Generating Asymptotically Non-Terminant Initial Variable Values for Linear Diagonalizable Programs

Rachid Rebiha[1]*, Nadir Matringe[2,3] and Arnaldo Vieira Moura[1]

[1] Institute of Computing UNICAMP, University of Campinas, SP. Brasil.
`rachid@ic.unicamp.br`
[2] Universit de Poitiers, Laboratoire Mathmatiques et Applications, France.
[3] Institue de Mathematiques de Jussieu, Université Paris 7-Denis Diderot, France.

Abstract

We present the key notion of *asymptotically non-terminant initial variable values* for non-terminant loop programs. We show that those specific values are directly associated to inital variable values for which the corresponding loop program does not terminate. Considering linear diagonalizable programs, we describe powerful computational methods that generate automatically and symbolically a semi-linear space represented by a linear system of equalities and inequalities. Each element of this space provides us with asymptotically non-terminant initial variable values. Our approach is based on linear algebraic methods. We obtain specific conditions using certain basis and matrix encodings related to the loop conditions and instructions.

1 Introduction

Research on formal methods for program verification [12, 15, 8, 17] aims at discovering mathematical techniques and developing their associated algorithms to establish the correctness of software, hardware, concurrent systems, embedded systems or hybrid systems. Static program analysis [12, 15], is used to check that a software is free of defects, such as buffer overflows or segmentation faults, which are safety properties, or termination and non-termination, which are liveness properties. Proving termination of while loop programs is necessary for the verification of liveness properties that any well behaved and engineered system, or any safety critical embedded system must guarantee. We could list here many verification approaches that are only practical depending on the facility with which termination can be automatically determined. Verification of temporal properties of infinite state systems [20] is another example.

Recent work on automated termination analysis of imperative loop programs has focused on a partial decision procedure based on the discovery and synthesis of ranking functions. Such functions map the loop variable to a well-defined domain where their value decreases further at each iteration of the loop [9, 10]. Several interesting approaches, based on the generation of *linear* ranking functions, have been proposed [3, 4] for loop programs where the guards and the instructions can be expressed in a logic supporting linear arithmetic. For the generation of such functions, there are effective heuristics [14, 10] and, in some cases, there are also complete methods for the synthesis of linear ranking functions [16]. On the other hand, it is easy to generate a simple linear terminant loop program that does not have a linear ranking function. And in this case such complete synthesis methods [16] fail to provide a conclusion about the termination or the non termination of such a program.

*Supported by FAPESP grant 2011/08947-1 and FAPESP grant BEPE 2013/04734-9.

In this work we address the non-termination problem for linear while loop programs. In other words, we consider the class of loop programs where the loop condition is a conjunction of linear inequalities and the assignments to each of the variables in the loop instruction block, are affine or linear forms. In matrix notation, *linear loop programs* will be represented in a general form as: while $(Bx > b)$, $\{x := Ax + c\}$ (i.e., A and B are matrices, b and c are vectors of real numbers, and that x is a vector of variables.). Without loss of generality, the termination/non-termination analysis for such a class of linear programs could be reduced to the problem of termination/non-termination for homogeneous linear programs [6, 21]. Those being programs where linear assignments consist of homogeneous expressions, and where the linear loop condition consists of at most one inequality. Concerning effective program transformations and simplification techniques, non-termination analysis for programs presented in a more complex form can often be reduced to an analysis of a program expressed in this basic affine form. Despite tremendous progress over the years [6, 5, 7, 13, 11, 2, 1], the problem of finding a practical, sound and complete methods for determining or analyzing non termination remains very challenging for this class of programs, and for all initial variable values. We started our investigation from our preliminary technical reports [19, 18] where we introduced a termination analysis in which algorithms ran in polynomial time complexity. Here, considering a non terminating loop, we introduce new static analysis methods that compute automatically, and in polynomial time complexity, the set of initial input variable values for which the program does not terminate, and also a set of initial inputs values for which the program does terminate. This justifies the innovation of our contributions, *i.e.*, none of the other mentioned related work is capable of generating such critical information over non-terminating loops. We summarize our contributions as follows:

- To the best of our knowledge, we introduce a new key notion for non-termination and termination analysis for loop programs: we identify the important concept of *asymptotically non-terminant initial variable values*, ANT for short. Any asymptotically non-terminant initial variable values can be directly related to an initial variable value for which the considered program does not terminate.

- Our theoretical contributions provide us with efficient computational methods running in polynomial time complexity and allowing the exact computation of the set of all asymptotically non-terminant initial variable values for a given loop program.

- We generate automatically a set of linear equalities and inequalities describing a semi-linear space that represents symbolically the set of all asymptotically non-terminant initial variable values. The set of ANT values contains the set of non-terminant initial variable values. On the other hand the complementary set of ANT values is a precise under-approximation of the set of terminant inputs for the same program.

