f_{m,r}(1) = am + b + r$. Therefore,

$$\begin{aligned}\Omega(d, f, t) = m + \Omega(d-1, f_{m,r}, f_{m,r}(1)) &= \frac{1}{a}\big((a+1) \Uparrow^{d-3} (am + t + 1)\big) - \frac{t+2}{a} \\ &= \frac{1}{a}\big((a+1) \Uparrow^{d-3} ((a+1) \Uparrow^{d-2} t)\big) - \frac{t+2}{a} \\ &= \frac{1}{a}\big((a+1) \Uparrow^{d-2} (t+1)\big) - \frac{t+2}{a},\end{aligned}$$

which is exactly (6) for d and t. □

In Proposition 6, $L(d, f)$ is a primitive recursive function on a and b for a fixed d, yet becomes Ackermannian when d is not fixed (for general f). Indeed, using Proposition 6, the Ackermann operator can directly be defined in terms of $L(d, f)$, which explains why $L(d, f)$ cannot be primitive recursive in general. This observation coincides with Figueira et al.'s conclusion in [34] that $L(d, f_{t,0})$, a function on t, is at level $\Gamma_{\lambda+k-1}$ of the fast-growing hierarchy for f at level λ. Their results answered Seidenberg's second question on non-primitive recursiveness by giving a lower bound on $L(d, f)$. In contrast, our approach gives an explicit recursive function (which is Ackermannian) as the maximal length $L(d, f)$, thereby an answer also to Seidenberg's request for an explicit expression of the bound. We will show in Section 6 that an explicit expression of the bound suffices to establish decidability of the termination problem of MPPs (see Remark 5).

5. Termination of MPPs with equality guards

5.1. Multi-path polynomial programs (MPPs)

The family of polynomial programs considered in this section is formally defined as

Definition 8 (MPPs with Equality Guards [24]). A *multi-path polynomial program (MPP) with equality guard* is of the form

$$\text{WHILE}\quad (G(\boldsymbol{x})=0)\quad \left\{\begin{array}{ll} & \boldsymbol{x}:=\boldsymbol{A}_1(\boldsymbol{x});\\ \|\ & \boldsymbol{x}:=\boldsymbol{A}_2(\boldsymbol{x});\\ & \vdots\\ \|\ & \boldsymbol{x}:=\boldsymbol{A}_{l-1}(\boldsymbol{x});\\ \|\ & \boldsymbol{x}:=\boldsymbol{A}_l(\boldsymbol{x});\end{array}\right\},\ \text{where} \tag{7}$$

1. $\boldsymbol{x}\in\mathbb{K}^d$ denotes the vector of program variables, where $\mathbb{K}$ is assumed to be an *effective* field in this and the next sections (see Remark 3-*1* for a detailed explanation);
2. $G\in\mathbb{K}[\boldsymbol{x}]$ in the loop condition (guard) is a polynomial;
3. $\boldsymbol{A}_i\in\mathbb{K}^d[\boldsymbol{x}]$ $(1\le i\le l)$ are vectors of polynomials describing assignments to variables in the loop body;
4. $\|$ is interpreted as a nondeterministic choice between the l assignments.

Remark 3.

1. $\mathbb{K}$ is required to be *effective* (e.g., the field $\mathbb{Q}$, $\overline{\mathbb{Q}}$, the simple extension field $\mathbb{Q}(\pi)$, or the field of rational functions over $\mathbb{Q}$), since the program (7) should be *executable* on a computer. Specifically, for an input $\boldsymbol{x}$, the symbolic computation of $G(\boldsymbol{x})$ outputs an expression e consisting of components of $\boldsymbol{x}$, coefficients of G (denoted by Coeff_0 the set of them), and operations of addition, subtraction and multiplication in $\mathbb{K}$. Then, to run the loop, it should be decidable whether $e=0$. Furthermore, after an iteration of the loop for some $\boldsymbol{A}_i$, the coefficients of polynomials in $\boldsymbol{A}_i$ (denoted by Coeff_i the set of them) are involved in the expression. The effectiveness of $\mathbb{K}$ guarantees that checking whether an expression of this form is zero or not is decidable. In our discussion of an MPP, if the field is not effective (e.g., $\mathbb{K}=\mathbb{R}$), we always assume that $e=0$ is decidable for every $e\in\mathrm{Expr}_{\mathbb{K}}(\mathrm{Coeff}\cup\{\boldsymbol{x}\})$, where $\mathrm{Coeff}=\mathrm{Coeff}_0\cup\mathrm{Coeff}_1\cup\cdots\cup\mathrm{Coeff}_l$ and $\mathrm{Expr}_{\mathbb{K}}(S)$ denotes the set of all expressions from a finite set $S\subseteq\mathbb{K}$ and operators in $\mathbb{K}$. This condition is sufficient to establish the decidability of termination of MPP (7) (cf. Theorem 6).
2. The loop guard of MPP (7) can be extended to a more general form $\bigwedge_{i=1}^{M}\bigvee_{j=1}^{N_i}G_{ij}(\boldsymbol{x})=0$. Here, each disjunctive clause $\bigvee_{j=1}^{N_i}G_{ij}(\boldsymbol{x})=0$ is equivalent to $G_i(\boldsymbol{x})=0$ with $G_i\triangleq\prod_{j=1}^{N_i}G_{ij}$, then the CNF form $\bigwedge_{i=1}^{M}G_i(\boldsymbol{x})=0$ can be handled by applying the same reduction as in [24]. It is nevertheless essential to assume that inequalities will never occur in guards, otherwise the termination problem will become undecidable, even not semi-decidable [7].
3. We are interested in the problem of deciding termination of the loop for a given initial value of $\boldsymbol{x}$. If the input $\boldsymbol{x}$ is subject to semi-algebraic constraints and all coefficients of polynomials are rational numbers, our decidability result still holds for $\mathbb{K}=\mathbb{R}$ according to [31]. This further suggests the decidability of *universal* termination for $\mathbb{R}$ (and for $\mathbb{C}$, which reduces to $\mathbb{R}$ in the case of universal termination).

Example 5 (`liu1` [24]). Consider the MPP

$$\text{WHILE}\quad (x+y=0)\quad \left\{\begin{array}{ll} & (x,y):=(x+1,y-1);\\ \|\ & (x,y):=(x^2,y^2);\end{array}\right\}.$$

We have $d=2$, $G(x,y)=x+y$, $\boldsymbol{A}_1(x,y)=(x+1,y-1)$, and $\boldsymbol{A}_2(x,y)=(x^2,y^2)$. Note that this example, amongst others, cannot be handled by Liu et al.'s technique based on under/over-approximating the NTI [24]. See the experiments in Section 8.

Example 6. Consider a nondeterministic quantum program [35] of the form

$$\text{WHILE}\quad (\mathit{Mea}[\rho]=0)\quad \left\{\begin{array}{ll} & \rho:=\mathcal{E}_1(\rho);\\ \|\ & \rho:=\mathcal{E}_2(\rho);\\ & \vdots\\ \|\ & \rho:=\mathcal{E}_{l-1}(\rho);\\ \|\ & \rho:=\mathcal{E}_l(\rho);\end{array}\right\},\quad \text{where}$$

1. $\rho\in\mathbb{C}^{d^2}$ is a $d\times d$ density matrix.
2. $\mathit{Mea}=\{M_0,M_1\}$ is a two-outcome quantum measurement, where M_0 and M_1 are $d\times d$ complex matrices.
3. $\mathcal{E}_i$ are quantum super-operators that perform linear transformations over $\mathbb{C}^{d^2}$.

In this example, $\boldsymbol{x}=\rho$, $G(\boldsymbol{x})=\mathrm{tr}(M_0\rho M_0^\dagger)$ and $\boldsymbol{A}_i=\mathcal{E}_i$, where $\mathrm{tr}(M)$ is the trace of a matrix M, and $M^\dagger=(M^{\mathrm{T}})^*$ is the complex conjugate of the transpose of M. Clearly, it is a multi-path linear program over $\mathbb{C}^{d^2}$.

In [35], the set of non-terminating inputs (diverging states, in their terminology) of the loop in Example 6 plays a key role in deciding the termination of nondeterministic quantum programs. We will show in the experiments that our approach can be used to discover termination/nontermination proofs of such quantum programs.

5.2. Execution of MPPs

Given an input $\boldsymbol{x}$, the behavior of MPP (7) on $\boldsymbol{x}$ is determined by the nondeterministic choices of $\boldsymbol{A}_i$ $(1 \le i \le l)$, and consequently all the possible executions form a tree.

Definition 9 (Execution Tree [24]). The *execution tree* of MPP (7) on input $\boldsymbol{x} \in \mathbb{K}^d$ is defined inductively as follows:

1. The root is the input value of $\boldsymbol{x}$.
2. A node $\widetilde{\boldsymbol{x}}$ is a leaf node if $G(\widetilde{\boldsymbol{x}}) \neq 0$; otherwise, $\widetilde{\boldsymbol{x}}$ has l children $\boldsymbol{A}_1(\widetilde{\boldsymbol{x}}), \boldsymbol{A}_2(\widetilde{\boldsymbol{x}}), \dots, \boldsymbol{A}_l(\widetilde{\boldsymbol{x}})$, and there is a directed edge from $\widetilde{\boldsymbol{x}}$ to $\boldsymbol{A}_i(\widetilde{\boldsymbol{x}})$, labeled by i, for $1 \le i \le l$.

Now we consider paths in the execution tree. Let $\Sigma = \{1, 2, \dots, l\}$ denote the set of indices of the assignments $\boldsymbol{A}_i$ $(1 \le i \le l)$. For any finite string $\sigma = a_1 a_2 \cdots a_s \in \Sigma^*$, we write $|\sigma| = s$ for its length and $\mathrm{Pre}(\sigma) = \{a_1 \cdots a_i \mid i = 0, 1, \dots, s-1\}$ for the set of its proper prefixes. The concatenation of two strings $\sigma = a_1 \cdots a_s$ and $\tau = b_1 \cdots b_t$ is written as $\sigma\tau = a_1 \cdots a_s b_1 \cdots b_t$. For simplicity, we let $\boldsymbol{A}_\sigma = \boldsymbol{A}_{a_s} \circ \cdots \circ \boldsymbol{A}_{a_2} \circ \boldsymbol{A}_{a_1}$ and $\boldsymbol{A}_\epsilon = \mathbf{id}$ (i.e $\boldsymbol{A}_\epsilon(\boldsymbol{x}) \equiv \boldsymbol{x}$) for the empty string ϵ. Then $\boldsymbol{A}_{\sigma\tau} = \boldsymbol{A}_\tau \circ \boldsymbol{A}_\sigma$. Similarly, for an infinite sequence $\sigma = a_1 a_2 \cdots \in \Sigma^\omega$, we define $|\sigma| = \infty$ and $\mathrm{Pre}(\sigma) = \{a_1 \cdots a_i \mid i = 0, 1, \dots\}$. The concatenation of a finite string $\tau = b_1 \cdots b_t \in \Sigma^*$ and an infinite string $\sigma = a_1 a_2 \cdots \in \Sigma^\omega$ is defined as $\tau\sigma = b_1 \cdots b_t a_1 a_2 \cdots \in \Sigma^\omega$.

