Report on
Geometric Nontermination Arguments

Seminar: Verification Techniques

Lars Helge Scheel
Supervision: Jera Hensel

WS 2016

Abstract

In the paper [2] Leike and Heizmann present a new nontermination argument called *Geometric Nontermination Argument* (GNA). Termination is not decidable in general, but it can be decided for sufficiently restricted programs. In this case, we severely restrict ourselves to linear `while`-loops without any control constructs other than the guard. The idea is to analyze larger programs that contain subprograms restricted in this way. Given a deterministic linear loop program, a GNA can be found using an SMT solver. GNAs define pointwise geometric sums that yield infinite execution sequences. This is an advantage because the programmer might be interested in the reason for nontermination.

1 Introduction

There are a variety of approaches to termination analysis. Of course, for Turing complete programming languages, any approach can not be sound and complete. Since soundness is more important, we are left with incomplete approaches. Leike and Heizmann point towards linear ranking functions [3], which are termination arguments, and linear recurrence sets [1], which are nontermination arguments. In both cases, the existence is decidable and sufficient to prove termination and nontermination respectively. However, the existence is not always implied, and thus the approaches are incomplete.

The subset of programs, chosen in order to obtain soundness and completeness for the GNAs, is called the set of linear lasso programs. A linear lasso program consists of a single line of code, followed by a `while`-loop. These two parts are called STEM and LOOP. The STEM allows for variables to be set and the LOOP allows for linear updates of the variables. These linear updates give rise to an update matrix, and a further restriction is that this matrix may only have real, non-negative eigenvalues. In Figure 1, we can see an example lasso program.

In STEM, b is set to 1. Notice that since a is not set, in the analysis all possible values for a are considered. LOOP is guarded by $a + b \geq 4$, thereby restricting $a \geq 3$ for the first iteration. Also notice that the variable updates in the loop are linear and deterministic. Assume $a = 3$,

$b \leftarrow 1$
while $a + b \geq 4$ **do**
  $a \leftarrow 3 \cdot a + b$
  $b \leftarrow 2 \cdot b$
end while

Figure 1: Example linear lasso program.

then executing the program yields the following infinite sequence:

$$\overbrace{\begin{pmatrix}3\\ undefined\end{pmatrix}, \underbrace{\begin{pmatrix}3\\1\end{pmatrix}}_{\text{STEM}}, \begin{pmatrix}10\\2\end{pmatrix}, \begin{pmatrix}32\\4\end{pmatrix}, \begin{pmatrix}100\\8\end{pmatrix}, \dots}^{\text{LOOP}}$$

We represent the intermediate program states as vectors over $\mathbb{R}$. The LOOP part of the infinite sequence can also be described by the following pointwise sum of geometric series. Increase k for subsequent states:

$$\underbrace{\begin{pmatrix}3\\1\end{pmatrix}}_{\text{starting point}} + \sum_{i=0}^{k} \underbrace{\begin{pmatrix}4 & 3\\0 & 1\end{pmatrix}}_{\text{direction}} \cdot \underbrace{\begin{pmatrix}3 & 1\\0 & 2\end{pmatrix}^{i}}_{\text{speed}} \cdot \begin{pmatrix}1\\1\end{pmatrix}$$

The starting point is intuitive as it is. However, it is not intuitively clear how to compose the sum. Since the variables grow exponentially, there must be some factors in the sum that grow accordingly. This is the purpose of the speed matrix, we see that it contains the factors from the linear loop update. The direction matrix is actually a basis, enabling us to separate the speed part. So we can fold it and get the correct, ever-changing distance to the last state.

2 Preliminaries

In order to define the restricted loop programs, we first need to define some notation. We will use boldly printed $\mathbf{0}, \mathbf{1}$ to refer to a vector of zeros and ones respectively and I is the identity matrix. The dimension results from the context. We are going to use $Ax \leq b$ as a notation for a conjunction of linear constraints, where $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$:

$$Ax \leq b := \bigwedge_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j \leq b_i.$$

Note that this does not imply that x has to be bounded. Now we are ready to define linear lasso programs, which are the focus of our nontermination analysis. We first define a potentially nondeterministic version, that is somewhat easier to handle for later proofs. The completeness result only applies to deterministic lasso programs.