Example 1.1. (Motivating Example) *Consider the following program depicted below on the left. We show a part of the output of our algorithm below on the right.*

(i) Pseudo code:

```
while(2x+3y-z>0){
  x:= y + z;
  y:=-(1/2)x+(3/2)y-(1/2)z;
  z:=(3/2)x-(3/2)y+(1/2)z;}
```

(ii) Output of our prototype:

```
Locus of ANT
[[4u[1]+u[2]+u[3]>0]
AND[u[1]+4u[2]+4u[3]>0]
AND[-u[1]+u[2]-u[3]=0]]
OR[[-u[1]+u[2]-u[3]>0]]
```

The semi-linear $ANT = \{u = (u_1, u_2, u_3)^\top \in E \mid 4u_1 + u_2 + u_3 > 0 \wedge u_1 + 4u_2 + 4u_3 > 0 \wedge -u_1 + u_2 - u_3 = 0\} \cup \{u = (u_1, u_2, u_3)^\top \in E \mid -u_1 + u_2 - u_3 > 0\}$ *represents symbolically all asymptotically initial variable values that are directly associated to initial variable values for which the program does not terminate. On the other hand, the complementary of this set* $co\text{-}ANT = \{u = (u_1, u_2, u_3)^\top \in E \mid 4u_1 + u_2 + u_3 \leq 0 \vee u_1 + 4u_2 + 4u_3 \leq 0 \vee -u_1 + u_2 - u_3 \neq 0\} \cap \{u = (u_1, u_2, u_3)^\top \in E \mid -u_1 + u_2 - u_3 \leq 0\}$ *is a precise under-approximation of the set of all initial variable values on which the program terminates.* $\square$

The rest of this article is organized as follows. Section 2 can be seen as a preliminary section where we introduce some key notions and results from linear algebra, which will be used to build our computational methods. In this section, we also present our computational model for programs and some further notations. Section 3 introduces the new notion of *asymptotically non-terminant initial variable values.* Section 4 provides the main theoretical contributions of this work. This section also presents our computational methods that generate a symbolic representation of the asymptotically non-terminant variable values for linear programs. We provide a complete dicussion in Section 5. Finally, Section 6 states our conclusions.

2 Linear Algebra and Linear Loop Programs

Here, we present key linear algebraic notions and results which are central in the theoretical and algorithmic development of our methods. We denote by $\mathcal{M}(m, n, \mathbb{K})$ the set of $m \times n$ matrices with entries in $\mathbb{K}$, and simply $\mathcal{M}(n, \mathbb{K})$ when $m = n$. The Kernel of A, also called its *nullspace*, denoted by $Ker(A)$, is $Ker(A) = \{v \in \mathbb{K}^n \mid A \cdot v = 0_{\mathbb{K}^m}\}$. In fact, when we deal with square matrices, these Kernels are *Eigenspaces.* Let A be a $n \times n$ square matrix with entries in $\mathbb{K}$. A nonzero vector $x \in \mathbb{K}$ is an eigenvector for A associated with the eigenvalue $\lambda \in \mathbb{K}$ if $A \cdot x = \lambda x$, i.e., $(A - \lambda I_n) \cdot x = 0$ where I_n is the $n \times n$ identity matrix. The nullspace of $(A - \lambda I_n)$ is called the *eigenspace* of A associated with eigenvalue λ. Let n be a positive integer, we will denote $\mathbb{R}^n$ by E and its canonical basis by $B_c = (e_1, \ldots, e_n)$. Let A be a square matrix in $\mathcal{M}(n, \mathbb{R})$. Let us introduce the notation $Spec(A)$ for the set of eigenvalues of A in $\mathbb{R}$, and we will write $Spec(A)^*$ for the set $Spec(A) - \{0\}$. For $\lambda \in Spec(A)$, we will denote by $E_\lambda(A)$ the corresponding eigenspace. Throughout the paper we write d_λ for the dimension of $E_\lambda(A)$. We will say that A is diagonalizable if $E = \oplus_{\lambda \in Spec(A)} E_\lambda(A)$. Let A be a diagonalizable matrix. For each eigenspace $E_\lambda(A)$, we will take a basis $B_\lambda = (e_{\lambda,1}, \ldots, e_{\lambda,d_\lambda})$, and we define

$$B = \cup_{\lambda \in Spec(A)} B_\lambda$$

as a basis for E.

Definition 2.1. *Let x belong to E. We denote by x_λ its component in $E_\lambda(A)$. If x admits the decomposition $\sum_{\lambda \in Spec(A)} (\sum_{i=1}^{d_\lambda} x_{\lambda,i} e_{\lambda,i})$ in B, we have $x_\lambda = \sum_{i=1}^{d_\lambda} x_{\lambda,i} e_{\lambda,i}$.* $\square$

We denote by P the transformation matrix corresponding to B, whose columns are the vectors of B, decomposed in B_c. Letting d_i denote the integer d_{λ_i} for the ease of notation, we recall the following lemma.

Lemma 2.1. *We have $P^{-1}AP = diag(\lambda_1 Id_1, \ldots, \lambda_t Id_t)$. We denote by D the matrix $P^{-1}AP$.* $\square$

As we will obtain our results using the decomposition of x in B, we recall how one obtains it from the decomposition of x in B_c.

Lemma 2.2. *Let $x \in E$. If $x = \sum_{i=1}^{n} x_i e_i = (x_1, ..., x_n)^\top \in B_c$, and x decomposes as $\sum_{j=1}^{t}(\sum_{i=1}^{d_j} x_{\lambda_j,i} e_{\lambda_j,i})$ in B, the coefficients $x_{\lambda_j,i}$ are those of the column vector $P^{-1}x$ in B_c.* □

Throughout the paper we write $<\ ,\ >$ for the canonical scalar product on E, which is given by $< x, y >= \sum_{i=1}^{n} x_i y_i$, and recall, as it is standard in static program analysis, that a primed symbol x' refers to the next state value of x after a transition is taken. Next, we present *transition systems* as representations of imperative programs and *automata* as their computational models.