Then any finite or infinite path from the root in the execution tree can be identified by a finite or infinite string over Σ. Specifically, for any $\sigma = a_1 a_2 \cdots \in \Sigma^* \cup \Sigma^\omega$, the corresponding execution path is

$$\boldsymbol{x} \xrightarrow{\boldsymbol{A}_{a_1}} \boldsymbol{A}_{a_1}(\boldsymbol{x}) \xrightarrow{\boldsymbol{A}_{a_2}} \boldsymbol{A}_{a_2}(\boldsymbol{A}_{a_1}(\boldsymbol{x})) \xrightarrow{\boldsymbol{A}_{a_3}} \cdots.$$

Moreover, any node in the execution tree is of the form $\boldsymbol{A}_\sigma(\boldsymbol{x})$, where $\sigma = a_1 a_2 \cdots \in \Sigma^*$ represents the execution history. Its ancestor nodes are $\boldsymbol{A}_\tau(\boldsymbol{x})$, with $\tau \in \mathrm{Pre}(\sigma)$. According to the definition of the execution tree, we have $G(\boldsymbol{A}_\tau(\boldsymbol{x})) = 0$. Then all paths in the execution tree are given as follows.

Definition 10 (Execution Paths [24]). The set of *execution paths* of MPP (7) for an input $\boldsymbol{x} \in \mathbb{K}^d$ is defined as

$$\mathrm{Path}(\boldsymbol{x}) \triangleq \{\sigma \in \Sigma^* \cup \Sigma^\omega \mid \forall \tau \in \mathrm{Pre}(\sigma) :\ G(\boldsymbol{A}_\tau(\boldsymbol{x})) = 0\}.$$

For any path σ, we write the set of corresponding polynomials as $T_\sigma^- \triangleq \{G \circ \boldsymbol{A}_\tau \mid \tau \in \mathrm{Pre}(\sigma)\}$. Then we have

$$\sigma \in \mathrm{Path}(\boldsymbol{x}) \iff \boldsymbol{x} \in Z(T_\sigma^-).$$

5.3. Termination of MPPs

Now we formalize the meaning of "termination" of MPP (7). Intuitively, "a program terminates" means that its execution will be accomplished within a finite number of steps, i.e., the execution tree is finite. Formally, we have

Definition 11 (Termination).

1. For an input $\boldsymbol{x} \in \mathbb{K}^d$, MPP (7) is *terminating* if $|\mathrm{Path}(\boldsymbol{x})| < \infty$, i.e., $\mathrm{Path}(\boldsymbol{x})$ is a finite set; it is *non-terminating* otherwise.
2. $D \triangleq \{\boldsymbol{x} \in \mathbb{K}^d \mid |\mathrm{Path}(\boldsymbol{x})| = \infty\}$ is the *set of non-terminating inputs* (**NTI**) of MPP (7).

By König's lemma [36], the execution tree is infinite if and only if it contains an infinite path, i.e.,

$$|\mathrm{Path}(\boldsymbol{x})| = \infty \iff \mathrm{Path}(\boldsymbol{x}) \cap \Sigma^\omega \neq \emptyset$$

for all $\boldsymbol{x} \in \mathbb{K}^d$. The NTI can thus be expressed as

$$D = \bigcup\nolimits_{\sigma \in \Sigma^\omega} Z(T_\sigma^-). \tag{8}$$

We are further interested in program termination within a fixed number of iterations:

Definition 12 (n-Termination). For a natural number $n \in \mathbb{N}$,

1. MPP (7) is *n-terminating* for an input $\boldsymbol{x} \in \mathbb{K}^d$, if $|\sigma| \le n$ for all $\sigma \in \mathrm{Path}(\boldsymbol{x})$.
2. The *set of n-non-terminating inputs* (n-**NTI**) of MPP (7) is defined as $D_n \triangleq \{\boldsymbol{x} \in \mathbb{K}^d \mid \exists \sigma \in \mathrm{Path}(\boldsymbol{x}) :\ |\sigma| > n\}$.

The n-NTI can be expressed in a similar way to (8). For clarity, we set $T_\sigma \triangleq T_\sigma^- \cup \{G \circ \boldsymbol{A}_\sigma\}$, then it is easy to verify that $T_{\sigma a}^- = T_\sigma$ for any $\sigma \in \Sigma^*$ and $a \in \Sigma$. In particular, $T_\epsilon = \{G\}$.

Proposition 7.

$D_n = \bigcup_{|\sigma|=n} Z(T_\sigma)$.

Proof. From Definition 12-2,

$$\begin{aligned}
\boldsymbol{x} \in D_n &\iff \exists \sigma = a_1 a_2 \cdots \in \mathrm{Path}(\boldsymbol{x}) :\ |\sigma| > n \\
&\iff \exists \sigma' = a_1 a_2 \cdots a_{n+1} \in \mathrm{Path}(\boldsymbol{x}) :\ |\sigma'| = n+1 \\
&\iff \exists \sigma'' = a_1 a_2 \cdots a_n \in \Sigma^n, a_{n+1} \in \Sigma :\ \boldsymbol{x} \in Z(T_{\sigma''a_{n+1}}^-) = Z(T_{\sigma''}).
\end{aligned}$$

The result holds. □

In particular, we note that $D_0 = Z(T_\epsilon) = Z(\{G\})$.

6. Decidability of the termination problem

We prove that it is decidable whether an MPP of the form (7) is terminating on a given input $\boldsymbol{x}$. This is done by an algorithm computing the NTI D for an MPP, and thus it suffices to decide whether $\boldsymbol{x} \in D$.

Our idea is essentially to compute the ideal of a polynomial ascending chain. More clearly, consider the simplest case of $l = 1$, i.e., an MPP with a single path. In this case, the NTI is

$$D = \{\boldsymbol{x} \mid G(\boldsymbol{A}_1^k(\boldsymbol{x})) = 0 \text{ for all } k \geq 0\},$$

which is the set of zeros of the ideal I generated by a infinite sequence of polynomials $G, G\circ\boldsymbol{A}_1, \cdots, G\circ\boldsymbol{A}_1^k, \cdots$. From Theorem 1, there exists a number N such that

$$I = I_N \triangleq \langle G, G\circ\boldsymbol{A}_1, \ldots, G\circ\boldsymbol{A}_1^N\rangle$$

and thus $D = D_N = Z(I_N)$. One can certify that the program does not terminate on input $\boldsymbol{x}$, i.e., $\boldsymbol{x} \in D$, by iteratively checking $G\big(\boldsymbol{A}_1^n(\boldsymbol{x})\big) = 0$ with $n = 0, 1, \ldots, N$. Therefore, the key to certifying termination is to determine when the number N is reached (and thus the procedure stops). This is achieved by two steps:

1. (*Strict chain condition*) Suppose

   $$G(\boldsymbol{A}_1^k(\boldsymbol{x})) \in I_{k-1} = \langle G(\boldsymbol{x}), G(\boldsymbol{A}_1(\boldsymbol{x})), \ldots, G(\boldsymbol{A}_1^{k-1}(\boldsymbol{x}))\rangle,$$

   then it is obtained by replacing $\boldsymbol{x}$ with $\boldsymbol{A}_1(\boldsymbol{x})$ that

   $$G(\boldsymbol{A}_1^{k+1}(\boldsymbol{x})) \in \langle G(\boldsymbol{A}_1(\boldsymbol{x})), \ldots, G(\boldsymbol{A}_1^k(\boldsymbol{x}))\rangle \subseteq I_k.$$

   Namely, $I_k = I_{k-1}$ implies $I_{k+1} = I_k$. It actually means that the ascending chain of ideals I_n (resp. the descending chain of D_n) is strict.
2. (*Complexity*) Let $N = \min\{n \mid G(\boldsymbol{A}_1^n(\boldsymbol{x})) \in I_{n-1}\} - 1$. It is exactly the number of iterations we need. Moreover, let $g = \deg(G)$, $a = \deg(\boldsymbol{A}_1)$ and $f(i) = ga^{i-1}$, then $\deg(G\circ\boldsymbol{A}^i) \leq f(i)$. The ascending chain of I_n is thus f-bounded and $N \leq L(d, f)$ from Definition 4.

Similar results for general MPPs are present in the rest of this section.

6.1. Characterization of the NTI

We investigate the mathematical structure of the NTI, which immediately implies the decidability of the termination problem. Denote by $L(d, g, a)$ the obtained maximal length $L(d, f)$ of polynomial ascending chains under a specific control function $f(i) = ga^{i-1}$, which will play a key role in representing the complexity of several algorithms in the forthcoming discussion.

Definition 13 (Backward subset). For any $X \subseteq \mathbb{K}^d$, its *backward subset* under MPP (7) is defined as

$$\mathrm{Back}(X) \triangleq \bigcup_{a\in\Sigma}(X \cap \boldsymbol{A}_a{}^{-1}(X)) = \{x \in X \mid \exists a \in \Sigma : \boldsymbol{A}_a(x) \in X\}.$$

Lemma 2. *For any $n \geq 0$, $\mathrm{Back}(D_n) = D_{n+1}$.*

Proof. First, we prove that $D_{n+1} \subseteq \mathrm{Back}(D_n)$. For any $\boldsymbol{x} \in D_{n+1}$, it follows from Proposition 7 that $\boldsymbol{x} \in Z(T_\sigma)$ for some $\sigma = a_1 \cdots a_{n+1} \in \Sigma^{n+1}$. Then by definition $\boldsymbol{A}_{a_1}(\boldsymbol{x}) \in Z(T_{a_2\cdots a_{n+1}}) \subseteq D_n$, i.e., $\boldsymbol{x} \in \boldsymbol{A}_{a_1}^{-1}(D_n)$. Note that $\boldsymbol{x} \in D_{n+1} \subseteq D_n$, we have $\boldsymbol{x} \in D_n \cap \boldsymbol{A}_{a_1}^{-1}(D_n) \subseteq \mathrm{Back}(D_n)$.

Conversely, to show that $\mathrm{Back}(D_n) \subseteq D_{n+1}$, suppose $\boldsymbol{x} \in \mathrm{Back}(D_n)$. We have $\boldsymbol{x} \in D_n$ and $\boldsymbol{x} \in \boldsymbol{A}_a^{-1}(D_n)$ for some $a \in \Sigma$. Then $G(\boldsymbol{x}) = 0$ and $\boldsymbol{A}_a(\boldsymbol{x}) \in Z(T_\sigma)$ for some $\sigma \in \Sigma^n$. As $\{G\} \cup \{f\circ\boldsymbol{A}_a \mid f \in T_\sigma\} = T_{a\sigma}$, we obtain $\boldsymbol{x} \in Z(T_{a\sigma}) \subseteq D_{n+1}$. This completes the proof. □

Remark 4. The Back operator defined in Definition 13 is reminiscent of the standard pre-image operator employed in transition systems; see, e.g., [18]. Specifically, the pre-image operator for a relation $R \subseteq S \times S$ of a set S is

$$\mathrm{pre}_R(X) \triangleq \{s \in S \mid \exists x \in X : (s, x) \in R\}$$

for any subset $X \subseteq S$. Then, the pre-image operator for the transition relation induced by an MPP is defined as

$$\mathrm{pre}(X) \triangleq \{\boldsymbol{x} \in \mathbb{K}^d \mid G(\boldsymbol{x}) = 0 \text{ and } \exists a \in \Sigma : \boldsymbol{A}_a(\boldsymbol{x}) \in X\} = \bigcup_{a\in\Sigma}(D_0 \cap \boldsymbol{A}_a{}^{-1}(X)).$$

Note that $D_{n+1} = \mathrm{pre}(D_n)$ for all $n \geq 0$. Therefore, Back and pre coincide on the sets D_n, $n = 0, 1, \ldots$, and the results in Theorem 5 also follow from [18]. However, they may differ in the general case. In particular, while $\mathrm{Back}(X)$ is always a subset of X, $\mathrm{pre}(X)$ may not be. We use the operator Back in the paper rather than pre because the former is defined independently of G, the guard polynomial of the MPP. This makes it particularly useful in the study of extensions of MPPs with loop guards of different types, as discussed in Section 7.

