

DISCOVERING NON-TERMINATING INPUTS FOR MULTI-PATH POLYNOMIAL PROGRAMS*

LIU Jiang · XU Ming · ZHAN Naijun · ZHAO Hengjun

DOI: 10.1007/s11424-014-2145-6

Received: 10 July 2012 / Revised: 18 February 2014

©The Editorial Office of JSSC & Springer-Verlag Berlin Heidelberg 2014

Abstract This paper investigates the termination problems of multi-path polynomial programs (MPPs) with equational loop guards. To establish sufficient conditions for termination and nontermination simultaneously, the authors propose the notion of strong/weak non-termination which under/over-approximates non-termination. Based on polynomial ideal theory, the authors show that the set of all strong non-terminating inputs (SNTI) and weak non-terminating inputs (WNTI) both correspond to the real varieties of certain polynomial ideals. Furthermore, the authors prove that the variety of SNTI is computable, and under some sufficient conditions the variety of WNTI is also computable. Then by checking the computed SNTI and WNTI varieties in parallel, termination properties of a considered MPP can be asserted. As a consequence, the authors establish a new framework for termination analysis of MPPs.

Keywords Polynomial ideals, polynomial programs, termination analysis.

LIU Jiang

Chongqing Key Laboratory of Automated Reasoning and Cognition, CIGIT, Chinese Academy of Sciences, Chongqing 400714, China. Email: liujiang@cigit.ac.cn.

XU Ming

Department of Computer Science and Technology, East China Normal University, Shanghai 200241, China. Email: mxu@cs.ecnu.edu.cn.

ZHAN Naijun

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China. Email: znj@ios.ac.cn.

ZHAO Hengjun (Corresponding author)

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China; Chongqing Key Laboratory of Automated Reasoning and Cognition, CIGIT, Chinese Academy of Sciences, Chongqing 400714, China. Email: zhaohj@ios.ac.cn.

*This research was supported by the National Basic Research Program of China under Grant No. 2014CB340700, the National Science and Technology Major Project of China under Grant No. 2012ZX01039-004, the National Natural Science Foundation of China under Grant Nos. 91118007, 11071273, 61202131, 11401218, cstc2012ggB40004, cstc2013jjys40001, SRFDP under Grant No. 20130076120010, the Open Project of Shanghai Key Laboratory of Trustworthy Computing under Grant No. 07dz22304201307, and West Light Foundation of Chinese Academy of Sciences.

◊ *This paper was recommended for publication by Editor LI Ziming.*

1 Introduction

Termination analysis plays an important role in program verification and testing, and has attracted an increasing attention recently^[1, 2]. However, the decision problem of program termination is equivalent to the famous halting problem^[3], and hence is undecidable. Thus, a complete method for termination analysis for programs, even for the general linear or polynomial program, is impossible^[4–6]. To achieve positive results, a practical way analogous to [7] is to establish sufficient conditions for termination and nontermination simultaneously for a considered class of programs, and then check these conditions in parallel. We adopt this strategy of termination analysis in this work.

In this paper, we use multi-path polynomial programs (MPPs) as program models for termination analysis. The MPP model proposed in [5] is an expressive class of loops with multiple paths, polynomial loop guards and assignments, which enable practical code abstraction and analysis. It was proved in [5] that the termination problem of MPPs is generally undecidable. To get some computational results, we restrict to MPPs in the context whose loop guards are polynomial equations.

The semantics of an MPP with ℓ paths can be explained as follows. Given an input $\mathbf{x} \in \mathbb{R}^n$, if at the first iteration \mathbf{x} satisfies the loop guard, then one of the multiple paths in the loop body will be nondeterministically selected and the corresponding assignment will be used to update the value of \mathbf{x} , which results in ℓ possible values of \mathbf{x} ; at each following iteration, we repeat the same kind of loop guard test and nondeterministic assignment for each of the possible values of \mathbf{x} obtained so far. Thus, the execution of an MPP on input $\mathbf{x} \in \mathbb{R}^n$ produces a tree structure of execution paths, which is caused by the multi-paths in the considered program model.

An input \mathbf{x} is called a non-terminating input if the execution tree on \mathbf{x} has an infinite path. Given an MPP, we aim to compute an explicit representation for the set of all non-terminating inputs (NTI), or certain kinds of approximations of NTI. The theory of polynomial ideals serves as a critical tool for developing our approach.

The basic idea can be explained using a very simple example:

$$\text{while } (G(x) = 0) \quad \{x := p(x)\}.$$

The above MPP is composed of a single program variable x , the loop guard $G(x) = 0$ and the polynomial assignment $p(x)$. If the MPP is non-terminating upon an input $x \in \mathbb{R}$, then $x, p(x), p(p(x)), \dots$ must persistently satisfy $G(x) = 0$, i.e., $G(x) = G(p(x)) = G(p(p(x))) = \dots = 0$. Thus, any non-terminating input is a common real root of the set of polynomials $\mathcal{G} \triangleq \{G(x), G(p(x)), G(p(p(x))), \dots\}$, or equivalently, an element in the real variety of the ideal \mathcal{I} generated by \mathcal{G} . By a well-known result in polynomial ideal theory, \mathcal{I} is generated by a finite number of elements in \mathcal{I} . If this finite set of generators of \mathcal{I} can be computed, their common real roots comprise NTI.

For general MPPs with more than one paths, the NTI is not directly connected to the real variety of a polynomial ideal. Instead, we propose the notions of strong and weak non-termination, such that the set of all strong non-terminating inputs (SNTI) is a subset of NTI,

and the set of all weak non-terminating inputs (WNTI) is a superset of NTI. Therefore the non-emptiness of SNTI is a sufficient condition for non-termination, and the emptiness of WNTI is a sufficient condition for termination. Furthermore, both SNTI and WNTI are shown to be real varieties of certain polynomial ideals, of which a finite set of generators can be computed (at least under some sufficient conditions). Thus for any MPP, its SNTI and WNTI can be investigated simultaneously for a better result of termination analysis. The proposed approach will be illustrated by some examples.

In the past, most well-established work on termination analysis can only be applied to linear programs, whose guards and assignments are linear. A classical method for establishing termination of a program, either linear or polynomial, makes use of a well-founded domain together with the so-called ranking function that maps the state space of the program to the domain. For single-path linear programs, Colón and Sipma^[8] utilized polyhedral cones to synthesize linear ranking functions. Podelski and Rybalchenko^[9], based on Farkas' lemma, presented a complete method to find linear ranking functions if they exist. These methods pay more attention to the search for ranking functions than to the inherent structure of loops, while Tiwari^[4] first noticed that the termination of a class of simple linear loops is closely related to the eigenvalues of assignment matrix, and proved that the termination problem of linear programs is decidable over \mathbb{R} . This theory was further developed by Braverman^[6] and Xia, et al.^[10, 11]. Xu, et al.^[12, 13], also developed Tiwari's work by constructing non-terminating witnesses.

It was shown in [5] that the termination of MPPs is undecidable by reduction from Diophantine equations. However, effective methods that are incomplete or relatively complete for analyzing termination still exist. Bradley, et al^[5]. proposed an approach to proving termination over \mathbb{R} through finite difference trees. Cook, et al.^[14] devised an algorithm to under-approximate the weakest preconditions for termination using decidable theories. Typically, with the development of computer algebra, more and more techniques from symbolic computation, for instance, Gröbner basis^[15, 16], quantifier elimination^[17] and recurrence relation^[18, 19], are borrowed and successfully applied to the verification of programs. Certainly, these techniques can also be applied to polynomial programs to discover termination or non-termination proofs. Chen, et al.^[20] proposed a relatively complete (w.r.t. a given template) method for generating polynomial ranking functions over \mathbb{R} by reduction to semi-algebraic system solving. On the other hand, Gupta, et al. proposed a practical method to search for counter-examples of termination^[21], by first generating lasso-shaped^[22] candidate paths and then checking the feasibility of the "lassoes" using constraint solving. Velroyen and Rümmer^[23] applied invariants to show that terminating states of a program are unreachable from certain initial states, and then identified these "bad" initial states by constraint-solving techniques. Brockschmidt, et al.^[24] detected non-termination and Null Pointer Exceptions for Java Bytecode by constructing and analyzing termination graphs, and implemented a termination prover AProVE.