Definition 2.2. *A* transition system *is given by $\langle x, L, \mathcal{T}, l_0, \Theta \rangle$, where $x = (x_1, ..., x_n)$ is a set of variables, L is a set of locations and $l_0 \in L$ is the initial location. A* state *is given by an interpretation of the variables in x. A* transition *$\tau \in \mathcal{T}$ is given by a tuple $\langle l_{pre}, l_{post}, q_\tau, \rho_\tau \rangle$, where l_{pre} and l_{post} designate the pre- and post- locations of τ, respectively, and the transition relation ρ_τ is a first-order assertion over $x \cup x'$. The transition guard q_τ is a conjunction of inequalities over x. Θ is the initial condition, given as a first-order assertion over x. The transition system is said to be* linear *when ρ_τ is an affine form.* □

We will use the following matrix notations to represent loop programs and their associated transitions systems.

Definition 2.3. *Let P be a loop program represented by the transition system $\langle x = (x_1, ..., x_n), l_0, \mathcal{T} = \langle l, l, q_\tau, \rho_\tau \rangle, l_0, \Theta \rangle$. We say that P is a* linear loop program *if the following conditions hold:*

- *Transition guards are conjunctions of linear inequalities. We represent the loop condition in matrix form as $Vx > b$ where $V \in \mathcal{M}(m, n, \mathbb{R})$ and $b \in \mathbb{R}^m$. By $Vx > b$ we mean that each coordinate of vector Vx is greater than the corresponding coordinate of vector b.*
- *Transition relations are affine or linear forms. We represent the linear assignments in matrix form as $x := Ax + c$, where $A \in \mathcal{M}(n, \mathbb{R})$ and $c \in \mathbb{R}^n$.*

The linear loop program *$P = P(A, V, b, c)$ will be represented in its most general form as* while *$(Vx > b)$, $\{x := Ax + c\}$.* □

In this work, we use the following linear loop program classifications.

Definition 2.4. *We identify the following three types of linear loop programs, from the more specific to the more general form:*

- Homogeneous*: We denote by $P^{\mathbb{H}}$ the set of programs where all linear assignments consist of* homogeneous *expressions, and where the linear loop condition consists of at most one inequality. If P is in $P^{\mathbb{H}}$, then P will be written in matrix form as* while *$(< v, x >> 0)$, $\{x := Ax\}$, where v is a $(n \times 1)$-vector corresponding to the loop condition, and where $A \in \mathcal{M}(n, \mathbb{R})$ is related to the list of assignments in the loop. We say that P has a* homogeneous *form and it will also be denoted as $P(A, v)$.*
- Generalized Condition*: We denote by $P^{\mathbb{G}}$ the set of linear loop programs where the loop condition is* generalized *to a conjunction of multiple linear homogeneous inequalities.*
- Affine Form*: We denote by P^{A} the set of loop programs where the inequalities and the assignments are generalized to affine expressions. If P is in P^{A}, it will be written as* while *$(Vx > b)$, $\{x := Ax + c\}$, for A and V in $\mathcal{M}(n, \mathbb{R})$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.*

□

Without lost of generality, the static analysis for the class of linear programs P^{A} could be reduced to the problem of termination/non-termination for homogeneous linear programs in $P^{\mathbb{H}}$. In this work we consider programs in $P^{\mathbb{H}}$. The generalization to programs in P^{A} can already be done and it will be reported elsewhere.

3 Asymptotically Non-terminant Variable Values

In this section we present the new notion of *asymptotically non-terminant* variable values. It will prove to be central in analysis of termination, in general. Let A be a matrix in $\mathcal{M}(n, \mathbb{R})$, and v be a vector in E. Consider the program $P(A, v)$: while $(< v, x >> 0)$, $\{x := Ax\}$, which takes values in E. We first give the definition of the termination of such a program.

Definition 3.1. *The program $P(A, v)$ is said to be terminating on $x \in E$, if and only if $< v, A^k(x) >$ is not positive, for some $k \geq 0$.* □

In other words, Definition 3.1 states that if the program $P(A, v)$ performs $k \geq 0$ loop iterations from initial variables $x$in E, we obtain $x := A^k(x)$. Thus, if $< v, A^k(x) >\leq 0$, the loop condition is violated, and so $P(a, v)$ terminates. Next, we introduce the following important notion.

Definition 3.2. *We say that $x \in E$ is an asymptotically non terminating value for $P(A, v)$ if there exists $k_x \geq 0$ such that $P(A, v)$ is non terminating on $A^{k_x}(v)$. We will write that x is ANT for $P(A, v)$, or simply x is ANT. We will also write that $P(A, v)$ is ANT on x.* □

Note that if $P(A, v)$ is non terminating on $A^{k_x}(x)$ then $< v, A^k(x) >$ is > 0 for $k \geq k_x$.

The following result follows directly from the previous definition.

Corollary 3.1. *An element $x \in E$ is ANT if and only if $< v, A^k(x) >$ is positive for k large enough.* □

If the set of ANT points is not empty, we say that the program $P(A, v)$ is ANT. We will also write NT for non terminant. For the programs we study here, the following lemma already shows the importance of such a set.