Theorem 5.

1. *clause]ite:algebraic For any $n \geq 0$, D_n is an algebraic set.*

2. *clause]ite:chain The algebraic sets D_n form a strict descending chain*

$$D_0 \supset D_1 \supset \cdots \supset D_N = D_{N+1} = \cdots,$$

and the chain length N satisfying $N \leq L(d,g,a) - 1$, where $g = \deg(G)$ and $a = \deg(\boldsymbol{A}_1) + \ldots + \deg(\boldsymbol{A}_l)$.

3. *clause]ite:D=DN $D = D_N$, i.e., the fixed-point of the chain is exactly the NTI. Consequently, the NTI is also an algebraic set.*

Proof. Clause 1 is directly from Proposition 7. The strict descending chain condition in Clause 2 follows straightforwardly from Lemma 2, and the bound $L(d,g,a)$ of length N follows from Proposition 10, noting that $N \leq \hat{N}$.

To prove Clause 3, it suffices to show $D_N \subseteq D$, since $D \subseteq D_n$ $(\forall n \geq 0)$ can be verified from the definition. For any $n \geq N$ and any $\boldsymbol{x} \in D_N = D_n$, we have $\exists \sigma \in \mathrm{Path}(\boldsymbol{x}) : |\sigma| > n$. Therefore, $|\mathrm{Path}(\boldsymbol{x})| = \infty$ and $\boldsymbol{x} \in D$. □

Theorem 5 suggests immediately that the NTI of MPP (7) can be obtained by computing $D = D_{L(d,g,a)-1}$ – where the value of $L(d,g,a)$ can be calculated via Theorem 4 and Definition 6 – and thus the termination of MPP (7) on an input $\boldsymbol{x}$ (i.e., $\boldsymbol{x} \notin D$) is certified by verifying $\bigvee_{f \in T_\sigma} f(\boldsymbol{x}) \neq 0$ for all $|\sigma| = L(d,g,a) - 1$ by Proposition 7. Note that $L(d,g,a) - 1$ is independent of the input $\boldsymbol{x}$ and the choices of execution paths, and thus a *uniform* upper bound on the running time of terminating executions. Formally,

Theorem 6 (Decidability). *MPP (7) terminates on an input if and only if it terminates within $L(d,g,a) - 1$ iterations of the WHILE-loop. Therefore, the termination problem of MPP (7) on an input is decidable by the following decision procedure: For each execution path of length $L(d,g,a) - 1$, run the program from the input; if for at least one path, the loop condition remains true during the execution, the answer is* `non-terminating`*, and* `terminating` *otherwise.*

Remark 5. The decision procedure of certifying termination in Theorem 6 is, however, of limited practical use. Since L grows Ackermannianly in d, the value of $L(d,g,a)$ is typically huge and computing T_σ with $|\sigma| = L(d,g,a) - 1$ is infeasible in practice. To derive a practically applicable algorithm, we turn to compute $D = D_N$ by iteratively calculating the n-NTI D_n for $n = 0, 1, \ldots$, until the stabilization condition $D_n = D_{n+1}$ is reached. This idea of computing D will be exploited in Algorithms 1 and 2. Although N is explicitly upper bounded by $L(d,g,a) - 1$ (in contrast to the implicit characterization given by the fast-growing hierarchy in [34]), the *tightness* of such a bound remains an open question. In fact, our experiments in Section 8 show that the exact value of N is always much smaller than the bound, which suggests that our bound can be further improved in future research.

The following properties of NTI will be used in Section 7.2 to tackle MPPs with inequality guards.

A set $X \subseteq \mathbb{K}^d$ is said to be *transitive* under MPP (7), if it can be finitely decomposed as $X = X_1 \cup X_2 \cup \cdots \cup X_n$ satisfying that for any $i \in \{1, 2, \ldots, n\}$, there exists $a \in \Sigma$ and $j \in \{1, 2, \ldots, n\}$ such that $\boldsymbol{A}_a(X_i) \subseteq X_j$. We have the following result:

Lemma 3. *Let $X \subseteq D_0$ be transitive, then $X \subseteq D$. Moreover, for any $\boldsymbol{x} \in X$, $\mathrm{Path}(\boldsymbol{x})$ contains an infinite path of the form $\sigma_0 \sigma_1^\omega$, where $\sigma_0, \sigma_1 \in \Sigma^*$, and σ_1 is a non-empty string.*

Proof. The transitivity of $X = X_1 \cup X_2 \cup \cdots \cup X_n$ allows existence of two functions $a : \{1, 2, \ldots, n\} \to \Sigma$ and $b : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that $\boldsymbol{A}_{a(i)}(X_i) \subseteq X_{b(i)}$ for all $i \in \{1, 2, \ldots, n\}$. Now for any $\boldsymbol{x} \in X$, $\boldsymbol{x} \in X_i$ for some $i \in \{1, \ldots, n\}$. Let $b_k = b^k(i)$ and $a_k = a(b^k(i))$ for all $k \geq 0$, we have the path

$$X_i = X_{b_0} \xrightarrow{\boldsymbol{A}_{a_0}} X_{b_1} \xrightarrow{\boldsymbol{A}_{a_1}} X_{b_2} \xrightarrow{\boldsymbol{A}_{a_2}} \cdots$$

satisfying $\boldsymbol{A}_{a_k}(X_{b_k}) \subseteq X_{b_{k+1}}$. Then for $\sigma = a_1 a_2 \cdots \in \Sigma^\omega$ and any $\tau \in \mathrm{Pre}(\sigma)$, we have $\boldsymbol{A}_\tau(X_i) = X_{b_{|\tau|}} \subseteq X \subseteq D_0$. So $\boldsymbol{A}_\tau(\boldsymbol{x}) \in D_0 = Z(\{G\})$, and thus $(G \circ \boldsymbol{A}_\tau)(\boldsymbol{x}) = 0$. Therefore $\sigma \in \mathrm{Path}(\boldsymbol{x})$ and $\boldsymbol{x} \in D$.

Moreover, note that $\{b_0, b_1, \ldots\} \subseteq \{1, 2, \ldots, n\}$ has at most n elements, then there exists $j \in \{0, 1, \ldots, n-1\}$ and $p \in \{1, \ldots, n\}$ such that $b_j = b_{j+p}$. Thus for all $k \geq 0$ and $t \geq j$, $b_{t+kp} = b^{t+kp}(i) = b^t(i) = b_t$. We have

$$\sigma = (a_1 a_2 \cdots a_{j-1})(a_j a_{j+1} \cdots a_{j+p-1})^\omega.$$

□

Theorem 7. *D is the largest transitive subset of D_0.*

Proof. According to Lemma 3, it suffices to prove that D is transitive. We use the irreducible decomposition $D = Y_1 \cup Y_2 \cup \cdots \cup Y_n$ to prove transitivity. For any $i \in \{1, 2, \ldots, n\}$, we have

$$Y_i \subseteq D = \mathrm{Back}(D) \subseteq \bigcup\nolimits_{a \in \Sigma} \boldsymbol{A}_a{}^{-1}(D) = \bigcup\nolimits_{a \in \Sigma} \bigcup\nolimits_{j=1}^{n} \boldsymbol{A}_a^{-1}(Y_j).$$

Note that Y_i is irreducible and for any $a \in \Sigma$ and any algebraic set $X = Z(T)$, $\boldsymbol{A}_a^{-1}(X) = Z(\{f \circ \boldsymbol{A}_a \mid f \in T\})$ is also algebraic. Then there exists $a \in \Sigma$ and $j \in \{1, 2, \ldots, n\}$ such that $Y_i \subseteq \boldsymbol{A}_a{}^{-1}(Y_j)$, i.e., $\boldsymbol{A}_a(Y_i) \subseteq Y_j$. □

Corollary 1. *For any $\boldsymbol{x} \in D$, there is an infinite path in $\mathrm{Path}(\boldsymbol{x})$ with a regular form $\sigma = \sigma_0 \sigma_1^\omega$, where $\sigma_0, \sigma_1 \in \Sigma^*$, and σ_1 is the non-empty string.*

6.2. Algorithms and their complexities

Now we are ready to formally present algorithms of computing D for MPP (7), or more precisely, of computing a finite set B of generating polynomials of D with $D = Z(B)$. Thus a program terminates on a given input $\boldsymbol{x}$ if and only if $\bigvee_{f\in B} f(\boldsymbol{x}) \neq 0$ which is decidable under the assumption in Remark 3-1. Furthermore, when the field $\mathbb{K} = \mathbb{R}$ and all polynomial coefficients are rational numbers, we can even check if a program terminates on a given input set X, a semi-algebraic set defined by a polynomial formula $\phi(\boldsymbol{x})$. It is equivalent to the unsatisfiability of $\bigwedge_{f\in B}(f(\boldsymbol{x}) = 0) \wedge \phi(\boldsymbol{x})$, which is decidable à la [31].

An algorithm for computing D is directly obtained from Theorem 5: it computes the fixed-point of the chain

$$D_0 \supset D_1 \supset \cdots \supset D_N = D_{N+1} = \cdots.$$

In the algorithm, D_n is represented by a set $S_n \triangleq \prod_{\sigma\in\Sigma^n} T_\sigma$ of generating polynomials, since by Proposition 7,

$$D_n = \bigcup_{|\sigma|=n} Z(T_\sigma) \overset{(*)}{=} Z\Big(\prod\nolimits_{\sigma\in\Sigma^n} T_\sigma\Big),$$

where the equality $(*)$ is from the fact that

$$Z(A_1) \cup Z(A_2) \cup \cdots \cup Z(A_k) = Z(A_1 \cdot A_2 \cdot \cdots \cdot A_k)$$

for sets $A_1, A_2, \ldots, A_k$ of polynomials. The algorithm incrementally computes $S_0, S_1, \ldots$ and stops as soon as the stabilization condition $Z(S_n) = Z(S_{n+1})$ is satisfied. The detailed procedure is presented in Algorithm 1, in which the stabilization condition is checked via the following proposition.