For more general programs, many other techniques, like predicate abstraction, parametric abstraction, fair assumption, Lagrangian relaxation, semidefinite programming, sum of squares and curve fitting^[25, 26], and so on, have been successfully applied.

The remainder of this paper is organized as follows. In Section 2, some concepts and results on polynomial ideals are reviewed. In Section 3, we first introduce MPPs and then define the notions of non-termination and strong/weak non-termination. In Sections 4 and 5 we show how to compute the set of all strong and weak non-terminating inputs of MPPs, respectively. Finally, we draw a conclusion in Section 6.

2 Preliminaries

In this section, we briefly recall some basic concepts and results on computational algebraic geometry, which serve as the theoretical tool for dealing with non-terminating inputs. For a detailed exposition to this subject, please refer to [27]. Throughout this paper, we use $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ to denote the set of natural, rational, and real numbers, respectively.

Definition 2.1 (Ideal) Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a vector of variables. A subset \mathcal{I} of the polynomial ring $\mathbb{Q}[\mathbf{x}]$ is an ideal if it satisfies:

- (i) $0 \in \mathcal{I}$;
- (ii) if $p, q \in \mathcal{I}$, then $p + q \in \mathcal{I}$;
- (iii) if $p \in \mathcal{I}$ and $r \in \mathbb{Q}[\mathbf{x}]$, then $rp \in \mathcal{I}$.

Definition 2.2 (Variety) Given a field \mathbb{K} extending \mathbb{Q} , the variety defined by $P \subseteq \mathbb{Q}[\mathbf{x}]$ is $\mathcal{V}(P) = \{\mathbf{x} \in \mathbb{K}^n \mid \forall p \in P : p(\mathbf{x}) = 0\}$. The real variety of P is the intersection of $\mathcal{V}(P)$ and \mathbb{R}^n , i.e., $\mathcal{V}(P) \cap \mathbb{R}^n$.

Let P be a nonempty subset of $\mathbb{Q}[\mathbf{x}]$. It is easy to verify that

$$\left\{ \sum_{i=1}^k r_i p_i \mid k \in \mathbb{N}, r_i \in \mathbb{Q}[\mathbf{x}], p_i \in P \right\}$$

is an ideal, denoted by $\langle P \rangle$, and P is called a set of generators, or a basis of $\langle P \rangle$. In particular, if P is a finite set $\{p_1, p_2, \dots, p_k\}$, then $\langle P \rangle$ is written as $\langle p_1, p_2, \dots, p_k \rangle$. An ideal \mathcal{I} is called principal if it can be generated by a single element, i.e., $\mathcal{I} = \langle p \rangle$ for some $p \in \mathbb{Q}[\mathbf{x}]$. It is easy to verify that $\mathcal{V}(P) = \mathcal{V}(\langle P \rangle)$ for any $P \subseteq \mathbb{Q}[\mathbf{x}]$.

Theorem 2.3 (Hilbert's basis theorem) *Every ideal $\mathcal{I} \subseteq \mathbb{Q}[\mathbf{x}]$ has a finite number of generators, i.e., $\mathcal{I} = \langle p_1, p_2, \dots, p_k \rangle$ for some $p_1, p_2, \dots, p_k \in \mathcal{I}$.*

In particular, for any $\mathcal{I} = \langle p_1, p_2, \dots, p_k \rangle$, a unique (reduced) Gröbner basis of \mathcal{I} can be computed using Buchberger's algorithm, under a fixed monomial ordering. Using Gröbner basis, the problems of deciding membership of an element in an ideal, testing equality or inclusion of ideals, and so on, can be effectively solved; a Gröbner basis is also helpful in checking whether a given ideal is principal.

Given an ideal $\mathcal{I} = \langle p_1, p_2, \dots, p_k \rangle$, by applying the so-called real-solution-isolation algorithms^[28, 29] to the system of polynomial equations $p_i(\mathbf{x}) = 0$ for $1 \leq i \leq k$, the emptiness of the real variety $\mathcal{V}(\mathcal{I}) \cap \mathbb{R}^n$ can be checked; and if it is nonempty and is zero-dimensional, the

finite set of points in $\mathcal{V}(\mathcal{I}) \cap \mathbb{R}^n$ can be isolated using arbitrarily small cubes (with rational endpoints).

All the above mentioned algorithms have been well-implemented in the computer algebra system Maple.

The following result can be derived from Hilbert’s Basis Theorem.

Theorem 2.4 (The Ascending Chain Condition) *Let $\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \mathcal{I}_3 \subseteq \dots$ be an ascending chain of ideals in $\mathbb{Q}[\mathbf{x}]$. Then there exists an integer N such that $\mathcal{I}_N = \mathcal{I}_{N+1} = \mathcal{I}_{N+2} = \dots$.[†]*

3 Multi-Path Polynomial Programs and Non-Termination

3.1 Multi-Path Polynomial Programs

Definition 3.1 (MPP with equational loop guard) A multi-path polynomial program with equational loop guard and ℓ paths is of the form

$$\text{while } (G(\mathbf{x}) = 0) \left\{ \begin{array}{l} \mathbf{x} := \mathbf{A}_1(\mathbf{x}); \\ \parallel \mathbf{x} := \mathbf{A}_2(\mathbf{x}); \\ \vdots \\ \parallel \mathbf{x} := \mathbf{A}_{\ell-1}(\mathbf{x}); \\ \parallel \mathbf{x} := \mathbf{A}_\ell(\mathbf{x}); \end{array} \right\}, \tag{1}$$

where

- $\mathbf{x} \in \mathbb{R}^n$ denotes a vector of program variables;
- $G(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ is a polynomial and $G(\mathbf{x}) = 0$ is the equational loop guard;
- “ \parallel ” interprets an exclusive choice of the ℓ paths;
- $\mathbf{A}_i \in \mathbb{Q}^n[\mathbf{x}]$ for $1 \leq i \leq \ell$ corresponding to each path is a vector of polynomials describing simultaneous assignments for all program variables.

Note that

- 1) MPP (1) allows arbitrary Boolean combinations of polynomial equations of the form $\bigvee_{i=1}^M \bigwedge_{j=1}^{N_i} G_{ij}(\mathbf{x}) = 0$ as loop guard, and assuming $G(\mathbf{x}) = 0$ would not lose any generality, since

$$\left[\bigvee_{i=1}^M \bigwedge_{j=1}^{N_i} G_{ij}(\mathbf{x}) = 0 \right] \iff \left[\prod_{i=1}^M \sum_{j=1}^{N_i} [G_{ij}(\mathbf{x})]^2 = 0 \right]. \tag{2}$$

- 2) Initial conditions on \mathbf{x} are not specified, since we are aiming to compute the set of all non-terminating inputs (as real varieties) for a given MPP; nevertheless, equational initial conditions on \mathbf{x} can be easily taken into account by manipulation of corresponding ideals.

[†]Here \mathcal{I}_N is called the fixed point of this chain.