Definition 2.1 (Linear Lasso Programs L = (STEM,LOOP))
A linear lasso program is a `while`*-loop with linear updates, preceded by an optional line of code. If this line is omitted, the program is also a linear loop program. The two parts are called stem and loop and they are defined by binary relations over the $\mathbb{R}$-vector space. The vectors $x, x' \in \mathbb{R}^n$ correspond to the n variables in two consecutive states.*
We define $(x, x') \in STEM$ if the variables in x' are set according to the line of code in the stem. For the loop, we use a matrix and the notation above to model not only the guard but also the update:

$$(x, x') \in LOOP \Leftrightarrow A \begin{pmatrix}x\\x'\end{pmatrix} \leq b$$

for some $A \in \mathbb{R}^{m \times 2n}$ and some $b \in \mathbb{R}^m$. Note that this implies linearity.

Restricting to determinism, we get an important subset: deterministic linear lasso programs.

Definition 2.2 (Deterministic Linear Lasso Programs L = (STEM,LOOP))
A deterministic linear lasso program is similar to the general version. But for any $(x, x') \in$ *LOOP it must hold that*

$$Gx \leq g \quad \wedge \quad Mx + m = x'.$$

Every deterministic linear lasso program is also a linear lasso program with identical STEM, and the matrix and vector for LOOP given as follows:

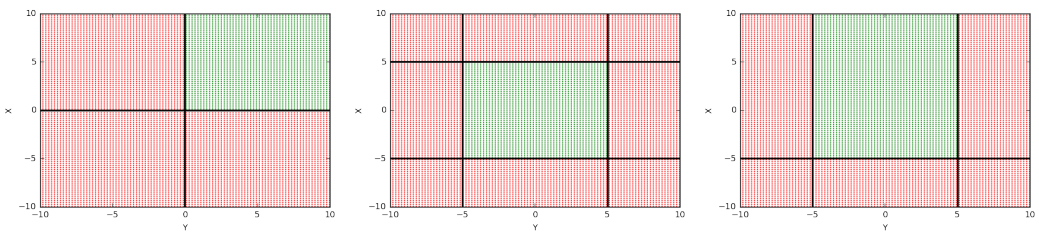
$$A = \begin{pmatrix} G & 0 \\ M & -I \\ -M & I \end{pmatrix}, \quad b = \begin{pmatrix} g \\ -m \\ m \end{pmatrix}$$

We can see that in this definition, the actual update matrix M *is used. And since the last two rows of* A *are the same except for their different signs, equality is enforced. This leads to a single possible subsequent state.*

Drawing attention to the `while` guard, in general not all points in $\mathbb{R}^n$ satisfy it, otherwise nontermination is guaranteed. Those points that satisfy $Gx > g$ must be excluded. As we have seen, this is a notation for a conjunction of linear constraints. Assuming the guard consists of a single restriction, then G may only have one row. In this case we have a simple dot product, any point satisfying the constraint lies in one half of the space. If $g = \mathbf{0}$, the separating (hyper)plane cuts the space at the origin. Multiple restrictions result in polyhedra, defined below.

Definition 2.3 (Convex Cone: $\{\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}} \mid \boldsymbol{Ax} \leq \mathbf{0}\}$, $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{m} \times \boldsymbol{n}}$)
A set of points defined like this is called a cone; it can be understood as a conjunction of half-spaces, where the separating hyperplanes include the origin. It can also be interpreted as the set of possible convex combinations of a finite set of generating vectors. We can see an example cone in Figure 2a. It corresponds to $A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$.

Definition 2.4 (Convex Polyhedron: $\{\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}} \mid \boldsymbol{Ax} \leq \boldsymbol{b}\}$, $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{m} \times \boldsymbol{n}}, \boldsymbol{b} \in \mathbb{R}^{\boldsymbol{m}}$)
A set of points defined like this is called a polyhedron; like the cone, it can be understood as a conjunction of half-spaces, but in this case the separating hyperplanes can have an offset to the origin. This enables us to define a square or cube, for example, but not all polyhedrons are bounded. We can see an example polyhedron in Figure 2b. It corresponds to $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}$, $b = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}$.



(a) Cone (b) Polytope (c) Polyhedron

Figure 2: Example polyhedrons in 2D.

These two definitions are similar and in fact, we can decompose polyhedra by the decomposition theorem for polyhedra [4, Cor. 7.1b]; the unbounded part of a polyhedron is a single cone and the rest is just a plain polytope (generalized polygon). So we can assume that any execution of linear lasso programs takes place in a polytope and cone defined by the guard. We now come to infinite executions.