Proposition 8. *$D_n = D_{n+1}$ iff $f \in I(Z(T_\sigma))$ for all $\sigma \in \Sigma^n$ and all $f \in \prod_{\tau\in\Sigma^{n+1}} T_\tau$.*

Proof. Since D_n is always a superset of D_{n+1}, by Proposition 7, we have

$$\begin{aligned} D_n = D_{n+1} &\iff D_n \subseteq D_{n+1} \iff \bigcup\nolimits_{\sigma\in\Sigma^n} Z(T_\sigma) \subseteq \bigcup\nolimits_{\tau\in\Sigma^{n+1}} Z(T_\tau) \\ &\iff \forall\sigma\in\Sigma^n:\ Z(T_\sigma) \subseteq \bigcup\nolimits_{\tau\in\Sigma^{n+1}} Z(T_\tau) = Z(\prod\nolimits_{\tau\in\Sigma^{n+1}} T_\tau) \\ &\iff \forall\sigma\in\Sigma^n:\ \prod\nolimits_{\tau\in\Sigma^{n+1}} T_\tau \subseteq I(Z(T_\sigma)). \end{aligned}$$

This completes the proof. □

Algorithm 1: Computing NTI from n-NTI.

input: The dimension d, the polynomial G and the polynomial vectors $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_l$.
output: The integer $N = \min\{n \geq 0 \mid D_n = D_{n+1}\}$ and a basis S_0 of D_N.

set of polynomial sets $S_0 \leftarrow \emptyset$; ▷ for generating polynomials of D_n
set of polynomial sets $S_1 \leftarrow \{T_\epsilon\}$; ▷ for generating polynomials of D_{n+1}
bool $b \leftarrow$ False;
integer $n \leftarrow -1$;
while $\neg b$ **do**
  $S_0 \leftarrow S_1$; $S_1 \leftarrow \emptyset$; $n \leftarrow n+1$;
  for $T_\sigma \in S_0$ **do**
    for $a \leftarrow 1$ **to** l **do**
      $T_{\sigma a} \leftarrow T_\sigma \cup \{G \circ \boldsymbol{A}_{\sigma a}\}$;
      $S_1 \leftarrow S_1 \cup \{T_{\sigma a}\}$;
  /* test if $D_n = D_{n+1}$ */
  $b \leftarrow$ True;
  for $T_\sigma \in S_0$ **do**
    for $f \in \prod S_1$ **do**
      $b \leftarrow b \wedge (f \in I(Z(T_\sigma)))$;
return n, S_0;

Complexity of Algorithm 1. By Theorem 5, the **while** loop terminates after N iterations, with N bounded by $L(d, g, a)$. In the nth iteration, there are mainly two computation steps: the first one is to add the polynomial set T_τ into S_1 for each $\tau \in \Sigma^{n+1}$ and thus will be executed $O(|\Sigma^{n+1}|) = O(l^{n+1})$ times; the second step concerns checking the condition $f \in I(Z(T_\sigma))$ for all $\sigma \in \Sigma^n$ and all $f \in \prod_{\tau\in\Sigma^{n+1}} T_\tau$, leading to $O(l^n(n+2)^{l^{n+1}})$ times of the membership checking. Thus, the total time complexity is $O(l^N(N+2)^{l^{N+1}})$.

Remark 6. As mentioned in Section 3.2, the step of checking $f \in I(Z(T_\sigma))$ in Algorithm 1 is decidable via Tarski's approach [31] for field $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$ and rational polynomial coefficients; while for fields beyond $\mathbb{R}$ and $\mathbb{C}$, this approach becomes inapplicable and a method of Gröbner bases proposed in the latter part of this section will be applied. Here, we study a special case that the field $\mathbb{K}$ is a subfield of $\mathbb{C}$ (e.g., the subfields $\mathbb{Q}$ of rationals and $\overline{\mathbb{Q}}$ of algebraic complex numbers), and show that the termination problem of this case can be solved by leveraging the case of $\mathbb{K} = \mathbb{C}$: Suppose that the n-NTI is obtained in the case of a field $\mathbb{K}_1$, written as $D_n\big|_{\mathbb{K}=\mathbb{K}_1}$. Then, the n-NTI with restriction to a subfield $\mathbb{K}_2 \subseteq \mathbb{K}_1$ can be derived as

$$D_n\Big|_{\mathbb{K}=\mathbb{K}_2} = D_n\Big|_{\mathbb{K}=\mathbb{K}_1} \cap \mathbb{K}_2^d.$$

Hence, an MPP of field $\mathbb{K}_2$ terminates on an input $\boldsymbol{x} \in \mathbb{K}_2$ (i.e., $\boldsymbol{x} \in D\big|_{\mathbb{K}=\mathbb{K}_2}$) if and only if $\boldsymbol{x} \in D\big|_{\mathbb{K}=\mathbb{K}_1}$, which can be checked by considering the field as $\mathbb{K}_1$. Moreover, this idea works even for subrings (rather than subfields) of $\mathbb{K}_1$. For instance, when the input value of $\boldsymbol{x}$ and the polynomial coefficients in MPP (7) are all integers (i.e., the subring $\mathbb{Z}$ of $\mathbb{C}$), one can interpret the field as $\mathbb{C}$ and compute by Algorithm 1 a set S of generating polynomials of the NTI $D\big|_{\mathbb{K}=\mathbb{C}}$, then the program terminates on input $\boldsymbol{x}$ if and only if $\bigvee_{f\in S} f(\boldsymbol{x}) \neq 0$, which is decidable as $\boldsymbol{x} \in \mathbb{Z}^d$ and $S \subseteq \mathbb{Z}[\boldsymbol{x}]$. Nonetheless, this method does not apply to *universal termination* for a subfield $\mathbb{K}_0$ of $\mathbb{C}$ (unless $\mathbb{K}_0 = \mathbb{R}$), i.e., to decide whether $D\big|_{\mathbb{K}=\mathbb{C}} \cap \mathbb{K}_0^d = \emptyset$. For instance, when the subfield is $\mathbb{Q}$, it is to determine if a set S of polynomials with rational coefficients has rational roots, which is generally undecidable (as related to the unsolvability of Diophantine equations [37]).

To address the case of an effective field $\mathbb{K}$ that is not a subfield of $\mathbb{C}$ (e.g., a finite field or the field of rational functions), we employ the method of Gröbner bases to refine the condition of $D_n = D_{n+1}$. Specifically, by constructing a Gröbner basis B_n of the set $\prod_{\sigma\in\Sigma^n} T_\sigma$ of generating polynomials of D_n, i.e., $D_n = Z(B_n)$, we check the condition $B_n = B_{n+1}$ instead of $D_n = D_{n+1}$, noting that the former always implies the latter (although the converse may not hold). This refined condition can be verified through ideal membership checking, which is decidable for any effective field. Formally, we express D_n by a set of generating polynomials as follows:

Proposition 9. *$D_n = Z(B_n)$ for any $n \in \mathbb{N}$, where $B_0 \triangleq \{G\}$ and $B_{n+1} \triangleq B_n \cup \prod_{a=1}^{l}(B_n \circ \boldsymbol{A}_a)$ for $n \in \mathbb{N}$. Here $B_n \circ \boldsymbol{A}_a \triangleq \{f \circ \boldsymbol{A}_a \mid f \in B_n\}$.*

Proof. We prove by induction on n. For $n = 0$, $D_0 = Z(\{G\}) = Z(B_0)$ from the definition; the claim holds. Suppose $D_n = Z(B_n)$, we prove that $D_{n+1} = Z(B_{n+1})$. It suffices to prove that $Z(B_{n+1}) = \mathrm{Back}(Z(B_n))$ from Lemma 2. From the definition of B_{n+1}, we have

$$\begin{aligned} Z(B_{n+1}) &= Z(B_n \cup \prod_{a=1}^{l}(B_n \circ \boldsymbol{A}_a)) = Z(B_n) \cap Z(\prod_{a=1}^{l}(B_n \circ \boldsymbol{A}_a)) \\ &= Z(B_n) \cap (\bigcup_{a\in\Sigma} Z(B_n \circ \boldsymbol{A}_a)) = \bigcup_{a\in\Sigma}(Z(B_n) \cap Z(B_n \circ \boldsymbol{A}_a)). \end{aligned}$$

On the other hand, from Definition 13,

$$\mathrm{Back}(Z(B_n)) = \bigcup_{a\in\Sigma}(Z(B_n) \cap \boldsymbol{A}_a{}^{-1}(Z(B_n))).$$

The proof is then completed by noting that

$$\begin{aligned} \boldsymbol{x} \in \boldsymbol{A}_a{}^{-1}(Z(B_n)) &\iff \boldsymbol{A}_a(\boldsymbol{x}) \in Z(B_n) \iff \forall f \in B_n : f\big(\boldsymbol{A}_a(\boldsymbol{x})\big) = 0 \\ &\iff \forall g \in B_n \circ \boldsymbol{A}_a : g(\boldsymbol{x}) = 0 \iff \boldsymbol{x} \in Z(B_n \circ \boldsymbol{A}_a), \end{aligned}$$

i.e., $Z(B_n \circ \boldsymbol{A}_a) = \boldsymbol{A}_a{}^{-1}(Z(B_n))$. □

Polynomials in B_n are incrementally computed by composition and multiplication of polynomials from G and $\boldsymbol{A}_a$, $a \in \Sigma$. Moreover, we compute a Gröbner basis of B_n so that the membership problem $f \in B_n$ can be efficiently solved by testing whether $\mathrm{Rem}(f, B_n) = 0$. Then, the condition $B_n = B_{n+1}$ can be verified by membership checking. In this way, a more effective and more efficient algorithm for computing the NTI is given in Algorithm 2, whose correctness is guaranteed by the following result.

Proposition 10. *$\forall n \in \mathbb{N} : D_{\hat{N}} = D_{\hat{N}+n}$ for $\hat{N} \triangleq \min\{n \mid B_n = B_{n+1}\}$, and $\hat{N} \leq L(d, g, a) - 1$, where $L(d, g, a)$ is the same as in Theorem 5.*

Proof. Note that $B_n \subseteq B_{n+1}$ for all n, thus $\hat{N}$ is well-defined due to Theorem 1. Moreover, $B_{n-1} = B_n$ implies $B_n = B_{n+1}$, according to the recurrence relation of B_n. Therefore, we have the strict ascending chain:

$$B_0 \subset B_1 \subset \cdots \subset B_{\hat{N}} = B_{\hat{N}+1} = \cdots.$$

On the other hand, $\deg(B_n) \leq g a^{n-1}$ since $\deg(B_0) = g$ and

$$\deg(B_{n+1}) = \deg(B_n \cup \prod_{a=1}^{l}(B_n \circ \boldsymbol{A}_a)) = \max(\deg(B_n), \deg(\prod_{a=1}^{l}(B_n \circ \boldsymbol{A}_a))) \leq \sum_{a=1}^{l}(\deg(B_n)\deg(\boldsymbol{A}_a)) = a\deg(B_n).$$

So, $\mathrm{gdeg}(\langle B_n\rangle) \leq g a^{n-1}$ and thus $\hat{N} \leq L(d, g, a) - 1$ according to Definition 4. □

Algorithm 2: Computing NTI by Gröbner bases.

input: The dimension d, polynomial G and polynomial vectors $\boldsymbol{A}_1, \dots, \boldsymbol{A}_l$.
output: The integer $\hat{N} = \min\{n \mid B_n = B_{n+1}\}$ and a Gröbner basis B of $D_{\hat{N}}$.

set of polynomials $B \leftarrow \{G\}$; ▷ for recording B_n
polynomial $f \leftarrow 0$;
polynomial $r \leftarrow 0$; ▷ to keep track of the remainder
integer $n \leftarrow -1$;
bool $c \leftarrow$ True;
/* test if $B_n = B_{n+1}$ */
while c **do**
  $c \leftarrow$ False; $n \leftarrow n+1$;
  for $f \in \prod_{i=1}^{l}(B \circ \boldsymbol{A}_i)$ **do**
    $r \leftarrow \mathrm{Rem}(f, B)$;
    if $r \neq 0$ **then**
      $B \leftarrow \mathrm{GB}(B \cup \{r\})$; $c \leftarrow$ True;
return n, B;

Complexity of Algorithm 2. There are $\hat{N}$ iterations of the **while** loop during the execution. In the nth iteration, there are at most $|B_{n+1}|$ polynomials to be added into B. Note that $|B_n| = O(2^{l^{n-1}})$. Thus, the time complexity is $O(2^{l^{\hat{N}}})$.