Lemma 3.1. *The program $P(A, v)$ is NT if and only if it is ANT.* □

Proof. It is clear that NT implies ANT (i.e., $NT \subseteq ANT$), as a NT point of $P(A, v)$ is of course ANT (with $k_x = 0$). Conversely, if $P(A, v)$ is ANT, call x an ANT point, then $A^{k_x}(x)$ is a NT point of $P(A, v)$, and so $P(A, v)$ is NT. □

As one can easily see, the set of NT points is included in the set of ANT points. But the most important property of the ANT set remains in the fact that each of its point provides us with an associated element in NT for the corresponding program. In other words, each element x in the ANT set, even if it does not necessarily belong to the NT set, refers directly to initial variable values $y_x = A^{k_{[x]}}(x)$ for which the program does not terminate, i.e., y_x is an NT point. We can say that there exists a number of loop iterations $k_{[x]}$, departing from the initial variable values x, such that $A^{k_{[x]}}(x)$ correspond to initial variable values for which $P(A, v)$ does not terminate. But, it does not necessarily implies that x is also an NT point for $P(A, v)$. In fact, program $P(A, v)$ could terminate on the initial variable values x by performing a number of loop iterations strictly smaller than $k_{[x]}$. On the other hand, the complementary set co-ANT provides us with a quite precise under approximation of the set of all initial variable values for which the program terminates.

The set of ANT points of a program is also important to the understanding of the termination of a program with more than one conditional linear inequality, as well as for termination over the rationals, for example. This will be reported elsewhere.

4 Automated generation of ANT loci

In this section we show how we generate automatically and exactly the *ANT Locus*, i.e., the set of all *ANT* points for linear diagonalizable programs over the reals. We represent symbolically the computed *ANT Locus* by a semi-linear space defined by linear equalities and inequalities. Consider the program $P(A, v)$: while $(< v, x >> 0)$, $\{x := Ax\}$. The study of *ANT* sets depends on the spectrum of A. Recall the introductory discussion and Definition 2.1, at Section 2.

Proposition 4.1. *For x in E, and $k \geq 1$, the scalar product $< v, A^k(x) >$ is equal to*

$$\sum_{\lambda \in Spec(A)} \lambda^k < v, x_\lambda >= \sum_{\lambda \in Spec(A)^*} \lambda^k < v, x_\lambda > . \quad \square$$

Proof. It is a direct consequence of the equality $A^k(x_\lambda) = \lambda^k x_\lambda$. $\square$

4.1 The regular case

We first analyze the situation where A is what we call regular:

Definition 4.1. *We say that A is regular, if $Spec(A) \cap \big[-Spec(A)\big]$ is an empty set, i.e.: if λ belongs to $Spec(A)$, then $-\lambda$ does not belong to $Spec(A)$.* $\square$

In this case, we make the following observation:

Proposition 4.2. *Let μ be the nonzero eigenvalue of largest absolute value, if it exists, such that $< v, x_\mu >$ is nonzero. For k large, the quantity $< v, A^k(x) >$ is equivalent to $\mu^k < v, x_\mu >$.* $\square$

Proof. Indeed, we can write $< v, A^k(x) >$ as

$$\mu^k < v, x_\mu > + \sum_{\{\lambda, |\lambda| < |\mu|\}} \lambda^k < v, x_\lambda >= \mu^k (< v, x_\mu > + \sum_{\{\lambda, |\lambda| < |\mu|\}} \frac{\lambda^k}{\mu^k} < v, x_\lambda >),$$

and $\frac{\lambda^k}{\mu^k}$ approaches zero when k goes to infinity. $\square$

We then define the following sets.

Definition 4.2. *For μ a **positive** eigenvalue of A, we define S_μ as follows:*

$$S_\mu = \{x \in E, < v, x_\mu >> 0, < v, x_\lambda >= 0 \ for \ |\lambda| > |\mu|\}$$

$\square$

In order to obtain S_μ for all **positive** eigenvalues μ of A, one needs to calculate $< v, x_\mu >$, and $< v, x_\lambda >$ for all eigenvalues λ such that $|\lambda| > |\mu|$. For all eigenvalues λ involved in the computation of S_μ, one also needs to evaluate the coefficients $c_{\lambda,i} =< v, e_{\lambda,i} >$ for all eigenvectors $e_{\lambda,i} \in B$ and $1 \le i \le d_\lambda$. Thus, we have $< v, x_\lambda >= \sum_{i=1}^{d_\lambda} c_{\lambda,i} x_{\lambda,i}$. Now, we only need to compute the coefficient $x_{\lambda,i}$, which are those of the column vector $P^{-1}x$ in B_c where P is the transformation matrix corresponding to B. See Lemma 2.2.

We ca now state the following theorem, allowing the exact computation of the ANT Locus for the regular case.