Example 3.2 Consider the following MPP

$$\text{while } (x - z^3 = 0 \vee y - z^2 = 0) \left\{ \begin{array}{l} (x, y, z) := (x, y + 2z + 1, z + 1); \\ \parallel \\ (x, y, z) := (x - 3y + 3z - 1, y + 2z - 1, z - 1); \end{array} \right\}.$$

According to (2), the loop guard $y - z^2 = 0 \vee x - z^3 = 0$ can be transformed to $(x - z^3)^2 \times (y - z^2)^2 = 0$, or equivalently $(x - z^3) \times (y - z^2) = 0$. There are two exclusive simultaneous assignments, i.e., $(x, y, z) := (x, y + 2z + 1, z + 1)$ and $(x, y, z) := (x - 3y + 3z - 1, y + 2z - 1, z - 1)$, which can be named by A_1 and A_2 in the manner of MPP (1). ■

3.2 The Execution of MPPs

We associate with each A_i in MPP (1) a different symbol a_i indexed by i for $1 \leq i \leq \ell$. Then the execution of MPP (1) on input $x \in \mathbb{R}^n$ can be graphically represented as a labeled execution tree illustrated by Figure 1. The nodes in the execution tree are elements in \mathbb{R}^n . A directed edge labeled by a_i stands for the execution of assignment A_i , and points to the updated value given by the execution of A_i .

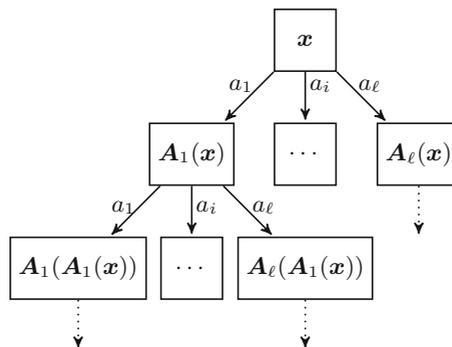


Figure 1 The labeled execution tree of MPP (1) on input x

Definition 3.3 (Execution tree) The execution tree is defined inductively as:

- the root is x ;
- for any node x' in the tree, it is a leaf node if $G(x') \neq 0$; otherwise x' has ℓ children $A_1(x'), A_2(x'), \dots, A_\ell(x')$, and there is a directed edge connecting x' and $A_i(x')$, labeled by a_i , for any $1 \leq i \leq \ell$.

To characterize the set of paths in Figure 1 starting from root x , we introduce below some conventional notions on alphabet and strings for self-containedness.

Let Σ be a finite set of symbols, called an alphabet. A finite or infinite string over Σ can be denoted respectively by $\tau = a_1 a_2 \dots a_s$ or $\tau = a_1 a_2 \dots a_s a_{s+1} \dots$, where $a_i \in \Sigma$ for $i = 1, 2, \dots$. A string without any symbol is called the empty string, denoted by ϵ . We identify strings with one symbol as the symbol itself. By convention, the set of all finite (including ϵ) and infinite

strings are denoted by Σ^* and Σ^ω , respectively. Let $|\tau|$ denote the length of a string τ . Thus $|\varepsilon| = 0$, $|\tau| = s$ if $\tau = a_1 a_2 \cdots a_s$, and $|\tau| = \infty$ if $\tau \in \Sigma^\omega$. Given $\tau \in \Sigma^* \cup \Sigma^\omega$, if there exist $\tau_1 \in \Sigma^*$ and $\tau_2 \in \Sigma^* \cup \Sigma^\omega$ such that $\tau = \tau_1 \cdot \tau_2$, where \cdot is the concatenation operator on strings, then τ_1 is called a prefix of τ ; if in addition $|\tau_1| = 1$, then τ_1 and τ_2 are called the head and tail of τ , denoted by $\text{head}(\tau)$ and $\text{tail}(\tau)$ respectively. The set of all prefixes of τ is denoted by $\mathbf{Pre}(\tau)$; if $\tau' \in \mathbf{Pre}(\tau)$ and $\tau' \neq \tau$, then τ' is called a proper prefix of τ .

Let Σ be the alphabet consists of all the labels in Figure 1, i.e., $\Sigma = \{a_1, a_2, \dots, a_\ell\}$. Then for any path in Figure 1 from root \mathbf{x} to some node \mathbf{x}' , the labels of the edges along this path generate a finite string τ over Σ ; furthermore, \mathbf{x}' results from \mathbf{x} by the iterated application of the assignments corresponding to the symbols appearing in τ .

Actually, if we define an induced function $\mathcal{A}_\tau : \mathbb{R}^n \rightarrow \mathbb{R}^n$ for any $\tau \in \Sigma^*$ as:

- $\mathcal{A}_\varepsilon = \mathbf{id}$, i.e., the identity mapping;
- $\mathcal{A}_{a_i} = \mathbf{A}_i$ for any $a_i \in \Sigma$, where \mathbf{A}_i is the i -th assignment in MPP (1);
- $\mathcal{A}_\tau = \mathcal{A}_{\text{tail}(\tau)} \circ \mathcal{A}_{\text{head}(\tau)}$ for τ with $|\tau| \geq 2$, where \circ denotes the composition of functions, i.e., $(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$ for $f, g : \mathbb{R}^n \rightarrow \mathbb{R}^n$,

then the set of paths starting from root \mathbf{x} of the labeled tree in Figure 1 can be represented by a set of strings over Σ , using the following definition.

Definition 3.4 (Execution path) Let Σ be what specified above for MPP (1). Then $\tau \in \Sigma^* \cup \Sigma^\omega$ is an execution path of MPP (1) on input $\mathbf{x} \in \mathbb{R}^n$ if

- $\tau = \varepsilon$; or
- $G(\mathcal{A}_{\tau'}(\mathbf{x})) = 0$ for all $\tau' \in \mathbf{Pre}(\tau), \tau' \neq \tau$ (i.e., τ' is a proper prefix of τ).

We denote the set of all execution paths of MPP (1) on input \mathbf{x} by $\mathbf{Path}(\mathbf{x})$. Then there is a one-to-one correspondence between $\mathbf{Path}(\mathbf{x})$ and the set of paths from root \mathbf{x} in Figure 1: For any execution path $\tau = a_1 a_2 \cdots a_s \cdots$ (finite or infinite), there is a path in Figure 1 from \mathbf{x} to a node $\mathcal{A}_\tau(\mathbf{x})$, labeled by the symbols $a_1, a_2, \dots, a_s, \dots$; conversely, for any path from \mathbf{x} to some node \mathbf{x}' in Figure 1, the juxtaposition of the labels along this path gives an execution path in $\mathbf{Path}(\mathbf{x})$.

Example 3.5 Consider the following MPP

$$\text{while } (x + y = 0) \left\{ \begin{array}{l} (x, y) := (x + 1, y - 1); \\ \parallel \\ (x, y) := (x^2, y^2); \end{array} \right\}. \tag{3}$$

If we associate symbols L, R with the first and second assignments of MPP (3) respectively, it is easy to verify that on input $(1, -1)$, MPP (3) has an infinite execution path $LLL \cdots$ and finite execution paths R, LR, LLR , and so on (see Figure 2).

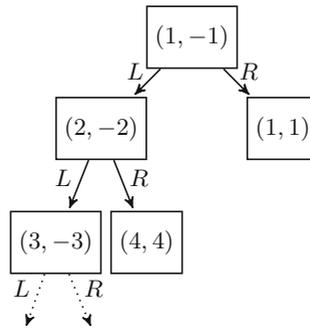


Figure 2 The execution tree of MPP (3) on input $(1, -1)$

3.3 Non-Termination Definitions

Definition 3.6 (Non-termination) An input $\mathbf{x} \in \mathbb{R}^n$ of MPP (1) is a non-terminating input if the MPP has an infinite execution path on \mathbf{x} , i.e., $\mathbf{Path}(\mathbf{x}) \cap \Sigma^\omega \neq \emptyset$; otherwise \mathbf{x} is a terminating input.