Remark 7. As discussed in Remark 6, the advantage of Algorithm 2 lies in its applicability to general effective fields (rather than $\mathbb{R}$ and $\mathbb{C}$ in Algorithm 1). However, as a trade-off, we have $\hat{N} \geq N$, i.e., Algorithm 2 may require more iterations than Algorithm 1. In the case of $\hat{N} > N$, we have $Z(B_n) = D_n = D_{n+1} = Z(B_{n+1})$, but $B_n \subset B_{n+1}$. This case is possible, as different polynomials may have the same set of roots. The example below demonstrates the case of $\hat{N} > N$:

$$\text{WHILE} \quad (x^2 + y^2 = 0) \quad \{(x, y) := (x, x + y);\}.$$

With this loop as input, the iteration in Algorithm 2 yields

$$\mathrm{GB}(B_0) = \{x^2 + y^2\},\ \mathrm{GB}(B_1) = \{x^2 + y^2, 2xy - y^2, y^3\},\ \mathrm{GB}(B_2) = \mathrm{GB}(B_3) = \{x^2, xy, y^2\},$$

thus $\hat{N} = 2$. For $\mathbb{K} = \mathbb{C}$, the output of Algorithm 1 is $N = 1$, as

$$D_0\big|_{\mathbb{K}=\mathbb{C}} = Z(B_0) = \{(c, \pm \mathrm{i}c) \mid c \in \mathbb{R}\},$$
$$D_1\big|_{\mathbb{K}=\mathbb{C}} = Z(B_1) = \{(0,0)\} = Z(B_2) = D_2\big|_{\mathbb{K}=\mathbb{C}};$$

while for $\mathbb{K} = \mathbb{R}$, the output of Algorithm 1 is $N = 0$, as

$$D_0\big|_{\mathbb{K}=\mathbb{R}} = D_0\big|_{\mathbb{K}=\mathbb{C}} \cap \mathbb{R}^2 = \{(0,0)\} = D_1\big|_{\mathbb{K}=\mathbb{C}} \cap \mathbb{R}^2 = D_1\big|_{\mathbb{K}=\mathbb{R}}.$$

7. Extensions and applications

In this section, we extend the obtained results on MPPs to more general polynomial programs, including polynomial guarded commands and MPPs with inequality guards. We further show potential applications to invariant generation.

7.1. Polynomial guarded commands

Consider a more general model of polynomial programs named *polynomial guarded commands* (PGCs), which are typical guarded commands [38] with all expressions being polynomials and guards being polynomial equalities:

Definition 14 (PGCs with Equality Guards). A *PGC with equality guards* is of the form

$$\begin{array}{ll} \text{DO} & G_1(\boldsymbol{x}) = 0 \longrightarrow \boldsymbol{x} := \boldsymbol{A}_1(\boldsymbol{x}); \\ \| & G_2(\boldsymbol{x}) = 0 \longrightarrow \boldsymbol{x} := \boldsymbol{A}_2(\boldsymbol{x}); \\ & \quad \vdots \\ \| & G_{l-1}(\boldsymbol{x}) = 0 \longrightarrow \boldsymbol{x} := \boldsymbol{A}_{l-1}(\boldsymbol{x}); \\ \| & G_l(\boldsymbol{x}) = 0 \longrightarrow \boldsymbol{x} := \boldsymbol{A}_l(\boldsymbol{x}); \\ \text{OD}, & \end{array} \quad \text{where} \tag{9}$$

1. $\boldsymbol{x} \in \mathbb{K}^d$ denotes the vector of program variables;

2. $G_i \in \mathbb{K}[\boldsymbol{x}]$ are polynomials. Like MPPs, a PGC admits general guards of the form $\bigvee_{i=1}^{M} \bigwedge_{j=1}^{N_i} G_{ij}(\boldsymbol{x}) = 0$;
3. $\boldsymbol{A}_i \in \mathbb{K}^d[\boldsymbol{x}]$ $(1 \le i \le l)$ are vectors of polynomials, describing the updates of program variables.

Remark 8.

1. Loosely speaking, given an input $\boldsymbol{x}$, whenever some guards G_i are satisfied, one of them is nondeterministically chosen and the corresponding assignment is taken. It repeats until none of the guards holds.
2. Evidently, when all $G_i(\boldsymbol{x})$ are identical, the PGC degenerates to an MPP; if $G_i(\boldsymbol{x}) = 0 \wedge G_j(\boldsymbol{x}) = 0$ has no solution in $\mathbb{K}$ for any $i \neq j$, the choice among these updates becomes deterministic.
3. Notice that the deterministic choice derived from (9) cannot be used to define a general deterministic choice

   $$\text{IF } G(\boldsymbol{x}) = 0 \text{ THEN } \boldsymbol{x} := \boldsymbol{A}_1(\boldsymbol{x}) \text{ ELSE } \boldsymbol{x} := \boldsymbol{A}_2(\boldsymbol{x}),$$

   as it is equivalent to

   $$\text{IF } G(\boldsymbol{x})=0 \longrightarrow \boldsymbol{x}:=\boldsymbol{A}_1(\boldsymbol{x}) \parallel G(\boldsymbol{x})\neq 0 \longrightarrow \boldsymbol{x}:=\boldsymbol{A}_2(\boldsymbol{x}) \text{ FI},$$

   which contains $G(\boldsymbol{x}) \neq 0$. In [6], Müller-Olm and Seidl proved that the Post Correspondence Problem (PCP) can be encoded into the extension of PGCs by allowing polynomial inequations in guards, indicating that termination of such an extension is undecidable in general.

Given an input $\boldsymbol{x}$, the set of paths starting from $\boldsymbol{x}$ is

$$\mathrm{Path}(\boldsymbol{x}) \triangleq \{\sigma \in \Sigma^* \cup \Sigma^\omega \mid \forall \tau \in \mathrm{Pre}(\sigma), i \in \Sigma : \tau i \in \mathrm{Pre}(\sigma) \cup \{\sigma\} \implies G_i(\boldsymbol{A}_\tau(\boldsymbol{x})) = 0\}.$$

For a path $\sigma \in \mathrm{Path}(\boldsymbol{x})$, the set of corresponding polynomials T_σ^- is $\{G_i(\boldsymbol{A}_\tau) \mid \tau i \in \mathrm{Pre}(\sigma) \cup \{\sigma\}\}$. Similarly, we have

$$\sigma \in \mathrm{Path}(\boldsymbol{x}) \iff \boldsymbol{x} \in Z(T_\sigma^-).$$

A PGC is *non-terminating* on a given input $\boldsymbol{x}$ if there exists $\sigma \in \mathrm{Path}(\boldsymbol{x})$ s.t. $|\sigma| = \infty$, otherwise it is *terminating*. We denote the set of non-terminating inputs of the PGC by D.

Analogous to MPPs, (9) is called n-terminating on an input $\boldsymbol{x} \in \mathbb{K}^d$, if $|\sigma| \le n$ for all $\sigma \in \mathrm{Path}(\boldsymbol{x})$, and the set of n-non-terminating inputs of (9) is defined as

$$D_n \triangleq \{\boldsymbol{x} \in \mathbb{K}^d \mid \exists \sigma \in \mathrm{Path}(\boldsymbol{x}) : |\sigma| > n\}.$$

Analogous to Theorem 5, the NTI D and n-NTI D_n of a PGC have the properties:

Theorem 8.

1. *For any $n \ge 0$, D_n is an algebraic set.*
2. *The algebraic sets D_n form a strict descending chain*

   $$D_0 \supset D_1 \supset \cdots \supset D_M = D_{M+1} = \cdots,$$

   with $M \le L(d, g, a)$ where $g = \max\{\deg(G_1), \ldots, \deg(G_l)\}$ and $a = \deg(\boldsymbol{A}_1) + \cdots + \deg(\boldsymbol{A}_l)$.
3. $D = D_M$.

Based on Theorem 8, algorithms akin to Algorithms 1 and 2 for computing the NTI of a given PGC can be obtained without substantial change.

7.2. MPPs with inequality guards

The method proposed in Section 6 becomes inapplicable as soon as inequalities are involved in the guards of MPPs. In fact, the termination problem in this case is generally undecidable (even not semi-decidable [7]) and thus cannot be completely solved. However, an incomplete method for proving nontermination can be achieved by our results. The basic idea is to strengthen an inequality guard by an equality and then try to prove that the obtained MPP is non-terminating under the strengthened guard.

For example, consider the following MPP with inequality guard:

$$\texttt{ineq} \quad \text{WHILE } (x \ge 0) \left\{ \begin{array}{l} (x, y) := (x - 3, y - 3); \\ \parallel\ (x, y) := (x - 3y, y^2 + x^2); \\ \parallel\ (x, y) := (1, 0); \end{array} \right\}$$

Denote by `ineq'` the program obtained by replacing $x \ge 0$ with $x - y^2 = 0$ in `ineq`. Obviously, $x - y^2 = 0 \implies x \ge 0$, and thus `ineq` is non-terminating for every non-terminating input of `ineq'`. By our method one can easily verify that with input $(x_0, y_0) = (1, -1)$ and $(4, 2)$, `ineq` does not terminate.

The key to proving nontermination using this method is finding appropriate strengthened polynomial equalities $G(\boldsymbol{x}) = 0$. According to Corollary 1, the program does not terminate on input $\boldsymbol{x}$ if and only if $\sigma_0\sigma_1^\omega \in \mathrm{Path}(\boldsymbol{x})$ for some $\sigma_0, \sigma_1 \in \Sigma^*$. Hence, the set of all possible polynomials G can be written as $\bigcup_{\sigma_0,\sigma_1\in\Sigma^*} I_{\sigma_0,\sigma_1}$, where

$$I_{\sigma_0,\sigma_1} \triangleq \{G \in \mathbb{R}[\boldsymbol{x}] \mid \sigma_0\sigma_1^\omega \in \mathrm{Path}(\boldsymbol{x})\} = \{G \in \mathbb{R}[\boldsymbol{x}] \mid G(\boldsymbol{A}_\tau(\boldsymbol{x})) = 0 \text{ for all } \tau \in \mathrm{Pre}(\sigma_0\sigma_1^\omega)\}.$$

So, G is an invariant along with a path $\sigma_0\sigma_1^\omega$ and thus can be computed by existing invariant-generation procedures which are discussed in Sections 7.3 and 9. Then it suffices to find one of them that is stronger than the original guard.