Theorem 4.1. *The program $P(A, v)$ is ANT on x if and only if x belongs to*

$$\bigcup_{\mu > 0 \in Spec(A)} S_\mu.$$

□

Proof. Let x belong to E. If all $< v, x_\lambda >$ are zero for $\lambda \in Spec(A)$, then $< v, A^k(x) >= 0$ and x is not ANT. Otherwise, let μ be the eigenvalue of largest absolute value, such that $< v, x_\mu >$ is nonzero. Then, according to Proposition 4.2, the sign of $< v, A^k(x) >$ is the sign of $\mu^k < v, x_\mu >$ for k large. If μ is negative, this sign will be alternatively positive and negative, depending on the parity of k, and x is not ANT. If μ is positive, this sign will be the sign of $< v, x_\mu >$, hence x will be ANT if and only if $< v, x_\mu >$ is positive. This proves the result. □

Example 4.1. (Running example) *Consider the program $P(A, v)$ depicted as follows:*

(i) Pseudo code:

```
while(x+y-2z>0){
  x:= x-4y-4z;
  y:= 8x-11y-8z;
  z:= -8x+8y+5z;}
```

(ii) Associated matrices:

$$A = \begin{pmatrix} 1 & -4 & -4 \\ 8 & -11 & -8 \\ -8 & 8 & 5 \end{pmatrix}, \text{ and } v = \begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix}.$$

Step 1*: Diagonalization of the matrix A:*

$$P = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}, D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -3 \end{pmatrix} \text{ and } P^{-1} = \begin{pmatrix} 1 & -1 & -1 \\ -2 & 3 & 2 \\ 2 & -2 & -1 \end{pmatrix}.$$

Using our notations, the obtained eigenvectors (i.e., the column vectors of P) are denoted as follows: $e_{1,1} = (1, 2, -2)^\top$; $e_{-3,1} = (1, 1, 0)^\top$; $e_{-3,2} = (1, 0, 1)^\top$.

Step 2*: We compute S_μ for all positive $\mu \in Spec(A)^*$:*

- *We compute first the coefficients $c_{\lambda,i}$:*
  $c_{1,1} =< v, e_{1,1} >=< (1, 1, -2)^\top, (1, 2, -2)^\top >= 7,$
  $c_{-3,1} =< v, e_{-3,1} >=< (1, 1, -2)^\top, (1, 1, 0)^\top >= 2,$
  $c_{-3,2} =< v, e_{-3,1} >=< (1, 1, -2)^\top, (1, 0, 1)^\top >= -1$

- *We compute the coefficient $x_{\lambda,i}$, which are those of the column vector $P^{-1} \cdot u$ in B_c, where $u = (u_1, u_2, u_3)$ is the vector encoding the initial variable values.*
  $$P^{-1}. \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} u_1 - u_2 - u_3) \\ -2u_1 + 3u_2 + 2u_3) \\ 2u_1 - 2u_2 - u_3) \end{pmatrix} = \begin{pmatrix} x_{1,1} \\ x_{-3,1} \\ x_{-3,2} \end{pmatrix}.$$
  We can now proceed to the generation of the linear constraints defining a semi-linear space

describing symbolically and exactly S_μ.
$< v, x_1 >= c_{1,1}x_{1,1} = 7(u_1 - u_2 - u_3)$
$< v, x_{-3} >= c_{-3,1}x_{-3,1} + c_{-3,2}x_{-3,2} = -6u_1 + 8u_2 + 5u_3)$
Hence, we have: $S_1 = \{u = (u_1, u_2, u_3)^\top \in E \mid (u_1 - u_2 - u_3 > 0) \wedge (-6u_1 + 8u_2 + 5u_3 = 0)\}$.

Step 3*: We apply Theorem 4.1 to generate the ANT Locus. It reduces to the semi-linear space* $S_1 = \{u = (u_1, u_2, u_3)^\top \in E \mid (u_1 - u_2 - u_3 > 0) \wedge (-6u_1 + 8u_2 + 5u_3 = 0)\}$. □

4.2 The general case: handling linear diagonalizable programs

For the general case, in the asymptotic expansion of $< v, A^k(x) >$ one can have compensations between $\lambda^k < v, x_\lambda >$ and $(-\lambda)^k < v, x_\lambda >$, as these terms can be zero when k is even, for instance, and of a well determined sign when k is odd. We thus need to take care of this issue. To this end, we introduce the following notation.

Definition 4.3. *If* λ *does not belong to Spec*(A)*, for any* $x \in E$*, we write* $x_\lambda = 0$. □

We have the following propositions, which give the asymptotic behavior of $< v, A^k(x) >$.

Proposition 4.3. *Let* μ *be the nonzero eigenvalue of largest absolute value, if it exists, such that* $< v, x_{|\mu|} + x_{-|\mu|} >$ *is nonzero. For* k *large, the quantity* $< v, A^{2k}(x) >$ *is equivalent to* $|\mu|^{2k}(< v, x_{|\mu|} + x_{-|\mu|} >)$. □

Proof. Indeed, we can write $< v, A^{2k}(x) >$ as

$$\mu^{2k}(< v, x_{|\mu|} > + < v, x_{-|\mu|} >) + \sum_{\{|\lambda|, |\lambda| < |\mu|\}} \lambda^{2k}(< v, x_{|\lambda|} > + < v, x_{-|\lambda|} >)$$

$$= \mu^{2k}(< v, x_{|\mu|} > + < v, x_{-|\mu|} >) + \sum_{\{|\lambda|, |\lambda| < |\mu|\}} \frac{\lambda^{2k}}{\mu^{2k}}(< v, x_{|\lambda|} > + < v, x_{-|\lambda|} >)).$$

and $\frac{\lambda^k}{\mu^k}$ approaches to zero when k goes to infinity. □

Proposition 4.4. *Let* μ *be the nonzero eigenvalue of largest absolute value, if it exists, such that* $< v, x_{|\mu|} - x_{-|\mu|} >$ *is nonzero. For* k *large, the quantity* $< v, A^{2k+1}(x) >$ *is equivalent to* $|\mu|^{2k+1}(< v, x_{|\mu|} - x_{-|\mu|} >)$. □

Proof. The proof is similar to the proof of Proposition 4.3. □

As in the previous Section, we introduce the following relevant sets.