Definition 2.5 (Nontermination)
A lasso program L = (STEM, LOOP) is called nonterminating if there exists an infinite execution sequence:

$$x_0, x_1, x_2, x_3, \cdots$$

where $(x_0, x_1) \in STEM$ *and* $(x_t, x_{t+1}) \in LOOP$ *holds for any* $t \geq 1$.

The infinite execution can be of diverse nature, it may be repeating itself or grow at different speeds. Later we will use a lemma to take advantage of the first case, as there are fixed points in the execution. In the completeness proof, we need to change the basis of our computation in order to show the existence of a GNA. In the first step we are going to use the jordan normal form, which already results in a matrix that is close to what we need for the GNA. We are aiming at a near-diagonal form.

In this context the terms eigenvalue and eigenvector, among others, are used and assumed to be known. Regular matrices are diagonizable. But if the matrix contains at least one defective eigenvalue, that is an eigenvalue whose geometric multiplicity is strictly smaller than the algebraic multiplicity, then the matrix is not diagonizable. We must use generalized eigenvectors to get to the Jordan normal form.

Definition 2.6 (Jordan Normal Form)
Given a matrix $M \in \mathbb{R}^{n \times n}$ *with real eigenvalues has a Jordan normal form (JNF), it is generally ambiguous. The JNF is a block diagonal matrix, in total it is a diagonal matrix with one superdiagonal containing ones and zeros. The blocks are called Jordan blocks and can be identified by the ones on the superdiagonal. They refer to an eigenvalue, the number of blocks corresponding to the same eigenvalue is equal to the dimension of the eigenspace of that eigenvalue. The size of a block corresponds to the difference between the geometric and algebraic multiplicity.*

$$J = \begin{pmatrix} J_1(\lambda_1) & & & \\ & J_2(\lambda_1) & & \\ & & J_1(\lambda_2) & \\ & & & J_1(\lambda_3) \end{pmatrix}, \quad J_i(\lambda) = \begin{pmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{pmatrix}$$

Here, J is an example for a total JNF, and $J_i(\lambda)$ *an example for a Jordan block corresponding to the defective eigenvalue* λ.

Since we restrict our programs so that the update matrix only has real eigenvalues, the following theorem is useful.

Theorem 2.1 (Jordan Normal Form)
For each real square matrix $M \in \mathbb{R}^{n \times n}$ *with real eigenvalues, there is an invertable real matrix* $V \in \mathbb{R}^{n \times n}$ *and JNF , so that*

$$M = VJV^{-1}$$

With these preliminaries out of the way, we can finally define GNAs.

3 Geometric Nontermination Arguments

As already mentioned, the infinite execution of lasso programs can be expressed as a pointwise sum of geometric series. GNAs do not directly consist of this expression, but rather a set of vectors and factors that can be used to construct the sum. During this section, when referring to a program, we fall back to the definition of a potentially nonderministic linear lasso program L = (STEM, LOOP) and the notations that come with it. We are going to prove the claims regarding soundness as well as completeness of the approach for the restricted class of programs. But first, we have to define the nontermination argument.

Definition 3.1 (Geometric Nontermination Argument)
A geometric nontermination argument is a tuple of the following form:

$$(x_0, x_1, y_1, \ldots, y_k, \lambda_1, \ldots, \lambda_k, \mu_1, \ldots, \mu_{k-1}).$$

It contains vectors as well as real factors. We call k the size of the GNA. Given L = (STEM, LOOP) with n variables, a corresponding GNA satisfies the following formulas:

$$\begin{aligned}
(\textit{domain}) \quad & x_0, x_1, y_1, \ldots, y_k \in \mathbb{R}^n,\ \lambda_1, \ldots, \lambda_k, \mu_1, \ldots, \mu_{k-1} \geq 0, \\
(\textit{init}) \quad & (x_0, x_1) \in \textit{STEM}, \\
(\textit{point}) \quad & A \begin{pmatrix} x_1 \\ x_1 + \sum_i y_i \end{pmatrix} \leq b, \\
(\textit{ray}) \quad & A \begin{pmatrix} y_i \\ \lambda_i y_i + \mu_{i-1} y_{i-1} \end{pmatrix} \leq \boldsymbol{0} \textit{ for all } 1 \leq i \leq k.
\end{aligned}$$

We use $y_0 = \boldsymbol{0}$ and $\mu_0 = 0$ for convenience.