For illustration, we propose a concrete procedure of generating G for $\mathbb{K} = \mathbb{Q}$. Let $g(\boldsymbol{x}) \geq 0$ be the polynomial inequality guard. We find the strengthened polynomial of the form $G(\boldsymbol{x}) = (1 + p(\boldsymbol{x}))g(\boldsymbol{x}) - q(\boldsymbol{x})$, where $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ are sum-of-squares polynomials (i.e., $\sum_i s_i^2(\boldsymbol{x})$ with $s_i(\boldsymbol{x}) \in \mathbb{Q}[\boldsymbol{x}]$) whose coefficients are parameters. Then $G(\boldsymbol{x}) = 0$ implies $g(\boldsymbol{x}) = q(\boldsymbol{x})/(1 + p(\boldsymbol{x})) \geq 0$. The parameters in $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ can be instantiated by solving the constraints $G(\boldsymbol{A}_\tau(\boldsymbol{x})) = 0$, $\tau \in \mathrm{Pre}(\sigma_0\sigma_1^\omega)$.

7.3. Invariant generation for MPPs

Now consider a variant problem: Given vectors of polynomials $\boldsymbol{A}_i$ and an set D of inputs, how to discover all *polynomial invariants* of the form $G(\boldsymbol{x}) = 0$ for the MPP with true as the loop guard and $\boldsymbol{x} := \boldsymbol{A}_1 \parallel \cdots \parallel \boldsymbol{x} := \boldsymbol{A}_l$ as the loop body? When $l = 1$, it is actually the dual problem to termination: A subset D of the NTI is given, while the polynomial G in the loop guard is unknown and needs to be found. Consider, for example,

$$(x_0, y_0) = (0, 0); \quad \text{WHILE} \ \ (\text{true}) \ \ \{(x, y) := (x + 1, y + 2x + 1)\}$$

for which $G(x, y) = x^2 - y$ serves as an invariant, since $(0, 0)$ is a non-terminating input of the MPP with $G(x, y) = 0$ in the loop guard.

As argued in [39], generating polynomial equality invariants for MPPs is interesting yet challenging, with promising applications in program verification. In the literature, however, only a partial solution to the problem is available, which requires the so-called *solvability assumption* [39]. In solvable MPPs, assignments $\boldsymbol{A}_i$ are essentially linear. In what follows, we show that by the approach proposed in this paper one can solve this problem without the solvability assumption (provided a bound on the degree of $G(\boldsymbol{x})$ instead). That is, given an MPP, a set D of inputs and any integer k, we can generate all polynomial invariants of the form $G(\boldsymbol{x}) = 0$ with $\deg(G) \leq k$ for the MPP.

Our method is essentially an application of Theorems 5 and 6. Consider an MPP with polynomial assignments $\boldsymbol{A}_i$, $1 \leq i \leq l$, and let $a = \max\{\deg(\boldsymbol{A}_1), \ldots, \deg(\boldsymbol{A}_l)\}$. Then polynomial $G(\boldsymbol{x})$ is an invariant if and only if the MPP of form (7) is always non-terminating for all execution paths and on all inputs in D. From Theorems 5 and 6, it suffices to consider execution paths with length $L(d, \deg(G), a) - 1$ instead of infinite execution paths. Formally, for any positive integer k, let $\Sigma_k = \{\sigma \in \Sigma^* \mid |\sigma| \leq L(d, k, a) - 1\}$.

Definition 15 (k-Invariants)**.** The set of *k-invariants* of the MPP is defined as $\mathrm{Inv}_k(D) \triangleq \{G \in \mathbb{K}[\boldsymbol{x}] \mid \deg(G) \leq k,\ G(\boldsymbol{A}_\sigma(\boldsymbol{x})) = 0 \text{ for all } \boldsymbol{x} \in D,\ \sigma \in \Sigma_k\}$.

Thanks to the explicit bound $L(d, k, a) - 1$, the number of all $\sigma \in \Sigma_k$ as well as the number of constraints on G is finite for a single input $\boldsymbol{x}$. The coefficients of G can thus be computed for a finite set D using standard constraint-solving techniques, noting that $\mathrm{Inv}_k(D)$ forms a linear space over $\mathbb{K}$ and a basis of it can be computed from the constraints. This ensures that the k-invariants are computable. Moreover, the computability of $\mathrm{Inv}_k(D)$ holds also for $\mathbb{K} = \mathbb{R}$ and a semi-algebraic set D of inputs à la [31].

By Theorems 5 and 6, we obtain the soundness of this invariant generation approach: all polynomials in $\mathrm{Inv}_k(D)$ are indeed invariants of the MPP, i.e.,

Proposition 11. *For any $G \in \mathrm{Inv}_k(D)$, the MPP with $G(\boldsymbol{x}) = 0$ as the loop guard and $\boldsymbol{A}_i$ as assignments does not terminate for any input in D and any path in Σ^ω.*

Completeness of the approach is obtained by noting that any polynomial invariant of degree k is in $\mathrm{Inv}_k(D)$. Therefore,

Theorem 9. *For an MPP on input set D, $\mathrm{Inv}_k(D)$ is exactly the set of all polynomial equality invariants of degree $\leq k$.*

The set of all polynomial equality invariants of an MPP can then be written as $\mathrm{Inv}(D) \triangleq \bigcup_k \mathrm{Inv}_k(D)$. Note that computing the set $\mathrm{Inv}(D)$ is in general impossible [40]; in fact, deciding if $\mathrm{Inv}(D) \neq \emptyset$, i.e., if $\mathrm{Inv}_k(D) \neq \emptyset$ for some k, is a difficult problem. Hence, it is reasonable to consider the invariants in $\mathrm{Inv}_k(D)$, i.e., polynomial invariants of a fixed degree k. However, most existing approaches in the literature are incomplete for generating polynomials in $\mathrm{Inv}_k(D)$. For instance, Rodríguez-Carbonell and Kapur's method is incomplete unless the program is *linear* [41] or *solvable* [39]; The computation of the strongest algebraic invariant by Hrushovski et al. [40] works only for affine programs; Colón's approach based on abstract interpretation [42] of approximating the set of invariants may miss some polynomials in the result (due to the use of pseudo ideals instead of ideals). In contrast, our method guarantees completeness of computing $\mathrm{Inv}_k(D)$ for general polynomial programs. To the best of our knowledge, this is the first work that can compute all degree-k ploynomial equality inviariants for every k. In additional, it provides an explicit bound $L(d, k, a)$ on the number of constraints. Nonetheless, we remark that such a bound is of limited practical use as it is typically huge, as discussed in Remark 5.

8. Implementation and experiments

We have implemented the proposed algorithms in Maple[3] for discovering non-terminating inputs for MPPs. The implementation takes an MPP as input, and gives the number N of iterations and a set of generating polynomials of the NTI as output. The procedure of checking termination on an input value $\boldsymbol{x}$ – by verifying if $\boldsymbol{x}$ is a common root of those generating polynomials – is straightforward and thus excluded from the implementation. Checking $f \in I(Z(T_\sigma))$ in Algorithm 1 is implemented by computing the radical ideals

[3] Available at https://github.com/Chenms404/MPPs.

$$\texttt{liu2}\quad \text{WHILE}\quad (x^2+1-y=0)\quad \left\{\begin{array}{l} (x,y) \coloneqq (x,x^2y);\\ \parallel\ (x,y) \coloneqq (-x,y);\end{array}\right\}$$

$$\texttt{prod}\quad \text{WHILE}\quad ((x-1)^2+(y-1)^2+z^2=0)\quad \left\{\begin{array}{l} (x,y,z) \coloneqq (2x,(y-1)/2,x+z);\\ \parallel\ (x,y,z) \coloneqq (2x,y/2,z);\end{array}\right\}$$

$$\texttt{liu3}\quad \text{WHILE}\quad ((x-1)(x-2)=0)\quad \left\{\begin{array}{l} x \coloneqq 1-x^2;\\ \parallel\ x \coloneqq x+1;\end{array}\right\}$$

$$\texttt{loop}\quad \text{WHILE}\quad ((x-2)^2+y^2=0)\quad \left\{\begin{array}{l} (x,y) \coloneqq (x+4,y);\\ \parallel\ (x,y) \coloneqq (x+2,y+1);\end{array}\right\}$$

$$\texttt{liu4}\quad \text{WHILE}\quad (x-z^3=0\vee y-z^2=0)\quad \left\{\begin{array}{l} (x,y,z) \coloneqq (x,y+2z+1,z+1);\\ \parallel\ (x,y,z) \coloneqq (x-3y+3z-1,y+2z-1,z-1);\end{array}\right\}$$

$$\texttt{var4}\quad \text{WHILE}\quad (w+x+y+z=0)\quad \left\{\begin{array}{l} (w,x,y,z) \coloneqq (w-2,x,y+2z+1,z+w);\\ \parallel\ (w,x,y,z) \coloneqq (w^2,x-3y+3z-1,y+2z-1,z-1);\end{array}\right\}$$

$$\texttt{qtwk}\quad \text{WHILE}\quad (x_2=0)\quad \left\{\begin{array}{l} \boldsymbol{x} \coloneqq \frac{1}{\sqrt{3}}\begin{bmatrix}1&1&0&-1\\1&-1&1&0\\0&1&1&1\\1&0&-1&1\end{bmatrix}\boldsymbol{x};\\ \parallel\ \boldsymbol{x} \coloneqq \frac{1}{\sqrt{3}}\begin{bmatrix}1&1&0&1\\-1&1&-1&0\\0&1&1&-1\\1&0&-1&-1\end{bmatrix}\boldsymbol{x};\end{array}\right\} \quad \text{with}\quad \boldsymbol{x}=\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}$$

Fig. 1. Example MPPs used for evaluation.

of T_σ, which essentially assumes that $\mathbb{K}$ is algebraically closed, i.e., is $\mathbb{C}$ in all examples; while implementation of Algorithm 2 allows the use of general $\mathbb{K}$. The computations of Gröbner bases and ideal membership attribute to the MAPLE package `Groebner` and `PolynomialIdeals`, respectively. Below, we demonstrate our approach with examples from the literature (see Fig. 1), evaluated on a 2.4GHz Intel Core-i5 processor with 16GB RAM running macOS Catalina.

Remark 9. Notice that all the experiments are conducted mainly for the purpose of testing and demonstrating the proposed algorithms. A fully automated tool with applications to real programs is not provided, due to the high complexity of the underlying procedures: the membership problem of polynomial ideals is EXPSPACE-complete in the size of the problem instance [29], while the computation of Gröbner bases has even higher worst-case complexity. The same issue applies already to the incomplete method [24] that only approximates the NTI and checks a group of sufficient (yet not necessary) conditions for termination and/or nontermination.