Definition 4.4. *For* $|\mu|$ *in* $|Spec(A)^*|$*, we define the sets* $S^0_{|\mu|}$ *and* $S^1_{|\mu|}$ *as follows:*

$$S^0_{|\mu|} = \{x \in E, < v, x_{|\mu|} + x_{-|\mu|} >> 0, < v, x_{|\lambda|} + x_{-|\lambda|} >= 0 \ for \ |\lambda| > |\mu|\}, \ and$$

$$S^1_{|\mu|} = \{x \in E, < v, x_{|\mu|} - x_{-|\mu|} >> 0, < v, x_{|\lambda|} - x_{-|\lambda|} >= 0 \ for \ |\lambda| > |\mu|\}.$$

□

In order to compute the sets $S^0_{|\mu|}$ and $S^1_{|\mu|}$, we obtain the coefficients $c_{\lambda,i} =< v, e_{\lambda,i} >$ for all eigenvalues λ. If the λ appearing as index in the coefficient $c_{\lambda,i}$ is not an eigenvalue, then we fix $c_{\lambda,i} = 0$ and $d_\lambda = 0$ (see Definition 4.3). Thus, we have $< v, x_{|\lambda|} + x_{-|\lambda|} >= \sum_{i=1}^{d_{|\lambda|}} c_{|\lambda|,i}x_{|\lambda|,i} + \sum_{i=1}^{d_{-|\lambda|}} c_{-|\lambda|,i}x_{-|\lambda|,i}$, and $< v, x_{|\lambda|} - x_{-|\lambda|} >= \sum_{i=1}^{d_{|\lambda|}} c_{|\lambda|,i}x_{|\lambda|,i} - \sum_{i=1}^{d_{-|\lambda|}} c_{-|\lambda|,i}x_{-|\lambda|,i}$.
We finally obtain the following main theorem.

Theorem 4.2. *The program $P(A,v)$ is ANT on x if and only if x belongs to the set*

$$\bigcup_{(|\mu|,|\mu'|)\in|Spec(A)^*|\times|Spec(A)^*|} S^0_{|\mu|}\cap S^1_{|\mu'|}.$$

□

Proof. It is obvious that x is ANT if and only if $< v, A^{2k}(x) >$ and $< v, A^{2k+1}(x) >$ are both positive for k large. Now, reasoning as in the proof of Theorem 4.1, but using Propositions 4.3 and 4.4 instead of Proposition 4.2, we obtain that $< v, A^{2k}(x) >$ is positive for k large if and only if x belongs to $S^0_{|\mu|}$ for some $\mu \in |Spec(A)^*|$, and that $< v, A^{2k+1}(x) >$ is positive for k large if and only if x belongs to $S^1_{|\mu'|}$ for some $\mu' \in |Spec(A)^*|$. The result follows. □

The following two examples illustrate how Theorem 4.2 applies to different cases.

Example 4.2. (Running example) *Consider the program $P(A,v)$ depicted as follows:*

(i) Pseudo code:

```
while(2x+3y-z>0){
    x:=15x+18y-8z+6s-5t;
    y:=5x+3y+z-t-3s;
    z:=-4y+5z-4s-2t;
    s:=-43x-46y+17z-14s+15t;
    t:=26x+30y-12z+8s-10t;}
```

(ii) Associated matrices:

$$A=\begin{pmatrix}15&18&-8&6&-5\\5&3&1&-1&-3\\0&-4&5&-4&-2\\-43&-46&17&-14&15\\26&30&-12&8&-10\end{pmatrix},\ v=\begin{pmatrix}1\\1\\2\\3\\1\end{pmatrix}.$$

Step 1*: Diagonalization of the matrix A:*

$$P=\begin{pmatrix}2&1&-1&1&1\\-1&0&2&0&-1\\-2&0&2&-1&-2\\-4&-1&0&-2&-1\\2&2&1&2&1\end{pmatrix},\ D=\begin{pmatrix}-3&0&0&0&0\\0&-1&0&0&0\\0&0&0&0&0\\0&0&0&1&0\\0&0&0&0&2\end{pmatrix}\ \text{and}\ P^{-1}=\begin{pmatrix}-3&-3&1&-1&1\\-1&-2&1&0&1\\-5&-4&1&-1&2\\10&10&-3&2&-4\\-7&-6&1&-1&3\end{pmatrix}.$$

We obtain the following eigenvectors written using our notation: $e_{0,1} = (-1,2,2,0,1)^\top$, $e_{1,1} = (1,0,-1,-2,2)^\top$, $e_{2,1} = (1,-1,-2,-1,1)^\top$, $e_{-1,1} = (1,0,0,-1,2)^\top$ *and* $e_{-3,1} = (2,-1,-2,-4,2)^\top$.