Example 3.7 Consider MPP (3). By Example 3.5, MPP (3) has an infinite execution path $LLL \dots$ on $(1, -1)$. So $(1, -1)$ is a non-terminating input of MPP (3).

The set of all non-terminating inputs of MPP (1) is denoted by **NTI**. Termination analysis of MPPs concerns such problems as

- 1) whether a given MPP terminates on all inputs \mathbf{x} ; or alternatively,
- 2) detection of non-terminating inputs for a given MPP.

It was proved in [5] that for general MPPs with inequality-typed loop guards, the problem of deciding whether **NTI** is an empty set is undecidable. To get some computational results for termination analysis of MPP (1), we propose the following two notions for under/over-approximating non-terminating input set.

Definition 3.8 (Strong non-termination) An input $\mathbf{x} \in \mathbb{R}^n$ of MPP (1) is a strong non-terminating input if the MPP has all the infinite strings as its execution paths, i.e., $\Sigma^\omega \subseteq \mathbf{Path}(\mathbf{x})$.

Intuitively, on a strong nonterminating input, the loop guard of MPP (1) is persistently satisfied no matter how many iterations have been executed, or which assignment is selected at each iteration.

Example 3.9 Consider the following MPP

$$\text{while } (x^2 + 1 - y = 0) \left\{ \begin{array}{l} (x, y) := (x, x^2 y); \\ \parallel \\ (x, y) := (-x, y); \end{array} \right\}. \quad (4)$$

We associate symbols L, R with the two assignments in MPP (4) respectively. Figure 3 illustrates the execution tree of MPP (4) on input $(1, 2)$. Note that both $(1, 2)$ and $(-1, 2)$ satisfy the loop guard. So it can be shown that $(1, 2)$ is a strong non-terminating input of MPP (4).

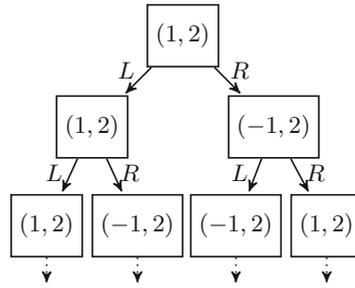


Figure 3 The execution tree of MPP (4) on input (1, 2)

Definition 3.10 (Weak non-termination) An input $\mathbf{x} \in \mathbb{R}^n$ of MPP (1) is a weak non-terminating input if for any $k \in \mathbb{N}$, there exists $\tau_k \in \Sigma^*$ such that

$$|\tau_k| = k \text{ and } G(\mathcal{A}_{\tau_k}(\mathbf{x})) = 0. \tag{5}$$

Intuitively, on a weak nonterminating input \mathbf{x} , we can find τ_k of any length $k \in \mathbb{N}$, such that when applying the induced function \mathcal{A}_{τ_k} to \mathbf{x} , the loop guard will be satisfied. However, as the set of such τ_k is not prefix-closed, according to Definition 3.4, τ_k is not necessarily an execution path for any k . Therefore we call such τ_k pseudo execution paths of MPP (1).

Example 3.11 Consider the following MPP

$$\text{while } ((x - 1)(x - 2) = 0) \left\{ \begin{array}{l} x := 1 - x^2; \\ \parallel \\ x := x + 1; \end{array} \right\}. \tag{6}$$

We associate symbols L, R with the first and second assignments, respectively. Note that the loop guard is satisfied if and only if $x = 1$ or $x = 2$. Then it is easy to verify that $x = 1$ is a weak non-terminating input of MPP (6), by choosing τ_k as $\varepsilon, R, LL, LLR, LLLL, \dots$ (see Figure 4). Note that $x = 1$ is actually a terminating input of MPP (6) and the dotted nodes in Figure 4 cannot be reached by real execution.

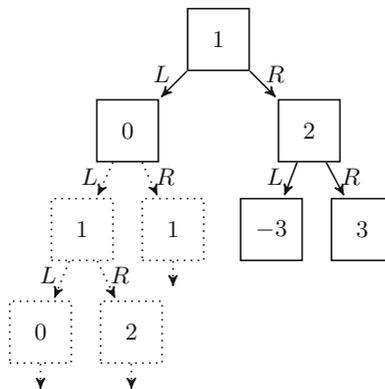


Figure 4 The (pseudo) execution paths of MPP (6) on input $x = 1$

Denote the set of all strong non-terminating inputs and weak non-terminating inputs by **SNTI** and **WNTI**, respectively. Then the relationship between the above notions of non-termination can be stated as the following proposition.

Proposition 3.12 *For MPP (1), the inclusions $\mathbf{SNTI} \subseteq \mathbf{NTI} \subseteq \mathbf{WNTI}$ hold.*

Proof The first inclusion follows from $\Sigma^\omega \subseteq \mathbf{Path}(\mathbf{x}) \implies \Sigma^\omega \cap \mathbf{Path}(\mathbf{x}) \neq \emptyset$. Second, for any $\mathbf{x} \in \mathbf{NTI}$, there exists $\tau \in \Sigma^\omega$ satisfying the condition of Definition 3.4. Then by choosing τ_k as the prefix of τ with length k , we can see from (5) that $\mathbf{x} \in \mathbf{WNTI}$. Thus \mathbf{NTI} is a subset of \mathbf{WNTI} . \blacksquare

Remark 3.13 Both inclusions in Proposition 3.12 may be proper: from Examples 3.5 and 3.7 we can see that $(1, -1) \in \mathbf{NTI}$, but $(1, -1) \notin \mathbf{SNTI}$ for MPP (3); Example 3.11 shows that $1 \in \mathbf{WNTI}$, but $1 \notin \mathbf{NTI}$ for MPP (6). However, if MPP (1) has a single path, we can conclude that $\mathbf{SNTI} = \mathbf{NTI} = \mathbf{WNTI}$.

According to Proposition 3.12, strong/weak non-termination can be used in under/over approximation analysis of non-termination: any element in **SNTI** is a witness to the non-termination of MPP (1), and conversely the emptiness of **WNTI** implies that MPP (1) terminates on all inputs.

4 The Computation of SNTI

In this section, based on polynomial ideal theory, we will reduce **SNTI** of any MPP (1) to the real variety of a polynomial ideal \mathcal{I} ; furthermore, we can always compute a finite basis of \mathcal{I} , thus giving a complete characterization of **SNTI**. This enables us to check the emptiness of **SNTI** for any MPP (1), or test whether $\mathbf{x} \in \mathbf{SNTI}$ for a given $\mathbf{x} \in \mathbb{R}^n$.

4.1 Iteration Trees

During the execution of MPPs, the assignments \mathbf{A}_i in the loop body are applied to program states \mathbf{x} . On the other hand, if we focus on the loop guard test of each iteration, the assignments can also be viewed as applications to the loop guard $G(\mathbf{x}) = 0$, which produce a series of polynomial equations, or equivalently, polynomials. Using these polynomials as nodes, we can construct an iteration tree (see Figure 5) which is very useful in the computation of non-terminating inputs.

Definition 4.1 (Iteration tree) An iteration tree of MPP (1) is defined inductively as:

- The root is $G(\mathbf{x})$, i.e., the polynomial in the loop guard of MPP (1);
- For any node $G(\mathbf{p}(\mathbf{x}))$ in the tree, where $\mathbf{p}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a vector function, it has ℓ directly successive nodes $G(\mathbf{A}_1(\mathbf{p}(\mathbf{x}))), G(\mathbf{A}_2(\mathbf{p}(\mathbf{x}))), \dots, G(\mathbf{A}_\ell(\mathbf{p}(\mathbf{x})))$; and there is a directed edge connecting $G(\mathbf{p}(\mathbf{x}))$ and $G(\mathbf{A}_i(\mathbf{p}(\mathbf{x})))$, labeled by a_i , for any $1 \leq i \leq \ell$.