One possible GNA corresponding to the example program is:

$$\left(\begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, 3, 1, 1 \right).$$

As mentioned, given a program, one can use a SMT solver to find any existing tuple that fulfills the formulas. The lambdas correspond to the eigenvalues of the update matrix. Looking back to the example program, we get the update matrix $\begin{pmatrix} 3 & 1 \\ 0 & 2 \end{pmatrix}$ which obviously has the eigenvector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ with eigenvalue 3. So in the case of an deterministic program, we can compute the eigenvalues and therefore reduce the formulas to linear constraints. Depending on the implementation of the SMT solver, they can be solved efficiently. The most common approach would be to break the constraints down and build a boolean skeleton connecting the smaller parts. A collaboration with a SAT solver follows; the SAT solver assigns truth values to the parts and an arithmetic backend checks whether the assignments are consistent. After some number of iterations a satisfying assignment will be found or its existence will be disproven.

3.1 Soundness

The GNA has tuple form by default. In the following we will explore how to construct a pointwise sum of geometric series from the tuple. During the next section, which also contains the soundness proof, we are going to explore how to construct the pointwise geometric sum from a GNA.

Proposition 3.1 (Soundness)
If a linear lasso program L has a GNA, then L is nonterminating.

Proof 3.1
To prove this proposition, we have to construct a pointwise sum of geometric series that yields an infinite number of subsequent states, i.e. an infinite execution. So assume there is an GNA, define two matrices

$$Y := (y_1, \ldots, y_k), \;\; U := diag(\lambda_1, \ldots, \lambda_k) + superdiag(\mu_1, \ldots, \mu_{k-1}),$$

where U has a form similar to JNF. Notice that Y might not be square if $k < n$ but U is square. We can now use Y as the direction matrix and U as the speed matrix. But first, the STEM part of the execution is already guaranteed by (init). $(x_t)_{t\geq 2}$ can then be calculated in the following way:

$$\begin{aligned} x_t &= x_1 + Y\mathbf{1} + YU\mathbf{1} + YU^2\mathbf{1} + \cdots + YU^{t-2}\mathbf{1} \\ &= x_1 + \sum_{i=0}^{t-2} YU^i\mathbf{1}. \end{aligned}$$

Now we have to prove that $(x_t, x_{t+1}) \in LOOP$. To achieve this, we first construct a vector of the form $\begin{pmatrix} x \\ x' \end{pmatrix}$ that satisfies $A\begin{pmatrix} x \\ x' \end{pmatrix} \leq b$, i.e. is in LOOP. Second we show that the constructed vector is equal to $\begin{pmatrix} x_t \\ x_{t+1} \end{pmatrix}$.

$$(domain) \Rightarrow \sum_{i=0}^{t-2} U^i\mathbf{1} = Z\mathbf{1} \text{ only has nonnegative entries.}$$

$$\Rightarrow YZ\mathbf{1} = \sum_i a_i y_i \text{ for } \underline{\text{some}} \text{ nonnegative } a_i$$

Note that a formula for a_i is not trivial. But we can safely assume that it is nonnegative. We can now multiply (ray) with a_i for each i, since the different dot products yield nonpositive scalars.

$$(ray) \Rightarrow A\begin{pmatrix} a_i y_i \\ a_i\lambda_i y_i + a_i\mu_{i-1}y_{i-1} \end{pmatrix} \leq \mathbf{0} \Rightarrow A\begin{pmatrix} \sum_i a_i y_i \\ \sum_i a_i\lambda_i y_i + a_i\mu_{i-1}y_{i-1} \end{pmatrix} \leq \mathbf{0}$$

Finally, we add (point) to get:

$$(point) \Rightarrow A\begin{pmatrix} x_1 + \sum_i a_i y_i \\ x_1 + \sum_i y_i + \sum_i a_i\lambda_i y_i + a_i\mu_{i-1}y_{i-1} \end{pmatrix} \leq b$$

This concludes the first part.
By our definition we get:

$$x_1 + \sum_i a_i y_i = x_1 + YZ\mathbf{1} = x_1 + Y\sum_{i=0}^{t-2} U^i\mathbf{1} = x_1 + \sum_{i=0}^{t-2} YU^i\mathbf{1} = x_t$$

$$\begin{aligned} x_1 + \sum_i y_i + \sum_i a_i\lambda_i y_i + a_i\mu_{i-1}y_{i-1} &= x_1 + Y\mathbf{1} + \sum_{i=0}^{t-2} a_i YUe_i \\ &= x_1 + Y\mathbf{1} + \sum_{i=1}^{t-1} YU^i\mathbf{1} = x_1 + \sum_{i=0}^{t-1} YU^i\mathbf{1} = x_{t+1} \end{aligned}$$

So, every pair in our infinite sequence satisfies LOOP.