The following observations can be drawn from the experimental results collected in Table 1:

1. The MAPLE implementation of Algorithm 2 works efficiently: it outperforms Algorithm 1 and solves all examples successfully within 6 s in total. Algorithm 1, however, tends to be strenuous when l or the expected N reaches 2, and gets timeouts for cases with $N \geq 3$.
2. N equals roughly to d in these examples, at least not getting too large in terms of d, which well explains that despite a worst-case complexity doubly exponential in N, Algorithm 2 may perform well in practice.
3. Example `qtwk` is adapted from a nondeterministic quantum program in [35] that models a quantum walk on an undirected graph of four vertices. Our approach succeeds in computing the NTI, which coincides with that in [35], and thus provides a proof of nontermination.

It is also worth highlighting that, as explained in [24], Examples `liu1` and `liu3` cannot be handled by the technique thereof based on under/over-approximating the NTI, since no information about termination can be inferred from the approximations, whereas they are successfully solved by the approach developed in this paper. The same argument applies to other examples like `overview`. A detailed comparison with the approach in [24] in terms of applicability and performance is unfortunately not possible as no publicly available implementation is provided therein.

Table 1
Evaluation results.

Name	Source	d	l	N	Time (sec)	
					Algorithm 1	Algorithm 2
`overview`	[⋆]	2	2	1	0.244	0.012
`liu1`	[24]	2	2	0	0.003	0.002
`liu2`	[24]	2	2	0	0.006	0.002
`loop`	[43]	2	2	2	90.881	0.013
`liu3`	[24]	1	2	2	121.418	0.009
`ineq'`	[⋆]	2	3	2	TO	0.056
`prod`	[39]	3	2	4	TO	0.948
`liu4`	[24]	3	2	4	TO	0.515
`var4`	[⋆]	4	2	5	TO	4.014
`qtwk`	[35]	4	2	1	0.438	0.036

Legends: the first two columns specify the name of the program, and the citation from where the program was adapted ([⋆] marks examples developed by the authors). d indicates the number of variables in the loop, l represents the number of branches in the loop body, and the last two columns give the time (in seconds) taken by the Algorithms. Timeouts (TO) here are set to 20 min. In addition, $\hat{N} = N$ holds for all the examples.

9. Related work

In the literature, various well-established techniques on termination analysis are tailored to linear programs, i.e., programs with linear guards and assignments. For single-path linear programs, Colón and Sipma utilized polyhedral cones to synthesize linear ranking functions [10]. Podelski and Rybalchenko [11], based on Farkas' lemma, presented a complete method to find linear ranking functions if they exist. Ben-Amram and Genaim [9] extended the above results in the following way: first, they proved that synthesizing linear ranking functions for single-path linear programs is in co-NP if program variables are interpreted over the integers $\mathbb{Z}$ (in contrast to the PTIME complexity over the rationals $\mathbb{Q}$ or reals $\mathbb{R}$); second, they proposed an approach to synthesizing *lexicographical ranking functions* for multi-path linear programs. More recently, Kovács and Varonka established in [44] the undecidability of termination for multi-path loops with affine updates and linear inequality conditions.

In recent years, there has been an increasing interest in the termination problem of nonlinear programs due to their omnipresence in safety-critical embedded systems. Bradley et al. [7] proposed an approach to proving termination of MPPs with polynomial behaviours over $\mathbb{R}$ through finite difference trees. A similar idea was exploited in [23] for termination analysis of nonlinear command sequences. Typically, with the development of computer-algebra systems, more and more techniques from symbolic computation, for instance, polynomial ideals [45–47], Gröbner bases [48,49], quantifier elimination [50] and recurrence relations [39,51], have been successfully applied to the verification of programs and control systems. Indeed, these techniques can also be applied to polynomial programs to discover termination or nontermination proofs (cf., e.g., [52] for a class of single-path polynomial loops). Chen et al. [13] proposed a relatively complete method (w.r.t. a given template) for generating polynomial ranking functions over $\mathbb{R}$ by a reduction to semi-algebraic system solving. Gupta et al. [15] proposed a practical method to search for counter-examples of termination, by first generating lasso-shaped [53] candidate paths and then checking the feasibility of the "lassoes" using constraint solving.

For more general programs, many other techniques, like predicate abstraction, parametric abstraction, Lagrangian relaxation, semidefinite programming, etc., have been successfully applied [14,54–56].

The following results are closely related to this paper. Tiwari identified in [5] a class of simple linear loops and proved that its universal termination problem is decidable over $\mathbb{R}$. Thereafter, the results were extended to affine loops over $\mathbb{Z}$. In particular, universal termination of linear loops is decidable over both integers [57] and rationals [8], and further results address loops with at most 4 variables [17] as well as loops with diagonalizable [17,18] or triangular [19] assignment matrices. Xia and Zhang [22] investigated an extension of Tiwari's simple linear loops by allowing nonlinear loop conditions and proved that the termination problem is still decidable over $\mathbb{R}$, yet becomes undecidable over $\mathbb{Z}$. Recently, Frohn et al. [58] proved by a reduction to the *existential theory of the reals* that termination of *triangular weakly nonlinear loops* over $\mathbb{R}$ is decidable, and nontermination over $\mathbb{Z}$ and $\mathbb{Q}$ is semi-decidable. It was further shown in [59] that the halting problem (w.r.t. a given input) is decidable for such loops over any ring that lies between $\mathbb{Z}$ and field $\mathbb{R}_{\mathbb{A}}$ of real algebraic numbers. In [7], Bradley et al. proved by a reduction from Diophantine equations that termination of MPPs with inequalities as loop conditions is not even semi-decidable. Additionally, Müller-Olm and Seidl [6] proved that the termination problem of affine programs with equalities and inequalities is undecidable. This article extends the picture by showing the decidability of termination for a family of MPPs with equality conditions.

Now we place our work in position with existing efforts in addressing Seidenberg's questions concerning the maximal length of Hilbert ascending chains. Most pertinently, Moreno-Socías has proposed in [33] a constructive method for explicitly representing an upper bound on the length of ascending chains of polynomial ideals. Unfortunately, the obtained bound is incorrect, namely, the constructed chain is in fact not the longest. We show that such a spuriously "maximal" length may lead to unsoundness issues when being applied to termination analysis of MPPs. Inspired by Moreno-Socías's results, we present an essentially different approach to

constructing a Hilbert ascending chain of the maximal length. A more technical comparison together with an illustrating example can be found in Section 4.

Moreover, there are results dedicated merely to the complexity of the length of Hilbert ascending chains (Seidenberg's second question). McAloon [60] proved that for a linear control function f, $L(d, f_t)$ is not primitive recursive on t, with f_t defined by $f_t(n) = f(n+t)$. Figueira et al. established in [34], through Dickson's lemma, the relation between the complexity of $L(d, f_t)$ w.r.t. t and that of f in the fast-growing hierarchy. These results yield a partial answer to Seidenberg's questions (i.e., on the non-primitive recursiveness) yet without addressing the question by providing an explicit form of $L(d, f)$. More recently, Steila and Yokoyama [61] presented an alternative proof by using H-closures and exploited termination bounds in terms of reverse mathematics. Compared with existing techniques, our method directly computes the maximal length of the chains rather than providing its lower/upper bounds. Such an explicit expression for $L(d, f)$ enables a straightforward way to show the underlying Ackermannian complexity. More importantly, it delivers insight into how many iterations a loop at most will take until termination, and hence is useful in building a decision procedure for termination. The explicit expression for $L(d, f)$ also fulfils Liu et al.'s aspirations for "a more precise complexity analysis" of the decision algorithms, as stated in [24].

A major difference between our results and the abovementioned ones is that we present a construction of the longest ascending chain and a recursive procedure to compute its length, hence Seidenberg's first question [1] on *explicitness* is completely answered. To the best of our knowledge, this is the first time that such an algorithm is designed for the general case. In contrast, existing work mainly focuses on the complexity problem, and thus only upper bounds (rather than the maximal value) of the length are obtained. Specifically, (1) In [62], the relaxation of the upper bound does not guarantee equality at each step, nor does it provide the length of the longest ascending chain; (2) The method of [34] does not guarantee equality (as noted in Remark 3.1), and consequently, it cannot provide the exact value of the maximal length. Instead, they presented the complexity using the fast-growing hierarchy. Our results on the Ackermannian nature of the upper bound align with theirs but are weaker; (3) [63] proves the complexity by reduction from the coverability problem of reset VAS, which is different from our algorithmic method. It is interesting to compare their method of proving Acermannian complexity $F_\omega(p(n))$ with the method in [34].

A method similar to our Algorithm 2 appears in [64]. The "zero-stable-restrict" algorithm therein computes the *weakest zero-stable transition formula*, which is analogous to the ideal $\langle B \rangle$ in Algorithm 2, for polynomial transition systems. These systems are similar to MPPs but are considerably more expressive. Nonetheless, a key difference lies in the scope of termination guarantees: In [64], the procedure for checking termination is complete only for transition formulas that admit *polynomial ranking functions* (PRFs). That is, if a formula terminates yet admits no PRF, its termination may be missed by their approach – even if non-polynomial ranking functions exist. In contrast, termination of MPPs is proved in full generality by our method. More specifically, our decidability results advance the understanding of the termination problem in two aspects:

1. We proved (in Theorem 6) that any terminating execution of MPP (7) admits a *computable*, *uniform* upper bound $L(d, g, a) - 1$ on its running time. This bound is independent of both the input values and the nondeterministic choices of execution paths. Notably, such uniformity is not established in [64] using the method of PRFs.
2. While [64] considers only polynomials over the field $\mathbb{Q}$ of rational numbers, our results apply to a broader class of (effective) fields $\mathbb{K}$, as discussed in Remark 3-1.

Both advances are derived from our results on computing the maximal length of polynomial ascending chains in Section 4.

10. Conclusion

We showed that the termination problem of a family of multi-path polynomial programs with equality conditions is decidable. This class of (multi-path) polynomial programs, to the best of our knowledge, is hitherto the largest fragment of nonlinear programs with a decidable termination problem: any extension of it by allowing inequalities or inequations in the guards may render the termination problem undecidable. Two variant algorithms for computing the set of non-terminating inputs, and thus deciding termination, are proposed, whose effectiveness was demonstrated on a collection of typical examples from the literature. To analyze complexities of the decision algorithms, we proposed a constructive method to compute the maximal length of Hilbert ascending chains of polynomial ideals under a control function. An explicit recursive function is then obtained for computation, which further implies the Ackermannian complexity of the length, thereby answering the questions raised by Seidenberg. We discussed potential extensions to more general polynomial programs including PGCs and MPPs with inequality guards. A relatively complete approach (in terms of a bound on the degree of polynomial equalities) to synthesize polynomial equality invariants for MPPs was also presented.

For future work, identifying the optimal upper bound on the running time of a terminating MPP remains an open question. Another problem of particular interest is to consider invariant generation without a priori specified degrees. Note that even for the simplest case, namely, with a single variable x (i.e., $d = 1$) and a unique assignment $A(x)$ (i.e., $l = 1$) in the MPP, the problem is non-trivial. In this case, a polynomial equality invariant $G(x) = 0$ exists if and only if the input state x_0 is a pre-periodical point of A. We note that the problem of deciding whether a rational point is pre-periodical for a given polynomial map is related to the Uniform Boundedness Conjecture [65].