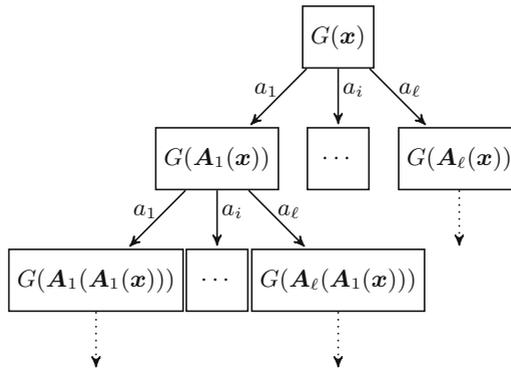


Figure 5 The iteration tree of MPP (1)

Remark 4.2 Iteration trees have the following features compared with execution trees:

- Each node in an iteration tree is a polynomial in $\mathbb{Q}[\mathbf{x}]$, rather than an element in \mathbb{R}^n ;
- Iteration trees are infinite trees with no leaf node, and each node has exactly ℓ children;
- An iteration tree is constructed statically without dynamically executing an MPP to get the trace of program state variables.

4.2 Computing SNTI as Real Varieties

Intuitively, a strong non-terminating input $\mathbf{x} \in \mathbb{R}^n$ of an MPP will persistently satisfy the loop guard $G(\mathbf{x}) = 0$, and thus must be a common real root of all the polynomials in the iteration tree of the MPP. To see this, it is convenient to consider the infinitely many polynomials in an iteration tree level by level, rather than by paths.

Lemma 4.3

$$\text{SNTI} = \mathcal{V} \left(\bigcup_{k \geq 0} \bigcup_{|\tau|=k} \{G(\mathcal{A}_\tau(\mathbf{x}))\} \right) \cap \mathbb{R}^n.$$

Proof By Definition 3.8, $\mathbf{x} \in \text{SNTI}$ if and only if for any $\tau \in \Sigma^\omega$, and any $\tau' \in \text{Pre}(\tau)$, $G(\mathcal{A}_{\tau'}(\mathbf{x})) = 0$, i.e.,

$$\text{SNTI} = \mathcal{V} \left(\bigcup_{\tau \in \Sigma^\omega} \bigcup_{\tau' \in \text{Pre}(\tau)} \{G(\mathcal{A}_{\tau'}(\mathbf{x}))\} \right) \cap \mathbb{R}^n.$$

Hence this lemma follows immediately from the fact

$$\bigcup_{\tau \in \Sigma^\omega} \bigcup_{\tau' \in \text{Pre}(\tau)} \{\tau'\} = \bigcup_{k \geq 0} \bigcup_{|\tau|=k} \{\tau\}.$$

■

To employ the correspondence between ideals and varieties and the nice properties of polynomial ideals, let

$$\mathcal{I}_k \hat{=} \left\langle \bigcup_{|\tau| \leq k} \{G(\mathcal{A}_\tau(\mathbf{x}))\} \right\rangle, \quad k = 0, 1, \dots \quad (7)$$

That is, \mathcal{I}_k is the ideal generated by all the polynomials from the top $k+1$ levels of the iteration tree. Then we have

Lemma 4.4

$$\mathbf{SNTI} = \mathcal{V} \left(\bigcup_{k \geq 0} \mathcal{I}_k \right) \cap \mathbb{R}^n .$$

Proof By (7) it is easy to show that

$$\bigcup_{k \geq 0} \mathcal{I}_k = \bigcup_{k \geq 0} \left\langle \bigcup_{|\tau| \leq k} \{G(\mathcal{A}_\tau(\mathbf{x}))\} \right\rangle = \left\langle \bigcup_{k \geq 0} \bigcup_{|\tau| \leq k} \{G(\mathcal{A}_\tau(\mathbf{x}))\} \right\rangle = \left\langle \bigcup_{k \geq 0} \bigcup_{|\tau|=k} \{G(\mathcal{A}_\tau(\mathbf{x}))\} \right\rangle .$$

Hence this lemma follows from Lemma 4.3. ▀

Lemma 4.4 can be further simplified to the following.

Lemma 4.5 *There exists an integer $N \in \mathbb{N}$ such that*

$$\mathbf{SNTI} = \mathcal{V}(\mathcal{I}_N) \cap \mathbb{R}^n .$$

Proof Note that $\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \dots$ forms an ascending chain of ideals. Hence this lemma follows from Theorem 2.4 and Lemma 4.4. ▀

Now we have reduced **SNTI** of MPP (1) to the real variety of a polynomial ideal. The following theorem shows that the integer N in Lemma 4.5 is actually computable.

Theorem 4.6 (Fixed point theorem) *If $\mathcal{I}_m = \mathcal{I}_{m+1}$, then $\mathcal{I}_m = \mathcal{I}_k$ for any $k \geq m+1$.*

Proof We prove this by induction on k for $k \geq m+1$.

Basis: $\mathcal{I}_m = \mathcal{I}_{m+1}$, which means $G(\mathcal{A}_\tau(\mathbf{x})) \in \mathcal{I}_m$ for any τ with $|\tau| \leq m+1$.

Induction: Assume that $\mathcal{I}_m = \mathcal{I}_k$ for some $k \geq m+1$, which means for any τ with $|\tau| = k$, there exist $g_\sigma(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$, for each $\sigma \in \Sigma^*$ with $|\sigma| \leq m$, such that

$$G(\mathcal{A}_\tau(\mathbf{x})) = \sum_{|\sigma| \leq m} g_\sigma(\mathbf{x}) G(\mathcal{A}_\sigma(\mathbf{x})) .$$

We will show that $\mathcal{I}_m = \mathcal{I}_{k+1}$. Note that for any τ with $|\tau| = k+1$, we have

$$\begin{aligned} G(\mathcal{A}_\tau(\mathbf{x})) &= G(\mathcal{A}_{\text{tail}(\tau)}(\mathcal{A}_{\text{head}(\tau)}(\mathbf{x}))) && (|\text{tail}(\tau)| = k) \\ &= \sum_{|\sigma| \leq m} g_\sigma(\mathcal{A}_{\text{head}(\tau)}(\mathbf{x})) G(\mathcal{A}_\sigma(\mathcal{A}_{\text{head}(\tau)}(\mathbf{x}))) && (\text{by induction hypothesis}) \\ &= \sum_{|\sigma| \leq m} g_\sigma(\mathcal{A}_{\text{head}(\tau)}(\mathbf{x})) G(\mathcal{A}_{\text{head}(\tau) \cdot \sigma}(\mathbf{x})) && (|\text{head}(\tau) \cdot \sigma| \leq m+1) \\ &\in \mathcal{I}_m && (\text{by induction basis}). \end{aligned}$$

By Theorem 4.6, N can be computed as the first integer k satisfying $\mathcal{I}_k = \mathcal{I}_{k+1}$. Now the main result on **SNTI** computation can be stated as:

Theorem 4.7 For MPP (1), an integer $N \in \mathbb{N}$ can be computed such that

$$\mathbf{SNTI} = \mathcal{V}(\mathcal{I}_N) \cap \mathbb{R}^n .$$

Proof It follows immediately from Lemma 4.5 and Theorem 4.6. ■

An abstract algorithm for computing **SNTI** can be given as:

Algorithm 4.8 (The Computation of **SNTI** for MPP (1))

- 1: $k \leftarrow 0$
- 2: **while** $\mathcal{I}_k \neq \mathcal{I}_{k+1}$ **do**
- 3: $k \leftarrow k + 1$
- 4: **end while**
- 5: **return** $\mathcal{V}(\mathcal{I}_k) \cap \mathbb{R}^n$

Example 4.9 We show the application of Algorithm 4.8 to three aforementioned MPPs.