Step 2*: Computing S_μ for all positive $\mu \in Spec(A)^*$:*

- *Our algorithm first computes the coefficients $c_{\lambda,i}$. We obtain :* $c_{0,1} = 6$, $c_{1,1} = -5$, $c_{2,1} = -6$, $c_{-1,1} = 0$ *and* $c_{-3,1} = -13$.
- *Then, our algorithm computes the coefficients of the decomposition of the initial variable values in B. They are those of the column vector $P^{-1}\cdot u$ in B_c where $u = (u_1,u_2,u_3,u_4,u_5)^\top$ is the vector encoding the initial variable values.*

$$P^{-1}.\begin{pmatrix}u_1\\u_2\\u_3\\u_4\\u_5\end{pmatrix}=\begin{pmatrix}-3u_1-3u_2+u_3-u_4+u_5\\-u_1-2u_2+u_3+u_5\\-5u_1-4u_2+u_3-u_4+2u_5\\10u_1+10u_2-3u_3+2u_4-4u_5\\-7u_1-6u_2+u_3-u_4+3u_5\end{pmatrix}=\begin{pmatrix}x_{-3,1}\\x_{-1,1}\\x_{0,1}\\x_{1,1}\\x_{2,1}\end{pmatrix}.$$

Now, the algorithm obtains all the non-zero terms appearing in the definition of $S^0_{|\lambda|}$ and $S^1_{|\lambda|}$:

$< v, x_{|1|} + x_{-|1|} >= c_{1,1}x_{1,1} = -5(10u_1 + 10u_2 - 3u_3 + 2u_4 - 4u_5)$

$< v, x_{|2|} + x_{-|2|} >= c_{2,1}x_{2,1} = -6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5)$

$< v, x_{|-3|} + x_{-|-3|} >= c_{-3,1}x_{-3,1} = -13(-3u_1 - 3u_2 + u_3 - u_4 + u_5)$
$< v, x_{|-3|} - x_{-|-3|} >= -c_{-3,1}x_{-3,1} = 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5)$
All the sets $S^0_{|\lambda|}$ and $S^1_{|\lambda|}$ can now be computed exactly:

$$\begin{aligned}
S^0_{|1|} &= \{u \in E \,|\, 5(10u_1 + 10u_2 - 3u_3 + 2u_4 - 4u_5) > 0 \;\wedge\; -6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5) = 0 \;\wedge \\
&\qquad - 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) = 0\};\\
S^1_{|1|} &= \{u \in E | 5(10u_1 + 10u_2 - 3u_3 + 2u_4 - 4u_5) > 0 \;\wedge\; -6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5) = 0 \;\wedge \\
&\qquad 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) = 0\};\\
S^0_{|2|} &= \{u \in E | \; - 6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5) > 0 \;\wedge\; -13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) = 0\};\\
S^0_{|2|} &= \{u \in E | \; - 6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5) > 0 \;\wedge\; -13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) = 0\};\\
S^1_{|2|} &= \{u \in E | \; - 6(-7u_1 - 6u_2 + u_3 - u_4 + 3u_5) > 0 \;\wedge\; 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) = 0\};\\
S^0_{|-3|} &= \{u \in E | \; - 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) > 0\};\\
S^1_{|-3|} &= \{u \in E | \; 13(-3u_1 - 3u_2 + u_3 - u_4 + u_5) > 0\}.
\end{aligned}$$

Step 3*: We apply Theorem 4.2 to generate the ANT Locus:*
The algorithm computes the following intersections: $S^0_1 \cap S^1_1$, $S^0_1 \cap S^1_2$, $S^0_1 \cap S^1_3$, $S^0_2 \cap S^1_1$, $S^0_2 \cap S^1_2$, $S^0_2 \cap S^1_3$, $S^0_3 \cap S^1_1$, $S^0_3 \cap S^1_2$ and $S^0_3 \cap S^1_1$.
In fact, the previous computational step already allows our algorithm to perform implicit simplifications. For instance, it appears that $S^0_{|1|} = S^1_{|1|}$, $S^0_{|1|} = S^1_{|1|} = S^0_{|-1|} = S^1_{|-1|}$, $S^0_{|2|} = S^1_{|2|}$ and that $S^0_{|-3|} \cap S^1_{|-3|}$ is the empty set. According to Theorem 4.2, the ANT locus reduces to the following semi-linear space:

$$\begin{aligned}
\{u = (u_1, u_2, u_3, u_4, u_5)^\top \in E \mid \;& - 10u_1 - 10u_2 + 3u_3 - 2u_4 + 4u_5 > 0 \;\wedge \\
& - 7u_1 - 6u_2 + u_3 - u_4 + 3u_5 = 0 \;\wedge \\
& - 3u_1 - 3u_2 + u_3 - u_4 + u_5 = 0\} \;\bigcup \\
\{u = (u_1, u_2, u_3, u_4, u_5)^\top \in E \mid \;& - 10u_1 - 10u_2 + 3u_3 - 2u_4 + 4u_5 > 0 \;\wedge \\
& - 3u_1 - 3u_2 + u_3 - u_4 + u_5) > 0 \;\wedge \\
& - 7u_1 - 6u_2 + u_3 - u_4 + 3u_5 = 0\} \;\bigcup \\
\{u = (u_1, u_2, u_3, u_4, u_5)^\top \in E \mid \;& 7u_1 + 6u_2 - u_3 + u_4 - 3u_5 > 0 \;\wedge \\
& - 3u_1 - 3u_2 + u_3 - u_4 + u_5 = 0\}. \quad \square
\end{aligned}$$