CRediT authorship contribution statement

Yangjia Li: Writing – original draft, Methodology, Formal analysis, Conceptualization; **Mingshuai Chen:** Writing – review & editing, Writing – original draft, Validation, Software, Formal analysis; **Liangran Zhao:** Validation, Methodology, Formal analysis;

Naijun Zhan: Writing – review & editing, Supervision, Funding acquisition, Formal analysis; **Hui Lu:** Methodology, Investigation, Formal analysis, Conceptualization; **Guohua Wu:** Validation, Supervision, Formal analysis; **Joost-Pieter Katoen:** Writing – review & editing, Validation, Supervision.

Data availability

A link to the code and experimental data has been shared in the paper.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback. This work has been funded by the Innovation Program for Quantum Science and Technology (No. 2024ZD0300502 and 2024ZD0300503), by the ZJNSF Major Program (No. LD24F020013), by the NSFC (No. 62572427, 62192732, W2511064, and 62502475), by the Zhejiang Key Laboratory Project (No. 2024E10001), by the Fundamental Research Funds for the Central Universities of China (No. 226-2024-00140), by the CAS Project for Young Scientists in Basic Research (No. YSBR-040), and by the MOE Tier 1 grant RG102/23.

References

[1] A. Seidenberg, On the length of a Hilbert ascending chain, Proc. Amer. Math. Soc. 29 (3) (1971) 443–450.
[2] B. Cook, A. Podelski, A. Rybalchenko, Proving program termination, Commun. ACM 54 (5) (2011) 88–98.
[3] L. Yang, C. Zhou, N. Zhan, B. Xia, Recent advances in program verification through computer algebra, Front. Comput. Sci. 4 (1) (2010) 1–16.
[4] A. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc. 43 (2) (1937) 230–265.
[5] A. Tiwari, Termination of linear programs, in: CAV'04, 3114 of *LNCS*, 2004, pp. 70–82.
[6] M. Müller-Olm, H. Seidl, A note on Karr's algorithm, in: ICALP'04, 3142 of *LNCS*, 2004, pp. 1016–1028.
[7] A. Bradley, Z. Manna, H. Sipma, Termination of polynomial programs, in: VMCAI'05, 3385 of *LNCS*, 2005, pp. 113–129.
[8] M. Braverman, Termination of integer linear programs, in: CAV'06, 4144 of *LNCS*, 2006, pp. 372–385.
[9] A. Ben-Amram, S. Genaim, Ranking functions for linear-constraint loops, J. ACM 61 (4) (2014) 26:1–26:55.
[10] M. Colón, H. Sipma, Synthesis of linear ranking functions, in: TACAS'01, 2031 of *LNCS*, 2001, pp. 67–81.
[11] A. Podelski, A. Rybalchenko, A complete method for the synthesis of linear ranking functions, in: VMCAI'04, 2937 of *LNCS*, 2004, pp. 239–251.
[12] A. Podelski, A. Rybalchenko, Transition invariants, in: LICS'04, 2004, pp. 32–41.
[13] Y. Chen, B. Xia, L. Yang, N. Zhan, C. Zhou, Discovering non-linear ranking functions by solving semi-algebraic systems, in: ICTAC'07, 4711 of *LNCS*, 2007, pp. 34–49.
[14] P. Cousot, Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming, in: VMCAI'05, 3385 of *LNCS*, 2005, pp. 1–24.
[15] A. Gupta, T. Henzinger, R. Majumdar, A. Rybalchenko, R. Xu, Proving non-termination, in: POPL'08, 2008, pp. 147–158.
[16] W. Harris, A. Lal, A. Nori, S. Rajamani, Alternation for termination, in: SAS'10, 6337 of *LNCS*, 2010, pp. 304–319.
[17] J. Ouaknine, J.S. Pinto, J. Worrell, On termination of integer linear loops, in: SODA'15, 2015, pp. 957–969.
[18] M. Bozga, R. Iosif, F. Konecný, Deciding conditional termination, Log. Methods Comput. Sci. 10 (3) (2014) 1–61.
[19] F. Frohn, J. Giesl, Termination of triangular integer loops is decidable, in: CAV'19, 11562 of *LNCS*, 2019, pp. 426–444.
[20] R. Rebiha, N. Matringe, A.V. Moura, Generating asymptotically non-terminant initial variable values for linear diagonalizable programs, in: SCSS'13, 2013, pp. 81–92.
[21] B. Xia, L. Yang, N. Zhan, Z. Zhang, Symbolic decision procedure for termination of linear programs, Form. Asp. Comput. 23 (2) (2011) 171–190.
[22] B. Xia, Z. Zhang, Termination of linear programs with nonlinear constraints, J. Symb. Comput. 45 (11) (2010) 1234–1249.
[23] D. Babić, B. Cook, A. Hu, Z. Rakamarić, Proving termination of nonlinear command sequences, Form. Asp. Comp. 25 (3) (2013) 389–403.
[24] J. Liu, M. Xu, N. Zhan, H. Zhao, Discovering non-terminating inputs for multi-path polynomial programs, J. Sys. Sci. Compl. 27 (6) (2014) 1286–1304.
[25] J. Ouaknine, J. Worrell, On linear recurrence sequences and loop termination, ACM SIGLOG News 2 (2) (2015) 4–13.
[26] M.F. Atiyah, I.G. MacDonald, Introduction to Commutative Algebra, Addison-Wesley Series in Mathematics, Addison-Wesley, 1969.
[27] A. Seidenberg, Constructive proof of Hilbert's theorem on ascending chains, Trans. Amer. Math. Soc. 174 (1972) 305–312.
[28] B. Buchberger, Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einum nulldimensionalen Polynomideal, Ph.D. thesis, University of Innsbruck, 1965.
[29] E. Mayr, Membership in polynomial ideals over $\mathbb{Q}$ is exponential space complete, in: STACS'89, 349 of *LNCS*, 1989, pp. 400–406.
[30] E.W. Mayr, Some complexity results for polynomial ideals, J. Complex. 13 (1997) 303–325.
[31] A. Tarski, A Decision Method for Elementary Algebra and Geometry, University of California Press, Berkeley, 1951.
[32] F. Macaulay, Some properties of enumeration in the theory of modular systems, Proc. London Math. Soc. s2-26 (1926) 531–555.
[33] G. Moreno-Socías, Length of polynomial ascending chains and primitive recursiveness, Math. Scand. 71 (1992) 181–205.
[34] D. Figueira, S. Figueira, S. Schmitz, P. Schnoebelen, Ackermannian and primitive-recursive bounds with Dickson's lemma, in: LICS'11, 2011, pp. 269–278.
[35] Y. Li, N. Yu, M. Ying, Termination of nondeterministic quantum programs, Acta Inf. 51 (1) (2014) 1–24.
[36] D. König, Über eine Schlussweise aus dem Endlichen ins Unendliche, Acta Sci. Math. 3 (2–3) (1927) 121–130.
[37] M. Davis, Hilbert's tenth problem is unsolvable, Am. Math. Mon. 80 (3) (1973) 233–269.
[38] E. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs, Commun. ACM 18 (8) (1975) 453–457.
[39] E. Rodríguez-Carbonell, D. Kapur, Generating all polynomial invariants in simple loops, J. Symb. Comp. 42 (4) (2007) 443–476.
[40] E. Hrushovski, J. Ouaknine, A. Pouly, J. Worrell, On strongest algebraic program invariants, J. ACM 70 (5) (2023) 1–22.
[41] E. Rodríguez-Carbonell, D. Kapur, Automatic generation of polynomial invariants of bounded degree using abstract interpretation, Sci. Comput. Program. 64 (1) (2007) 54–75.
[42] M. Colón, Approximating the algebraic relational semantics of imperative programs, in: Proceedings of the 11th International Static Analysis Symposium, 3148 of *Lecture Notes in Computer Science*, Springer International Publishing, 2004, pp. 296–311.
[43] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: POPL'77, 1977, pp. 238–252.

[44] L. Kovács, A. Varonka, What else is undecidable about loops?, in: RAMiCS'23, 13896 of *LNCS*, 2023, pp. 176–193.
[45] R. Rebiha, A.V. Moura, N. Matringe, Generating invariants for non-linear loops by linear algebraic methods, Formal Asp. Comput. 27 (5–6) (2015) 805–829.
[46] E. Sontag, Y. Rouchaleau, On discrete-time polynomial systems, Nonlinear Anal. Theory Methods Appl. 1 (1) (1976) 55–64.
[47] J. Liu, N. Zhan, H. Zhao, Computing semi-algebraic invariants for polynomial dynamical systems, in: EMSOFT'11, 2011, pp. 97–106.
[48] S. Sankaranarayanan, H. Sipma, Z. Manna, Non-linear loop invariant generation using Gröbner bases, in: POPL'04, 2004, pp. 318–329.
[49] M. Müller-Olm, H. Seidl, Computing polynomial program invariants, Inf. Proc. Lett. 91 (5) (2004) 233–244.
[50] D. Kapur, A quantifier-elimination based heuristic for automatically generating inductive assertions for programs, J. Sys. Sci. Compl. 19 (3) (2006) 307–330.
[51] L. Kovács, Reasoning algebraically about P-solvable loops, in: TACAS'08, 4963 of *LNCS*, 2008, pp. 249–264.
[52] Y. Li, Termination of single-path polynomial loop programs, in: ICTAC'16, 9965 of *LNCS*, 2016, pp. 33–50.
[53] B. Cook, A. Podelski, A. Rybalchenko, Termination proofs for systems code, in: PLDI'06, 2006, pp. 415–426.
[54] P. Cousot, R. Cousot, An abstract interpretation framework for termination, in: POPL'12, 2012, pp. 245–258.
[55] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, M. Sagiv, Proving conditional termination, in: CAV'08, 5123 of *LNCS*, 2008, pp. 328–340.
[56] H. Wu, Q. Wang, B. Xue, N. Zhan, L. Zhi, Z.-H. Yang, Synthesizing invariants for polynomial programs by semidefinite programming, ACM Trans. Program. Lang. Syst. 47 (2024) 1–35.
[57] M. Hosseini, J. Ouaknine, J. Worrell, Termination of linear loops over the integers, in: 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 118:1–118:13.
[58] F. Frohn, M. Hark, J. Giesl, On the decidability of termination for polynomial loops, CoRR abs/1910.11588 (2019) 1–60.
[59] M. Hark, F. Frohn, J. Giesl, Polynomial loops: beyond termination, in: LPAR'20, 73 of *EPiC Series in Computing*, 2020, pp. 279–297.
[60] K. McAloon, Petri nets and large finite sets, Theor. Comput. Sci. 32 (1984) 173–183.
[61] S. Steila, K. Yokoyama, Reverse mathematical bounds for the termination theorem, Ann. Pure Appl. Logic 167 (12) (2016) 1213–1241.
[62] G. PASTUSZAK, Ascending chains of ideals in the polynomial ring, Turk. J. Math. 44 (6) (2020) 2402-2414.
[63] M. Benedikt, T. Duff, A. Sharad, J. Worrell, Polynomial automata: zeroness and applications, in: 2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), IEEE, 2017, p. 1-12.
[64] S. Zhu, Z. Kincaid, Breaking the mold: nonlinear ranking function synthesis without templates, in: International Conference on Computer Aided Verification, Springer, 2024, pp. 431–452.
[65] P. Morton, J. Silverman, Rational periodic points of rational functions, Inter. Math. Res. Notices 2 (1994) 97–110.