- 1) For MPP (3), we get[‡] $\mathcal{I}_0 = \langle x + y \rangle \subsetneq \mathcal{I}_1 = \langle y^2, x + y \rangle \subsetneq \mathcal{I}_2 = \langle 1 \rangle = \mathcal{I}_3$. So $\mathbf{SNTI} = \mathcal{V}(\langle 1 \rangle) \cap \mathbb{R}^2 = \emptyset$.
- 2) For MPP (4), we get $\mathcal{I}_0 = \langle x^2 + 1 - y \rangle \subsetneq \mathcal{I}_1 = \langle y^2 - 2y, x^2 + 1 - y \rangle = \mathcal{I}_2$. So $\mathbf{SNTI} = \mathcal{V}(\langle y^2 - 2y, x^2 + 1 - y \rangle) \cap \mathbb{R}^2 = \{(1, 2), (-1, 2)\}$.
- 3) For MPP (6), we get $\mathcal{I}_0 = \langle x^2 - 3x + 2 \rangle \subsetneq \mathcal{I}_1 = \langle 1 \rangle = \mathcal{I}_2$. So $\mathbf{SNTI} = \mathcal{V}(\langle 1 \rangle) \cap \mathbb{R} = \emptyset$.

5 The Computation of **WNTI**

In this section, we will first reduce **WNTI** of MPP (1) to the real variety of a certain polynomial ideal \mathcal{J} . We then provide a sufficient criterion under which a finite basis of \mathcal{J} can be computed, and thus give a complete characterization of **WNTI** under such a criterion.

5.1 Identifying **WNTI** as Real Varieties

Intuitively, a weak nonterminating input is a real root of at least one of the polynomials at every level of the iteration tree in Figure 5, or equivalently, a real root of the product of the polynomials (called a product polynomial) at each level.

Lemma 5.1

$$\mathbf{WNTI} = \mathcal{V} \left(\bigcup_{j \geq 0} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right) \cap \mathbb{R}^n .$$

Proof It is a direct translation of (5) in Definition 3.10. ■

[‡]For the computation of Gröbner bases, we are assuming the pure lexicographic order $x \succ y$.

To adopt the language of polynomial ideals, we consider the ideals generated by the product polynomials from the top $k + 1$ levels of the iteration tree:

$$\mathcal{J}_k \hat{=} \left\langle \bigcup_{j \leq k} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right\rangle, \quad k = 0, 1, \dots \quad (8)$$

Then we have

Lemma 5.2

$$\mathbf{WNTI} = \mathcal{V} \left(\bigcup_{k \geq 0} \mathcal{J}_k \right) \cap \mathbb{R}^n.$$

Proof By (8) it is easy to show that

$$\begin{aligned} \bigcup_{k \geq 0} \mathcal{J}_k &= \bigcup_{k \geq 0} \left\langle \bigcup_{j \leq k} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right\rangle = \left\langle \bigcup_{k \geq 0} \bigcup_{j \leq k} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right\rangle \\ &= \left\langle \bigcup_{j \geq 0} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right\rangle. \end{aligned}$$

Hence this lemma follows from Lemma 5.1. ▀

Lemma 5.2 can be further simplified to

Lemma 5.3 *There exists an integer $N \in \mathbb{N}$ such that*

$$\mathbf{WNTI} = \mathcal{V}(\mathcal{J}_N) \cap \mathbb{R}^n.$$

Proof Note that $\mathcal{J}_0 \subseteq \mathcal{J}_1 \subseteq \mathcal{J}_2 \subseteq \dots$ forms an ascending chain of ideals. Hence this lemma follows from Theorem 2.4 and Lemma 5.2. ▀

Now we have proved that **WNTI** of MPP (1) is a real variety, a result analogous to what we have established for **SNTI** (Lemma 4.5). However, we do not have a criterion like Theorem 4.6 for checking the fixed point of the chain of ideals \mathcal{J}_k . Nevertheless, a sufficient criterion will be given under some conditions in the coming subsection.

5.2 A Sufficient Criterion for Computability of WNTI

Let

$$\widehat{\mathcal{J}}_{m,k} \hat{=} \left\langle \bigcup_{m \leq j \leq k} \left\{ \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \right\} \right\rangle, \quad 0 \leq m \leq k. \quad (9)$$

Intuitively, $\widehat{\mathcal{J}}_{m,k}$ is the ideal generated by the product polynomials from levels m to k in the iteration tree in Figure 5.

For the ideals $\widehat{\mathcal{J}}_{m,k}$ and \mathcal{J}_k defined in (9) and (8) respectively, we have $\widehat{\mathcal{J}}_{m,k} \subseteq \mathcal{J}_k$ for any $k \geq m \geq 0$, in particular, $\widehat{\mathcal{J}}_{0,k} = \mathcal{J}_k$ for any $k \geq 0$. Besides, noticing that for any $m \in \mathbb{N}$, the sequence of $\widehat{\mathcal{J}}_{m,k}$ for $k = m, m + 1, m + 2, \dots$ forms an ascending chain of ideals, we can actually prove

Lemma 5.4 For any $M, N \in \mathbb{N}$ with $M \leq N$, if $\widehat{\mathcal{J}}_{M,N}$ is the fixed point of the chain of ideals formed by $\{\widehat{\mathcal{J}}_{M,k}\}_{k \geq M}$, then \mathcal{J}_N is the fixed point of the chain of ideals formed by $\{\mathcal{J}_k\}_{k \geq 0}$.

Proof It is sufficient to prove that for any $k > N$, $\widehat{\mathcal{J}}_{M,N} = \widehat{\mathcal{J}}_{M,k}$ implies $\mathcal{J}_N = \mathcal{J}_k$. Note that $\widehat{\mathcal{J}}_{M,N} = \widehat{\mathcal{J}}_{M,k}$ implies

$$\forall j. N < j \leq k : \prod_{|\tau|=j} G(\mathcal{A}_\tau(\mathbf{x})) \in \widehat{\mathcal{J}}_{M,N} \subseteq \mathcal{J}_N,$$

which means the set of generators of \mathcal{J}_k are all elements of \mathcal{J}_N . Thus $\mathcal{J}_N = \mathcal{J}_k$. ■

As mentioned above, for any $M \in \mathbb{N}$, how to compute the fixed point of the chain of ideals $\widehat{\mathcal{J}}_{M,k}$ for $k \geq M$ is an open problem in general. However, a sufficient criterion can be given as follows, which can be seen as a weak version of Theorem 4.6.

Theorem 5.5 (Weak fixed point theorem) *If there exist $M, N \in \mathbb{N}$, $M \leq N$, such that $\widehat{\mathcal{J}}_{M,N}$ is a principal ideal, i.e., $\widehat{\mathcal{J}}_{M,N} = \langle H(\mathbf{x}) \rangle$ for some $H(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$, and in addition, $H(\mathbf{x})$ satisfies*

$$\prod_{|\tau|=1} H(\mathcal{A}_\tau(\mathbf{x})) \in \langle H(\mathbf{x}) \rangle, \tag{10}$$

then $\mathcal{J}_N = \mathcal{J}_k$ for all $k \geq N$.

Proof By Lemma 5.4, it suffices to prove that $\widehat{\mathcal{J}}_{M,N} = \widehat{\mathcal{J}}_{M,k}$ for all $k \geq N$. The proof will proceed by induction on k for $k \geq N$.