5 Discussions

This work is complementary to our previous work [19], which dealt with termination analysis. In [19] we first prove a sufficient condition for the termination of homogeneous linear programs. This statement was conjectured in the important work of [21], where the first attempt to prove this result contains non trivial mistakes. As we shown in [19], the actual proof of this sufficient condition requires expertise in several independent mathematical fields. Also, the necessary condition proposed in [21] does not apply as expected in practice. We then went to generalize the previous result and, to the best of our knowledge, we presented the *first necessary and sufficient condition* (NSC, for short) for the termination of linear programs. In fact, this NSC exhibits a complete decidability result for the class of linear programs on all initial values. Moreover, departing from this NSC, we showed the scalability of these approaches by demonstrating that one can directly extract a sound and complete computational method, running in polynomial time complexity, to determine termination or nontermination for linear programs. On the other hand, all other related and previous works mentioned in this paper do not provide any techniques capable of generating automatically the set of initial input variable values for which a loop does not terminate. The main contributions of this paper remain on a sound and complete computational method to compute the set of input variable values for which the programs do not terminate. The overall time complexity of our algorithm is also of order $\mathcal{O}(n^3)$. As can be seen, the main results, *i.e.*, Theorems 4.1 and 4.2, provide us with a direct symbolic representation of the ANT set. Even if those theorems are rigorously stated and proofs are quite technical, they are really easy to apply: we only need to compute the explicit terms $S^0_{|\mu|}$ and $S^1_{|\mu'|}$ in order to directly obtain a formula representing exactly and symbolically the ANT set. In a same manner, we extended this techniques to linear program not necessarily diagonalizable and we obtained similar theoretical and practical results. As their associated proofs are more technical, they would required more space to be fully expressed and we left them for an ensuing report. In other more recent work on termination static analysis for programs over the rationals or the integers with several conditional linear inequalities, we also show that the notion ANT remains central.

6 Conclusion

We presented the new notion of *asymptotically non-terminant initial variable values* for linear programs. Considering a linear diagonalizable program, our theoretical results provided us with sound, complete and fast computational methods allowing the automated generation of the sets of all asymptotically non-terminant initial variable values, represented symbolically and exactly by a semi-linear space, e.g., conjunctions and disjunctions of linear equalities and inequalities. Also, by taking the complementary set of the semi-linear set of ANT initial variable values, we obtain a precise under-approximation of the set of terminant initial variable values for the (non -terminant) program. Actually, this type of method can be vastly generalized, to tackle the termination and non-termination problem of linear programs not necessarily diagonalizable, with more than one conditional linear inequality, on rational or integer initial values, for instance. We leave this investigation for an ensuing report.

References

[1] Amir M. Ben-Amram and Samir Genaim. On the linear ranking problem for integer linear-constraint loops. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '13, pages 51–62, New York, NY, USA, 2013. ACM.

[2] Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud. On the termination of integer loops. In *VMCAI*, pages 72–87, 2012.

[3] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Linear ranking with reachability. In *In CAV*, pages 491–504. Springer, 2005.

[4] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination analysis of integer linear loops. In *In CONCUR*, pages 488–502. Springer-Verlag, 2005.

[5] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination of polynomial programs. In *In VMCAI'2005: Verification, Model Checking, and Abstract Interpretation, volume 3385 of LNCS*, pages 113–129. Springer, 2005.

[6] Mark Braverman. Termination of integer linear programs. In *In Proc. CAV06, LNCS 4144*, pages 372–385. Springer, 2006.

[7] Hong Yi Chen, Shaked Flur, and Supratik Mukhopadhyay. Termination proofs for linear simple loops. In *Proceedings of the 19th international conference on Static Analysis*, SAS'12, pages 422–438, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking.* MIT Press, Cambridge, MA, 2000.

[9] Michael Colón and Henny Sipma. Synthesis of linear ranking functions. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 67–81, London, UK, 2001. Springer-Verlag.

[10] Michael A. Colón and Henny B. Sipma. Practical methods for proving program termination. In *In CAV2002: Computer Aided Verification, volume 2404 of LNCS*, pages 442–454. Springer, 2002.

[11] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. *SIGPLAN Not.*, 41(6):415–426, June 2006.

[12] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.

[13] Patrick Cousot and Radhia Cousot. An abstract interpretation framework for termination. *SIGPLAN Not.*, 47(1):245–258, January 2012.

[14] Dennis Dams, Rob Gerth, and Orna Grumberg. A heuristic for the automatic generation of ranking functions. In *Workshop on Advances in Verification*, pages 1–8, 2000.

[15] Zohar Manna. *Mathematical Theory of Computation.* McGraw-Hill, 1974.

[16] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, pages 239–251, 2004.

[17] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th International Symposium in Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[18] Rachid Rebiha, Nadir Matringe, and Arnaldo V. Moura. A complete approach for termination analysis of linear programs. Technical Report IC-13-08, Institute of Computing, University of Campinas, February 2013.

[19] Rachid Rebiha, Nadir Matringe, and Arnaldo V. Moura. Necessary and sufficient condition for termination of linear programs. Technical Report IC-13-07, Institute of Computing, University of Campinas, February 2013.

[20] Henny B. Sipma, Tomás E. Uribe, and Zohar Manna. Deductive model checking. *Form. Methods Syst. Des.*, 15(1):49–74, July 1999.

[21] Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA*, volume 3114 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 2004.