Suppose $\widehat{\mathcal{J}}_{M,N} = \langle H(\mathbf{x}) \rangle = \widehat{\mathcal{J}}_{M,k}$ for some $k \geq N$. Then there exists $g(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ such that $\prod_{|\tau|=k} G(\mathcal{A}_\tau(\mathbf{x})) = g(\mathbf{x})H(\mathbf{x})$.

$$\begin{aligned} \prod_{|\tau|=k+1} G(\mathcal{A}_\tau(\mathbf{x})) &= \prod_{|\tau_1|=1} \prod_{|\tau_2|=k} G(\mathcal{A}_{\tau_2}(\mathcal{A}_{\tau_1}(\mathbf{x}))) \\ &= \prod_{|\tau_1|=1} g(\mathcal{A}_{\tau_1}(\mathbf{x}))H(\mathcal{A}_{\tau_1}(\mathbf{x})) && \text{(by induction hypothesis)} \\ &= \left[\prod_{|\tau_1|=1} g(\mathcal{A}_{\tau_1}(\mathbf{x})) \right] \left[\prod_{|\tau_1|=1} H(\mathcal{A}_{\tau_1}(\mathbf{x})) \right] \\ &\in \langle H(\mathbf{x}) \rangle && \text{(by the requirement (10)).} \end{aligned}$$

Hence we get $\widehat{\mathcal{J}}_{M,N} = \widehat{\mathcal{J}}_{M,k+1}$, which completes the proof by induction. ■

Now the main result on **WNTI** computation can be stated as follows, which is analogous to Theorem 4.7.

Theorem 5.6 *For MPP (1), if the requirements in Theorem 5.5 are satisfied, an integer N can be computed such that*

$$\mathbf{WNTI} = \mathcal{V}(\mathcal{J}_N) \cap \mathbb{R}^n.$$

Proof It follows immediately from Lemma 5.3 and Theorem 5.5. ■

5.3 Applying Theorem 5.6 to Compute WNTI

Based on Theorem 5.6, we present an abstract algorithm for computing **WNTI** of MPP (1):

Algorithm 5.7 (The Computation of **WNTI** for MPP (1))

```

1:  $m \leftarrow 0$ 
2: while  $m \leq M$  do
3:    $k \leftarrow m$ 
4:   while  $\widehat{\mathcal{J}}_{m,k} \neq \widehat{\mathcal{J}}_{m,k+1}$  do
5:      $k \leftarrow k + 1$ 
6:   end while
7:   if  $\widehat{\mathcal{J}}_{m,k} = \langle H(\mathbf{x}) \rangle$  and  $H(\mathbf{x})$  satisfies (10) then
8:     return  $\mathcal{V}(\mathcal{J}_k) \cap \mathbb{R}^n$ 
9:   else
10:     $m \leftarrow m + 1$ 
11:   end if
12: end while
13: return unknown

```

The meaning of Algorithm 5.7 is: for each $m \in \mathbb{N}$, we compute $\widehat{\mathcal{J}}_{m,k}$ for $k = m, m + 1, m + 2, \dots$ until $\widehat{\mathcal{J}}_{m,k} = \widehat{\mathcal{J}}_{m,k+1}$ (lines 3–5)[§]; then we test whether $\widehat{\mathcal{J}}_{m,k}$ is a principal ideal and the single generator satisfies (10) (line 7); if the test succeeds, then by Theorem 5.6 we get the **WNTI** (line 8); otherwise, we increase m by 1 (line 10) and continue the computation; the number M in line 2 is a prescribed bound on m to ensure termination of the whole algorithm, and when m exceeds M , Algorithm 5.7 terminates with the answer **unknown** (line 13).

We use Algorithm 5.7 to compute the **WNTI** of several MPPs.

Example 5.8 1) For MPP (3), we get $\widehat{\mathcal{J}}_{0,0} = \langle x + y \rangle = \widehat{\mathcal{J}}_{0,1}$, in which $x + y$ satisfies (10). So **WNTI** = $\mathcal{V}(\mathcal{J}_0) \cap \mathbb{R}^2 = \mathcal{V}(\langle x + y \rangle) \cap \mathbb{R}^2$.

2) For MPP (4), we get $\widehat{\mathcal{J}}_{0,0} = \langle x^2 + 1 - y \rangle = \widehat{\mathcal{J}}_{0,1}$, in which $x^2 + 1 - y$ satisfies (10). So **WNTI** = $\mathcal{V}(\mathcal{J}_0) \cap \mathbb{R}^2 = \mathcal{V}(\langle x^2 + 1 - y \rangle) \cap \mathbb{R}^2$.

3) For MPP (6), we get $\widehat{\mathcal{J}}_{1,1} = \langle x^4 + x^6 - x^3 - x^5 \rangle \subsetneq \widehat{\mathcal{J}}_{1,2} = \langle x^4 - x^3 \rangle = \widehat{\mathcal{J}}_{1,3}$, in which $x^4 - x^3$ satisfies (10). So **WNTI** = $\mathcal{V}(\mathcal{J}_2) \cap \mathbb{R} = \mathcal{V}(\langle x - 1 \rangle) \cap \mathbb{R} = \{1\}$.

4) Consider the following MPP

$$\text{while } (x - y^2 = 0) \left\{ \begin{array}{l} (x, y) := (x + 3, y + 1); \\ \parallel \\ (x, y) := (1, 0); \end{array} \right\}. \quad (11)$$

We can get $\widehat{\mathcal{J}}_{0,0} = \langle x - y^2 \rangle \subsetneq \widehat{\mathcal{J}}_{0,1} = \langle y - 1, x - 1 \rangle \subsetneq \widehat{\mathcal{J}}_{0,2} = \langle 1 \rangle = \widehat{\mathcal{J}}_{0,3}$. So **WNTI** = $\mathcal{V}(\mathcal{J}_2) \cap \mathbb{R}^2 = \mathcal{V}(\langle 1 \rangle) \cap \mathbb{R}^2 = \emptyset$, which further implies **NTI** = **SNTI** = \emptyset . ■

The results of Examples 4.9 and 5.8 can be summarized in Table 1.

[§]By Theorem 2.4, this iteration must terminate.

Table 1 A summary of the computation of non-terminating inputs

MPP	(3)	(4)	(6)	(11)
SNTI	\emptyset	$\{(1, 2), (-1, 2)\}$	\emptyset	\emptyset
NTI	?	$\supseteq \{(1, 2), (-1, 2)\}$?	\emptyset
WNTI	$\mathcal{V}(\langle x + y \rangle) \cap \mathbb{R}^2$	$\mathcal{V}(\langle x^2 + 1 - y \rangle) \cap \mathbb{R}^2$	$\{1\}$	\emptyset

We explain a bit about the row corresponding to **NTI** in Table 1. For MPP (4), it can be inferred that **NTI** is a superset of $\{(1, 2), (-1, 2)\}$ and must be nonempty. For MPP (11), the emptiness of **WNTI** implies the emptiness of **NTI**. For MPPs (3) and (6), we cannot infer any information from the **SNTI/WNTI**-approximations and therefore the **NTI** are marked with “?”. Actually from the analysis of Examples 3.5 and 3.11 we have known that $(1, -1)$ is in the **NTI** of MPP (3), while the **NTI** of MPP (6) is empty.

6 Conclusion

In this paper, we studied the detection of non-terminating inputs over \mathbb{R} for multi-path polynomial programs with equational loop guards (MPP (1)). We first defined on MPP the notions of non-terminating inputs (**NTI**) as well as its under/over-approximation, i.e., strong/weak non-terminating inputs (**SNTI/WNTI**). Then based on the well-known theory of polynomial ideals, we presented a complete approach for computing **SNTI**, and a sound but incomplete approach for computing **WNTI**, both as the real variety of a certain polynomial ideal. Once the **SNTI/WNTI** are computed, useful information on **NTI** can be inferred.

We have experimented with the proposed algorithms on several simple examples, by interactively calling existing functions in the Maple environment. We did these experiments mainly with the purpose of testing and demonstrating proposed algorithms. A fully automated tool with applications to real programs is not provided, due to the high complexity of our approach: the polynomial ideal membership problem is exponential space complete in the size of the problem instance^[30], and the computation of Gröbner basis has even higher worst case complexity; besides, for MPPs with more than one paths, the sizes of ideals \mathcal{I}_k or \mathcal{J}_k in our algorithm grow exponentially with k , which could be very large when the fixed point is reached. An estimation of how large k could eventually be is required for a more precise complexity analysis.

There are two theoretical problems left open in this paper:

- 1) Can we drop the restrictive assumption on **WNTI** computation and give a complete algorithm for computing **WNTI**?
- 2) Is **NTI** reducible to the real variety of a polynomial ideal? If so, how can this variety be computed (at least under some restrictions)?

A negative conjecture could be that a complete description of **WNTI** or **NTI** is not computable for MPP (1) in general. We are interested in answering these questions in future.

Acknowledgements We thank Professors ZHOU Chaochen, YANG Lu, LI Zhibin, XIA

Bican, WANG Mingsheng, and WANG Dinggang for their beneficial discussions on our work. We also thank anonymous reviewers for their valuable comments on the revision of this paper.

References

- [1] Cook B, Podelski A, and Rybalchenko A, Proving program termination, *Communications of the ACM*, 2011, **54**(5): 88–98.
- [2] Yang L, Zhou C, Zhan N, and Xia B, Recent advances in program verification through computer algebra, *Frontiers of Computer Science in China*, 2010, **4**(1): 1–16.
- [3] Turing A M, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, 1936, **42**(2): 230–265.
- [4] Tiwari A, Termination of linear programs, *Proceedings of CAV 2004* (ed. by Alur R and Peled D A), LNCS 3114, Springer, Berlin, 2004, 70–82.
- [5] Bradley A R, Manna A, and Sipma H B, Termination of polynomial programs, *Proceedings of VMCAI 2005* (ed. by Cousot R), LNCS 3385, Springer, Berlin, 2005, 113–129.
- [6] Braverman M, Termination of integer linear programs, *Proceedings of CAV 2006* (ed. by Ball T and Jones R B), LNCS 4144, Springer, Berlin, 2006, 372–385.
- [7] Harris W R, Lal A, Nori A V, and Rajamani S K, Alternation for termination, *Proceedings of SAS 2010* (ed. by Cousot R and Martel M), LNCS 6337, Springer, Berlin, 2010, 304–319.
- [8] Colón M A and Sipma H B, Synthesis of linear ranking functions, *Proceedings of TACAS 2001* (ed. by Margaria T and Wang Y), LNCS 2031, Springer, Berlin, 2001, 67–81.
- [9] Podelski A and Rybalchenko A, A complete method for the synthesis of linear ranking functions, *Proceedings of VMCAI 2004* (ed. by Steffen B and Levi G), LNCS 2937, Springer, Berlin, 2004, 239–251.
- [10] Xia B and Zhang Z, Termination of linear programs with nonlinear constraints, *Journal of Symbolic Computation*, 2010, **45**(11): 1234–1249.
- [11] Xia B, Yang L, Zhan N, and Zhang Z, Symbolic decision procedure for termination of linear programs, *Formal Aspects of Computing*, 2011, **23**(2): 171–190.
- [12] Xu M, Chen L, Zeng Z, and Li Z, Termination analysis of linear loops, *International Journal of Foundations of Computer Science*, 2010, **21**(6): 1005–1019.
- [13] Xu M and Li Z, Symbolic termination analysis of solvable loops, *Journal of Symbolic Computation*, 2013, **50**: 28–49.
- [14] Cook B, Gulwani S, Lev-Ami T, Rybalchenko A, and Sagiv M, Proving conditional termination, *Proceedings of CAV 2008* (ed. by Gupta A and Malik S), LNCS 5123, Springer, Berlin, 2008, 328–340.
- [15] Sankaranarayanan S, Sipma H B, and Manna Z, Non-linear loop invariant generation using Gröbner bases, *Proceedings of POPL 2004* (ed. by Jones N D and Leroy X), ACM Press, New York, 2004, 318–329.
- [16] Müller-Olm M and Seidl H, Computing polynomial program invariants, *Information Processing Letters*, 2004, **91**(5): 233–244.

- [17] Kapur D, A quantifier-elimination based heuristic for automatically generating inductive assertions for programs, *Journal of Systems Science and Complexity*, 2006, **19**(3): 307–330.
- [18] Rodríguez-Carbonell E and Kapur D, Generating all polynomial invariants in simple loops, *Journal of Symbolic Computation*, 2007, **42**(4): 443–476.
- [19] Kovács L, Reasoning algebraically about P-solvable loops, *Proceedings of TACAS 2008* (ed. by Ramakrishnan C R and Rehof J), LNCS 4963, Springer, Berlin, 2008, 249–264.
- [20] Chen Y, Xia B, Yang L, Zhan N, and Zhou C, Discovering non-linear ranking functions by solving semi-algebraic systems, *Proceedings of ICTAC 2007* (ed. by Jones C B, Liu Z, and Woodcock J), LNCS 4711, Springer, Berlin, 2007, 34–49.
- [21] Gupta A, Henzinger T A, Majumdar R, Rybalchenko A, and Xu R, Proving non-termination, *Proceedings of POPL 2008* (ed. by Necula G C and Wadler P), ACM Press, New York, 2008, 147–158.
- [22] Cook B, Podelski A, and Rybalchenko A, Termination proofs for systems code, *Proceedings of PLDI 2006* (ed. by Schwartzbach M I and Ball T), ACM Press, New York, 2006, 415–426.
- [23] Velroyen H and Rümmer P, Non-termination checking for imperative programs, *Proceedings of TAP 2008* (ed. by Beckert B and Hähnle R), LNCS 4966, Springer, Berlin, 2008, 154–170.
- [24] Brockschmidt M, Ströder T, Otto C, and Giesl J, Automated detection of non-termination and NullPointerExceptions for Java bytecode, *Proceedings of FoVeOOS 2011* (ed. by Beckert B, Damiani F, and Gurov D), LNCS 7421, Springer, Berlin, 2011, 123–141.
- [25] Cousot P, Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming, *Proceedings of VMCAI 2005* (ed. by Cousot R), LNCS 3385, Springer, Berlin, 2005, 1–24.
- [26] Shen L, Wu M, Yang Z, and Zeng Z, Finding positively invariant sets of a class of nonlinear loops via curve fitting, *Proceedings of SNC 2009* (ed. by Kai H and Sekigawa H), ACM Press, New York, 2009, 185–190.
- [27] Cox D, Little J, and O’Shea D, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, UTM, Springer, Berlin, 3 edition, 2007.
- [28] Xia B and Yang L, An algorithm for isolating the real solutions of semi-algebraic systems, *Journal of Symbolic Computation*, 2002, **34**(5): 461–477.
- [29] Xia B, Discoverer: A tool for solving semi-algebraic systems, *ACM Sigsam Bulletin*, 2007, **41**(2): 102–103.
- [30] Mayr E W, Membership in polynomial ideals over \mathbb{Q} is exponential space complete, *Proceedings of STACS 1989* (ed. by Monien B and Cori R), LNCS 349, Springer, Berlin, 1989, 400–406.