3.2 Completeness

Having proved soundness, it remains to prove completeness. In order to do this, we need a series of lemmata. The first one states that executions in a polytope must level off at a certain fixed point eventually. A fixed point $x^\star$ is a point that can be chained infinitely often: $(x^\star, x^\star) \in \text{LOOP}$. A bounded infinite execution is $x_0, x_1, x_2, \ldots$ where every state is bounded $|x_t| \leq d$ for some d.

Lemma 3.1 (Fixed Point)
Given a program L = (true, LOOP), i.e. a linear loop program, L has a bounded infinite execution iff L has a fixed point.

Proof 3.2
If L has an fixed point $x^\star$, then L has the infinite bounded execution $x^\star, x^\star, \ldots$. On the other hand, assume that L has an infinite bounded execution $(x_t)_{t\geq 0}$. Let $z_t := \frac{1}{k}\sum_{k=1}^{t} x_k$. Then $(z_t)_{t\geq 1}$ is a Cauchy sequence, and the point of convergence $z^\star$ is the desired fixed point. We know that the polyhedron Q bounding the execution contains all subsequent states $\begin{pmatrix} x_t \\ x_{t+1} \end{pmatrix}$ from the infinite execution,

$$Q := \{ \begin{pmatrix} x \\ x' \end{pmatrix} \mid A \begin{pmatrix} x \\ x' \end{pmatrix} \leq b \}.$$

Since it is convex, it also contains $\frac{1}{t}\sum_{k=1}^{t} \begin{pmatrix} x_k \\ x_{k+1} \end{pmatrix}$. Using a little trickery, we can derive that it also contains $\begin{pmatrix} z^\star \\ z^\star \end{pmatrix}$:

$$\begin{pmatrix} z_t \\ \frac{t+1}{t} z_{t+1} \end{pmatrix} = \frac{1}{t}\begin{pmatrix} \boldsymbol{0} \\ x_1 \end{pmatrix} + \frac{1}{t}\sum_{k=1}^{t} \begin{pmatrix} x_k \\ x_{k+1} \end{pmatrix} \Rightarrow \left(\begin{pmatrix} z_t \\ \frac{t+1}{t} z_{t+1} \end{pmatrix} - \frac{1}{t}\begin{pmatrix} \boldsymbol{0} \\ x_1 \end{pmatrix} \right) \in Q$$

With $t \to \infty$, we get that $z^\star$ is indeed a fixed point of the execution.

Corollary 3.1 (Bounded Infinite Executions)
For a linear loop program, any bounded infinite execution is captured by a GNA of size 0.

Proof 3.3
Just give the fixed point as x_0, x_1 and zero update. The resulting execution repeats the fixed point and is valid.

In the context of the previous lemma and corollary, it is imperative that LOOP is a topologically closed set. In other words, we may not use strict inequalities in the `while` guard. Otherwise we could have bounded infinite executions that come arbitrary close to a fixed point, but it is excluded by the guard. Also, we consider loop programs instead of lasso programs, because there might be a bounded infinite execution but the STEM does not allow for the fixed point to be reached. In order to use the results, we need to be able to separate the space of execution into a part where the execution has a fixed point and another part where it does not. We can then combine the GNAs resulting from the subspaces.

Lemma 3.2 (Loop Disassembly)
Given a nonterminating linear loop program with n variables over $\mathbb{R}^n$, analyze the execution projected to the subspaces $U \oplus V = \mathbb{R}^n$. If the execution is bounded in one of the

subspaces, say U, then there exists a fixed point x^U by the fixed point lemma. Assume there is a GNA of size k for the execution projected to V. Then we can construct a GNA of size k for the complete space $\mathbb{R}^n$.

Proof 3.4
The final GNA is constructed by adding the fixed point to the subspace: $V + x^U$. It is clear that the fixed point does not pose a problem for the infinite execution. For the rest, we assume a GNA of size k for the space V.

Finally we prove the completeness of the approach for deterministic linear loops with real nonnegative eigenvalues.

Theorem 3.1 (Completeness)
Given a nonterminating linear loop program with n variables of the form

while $Gx \leq g$ ***do***
  $x \leftarrow Mx + m$
end while

where M has nonnegative real eigenvalues, then there exists a GNA of size $k \leq n$.

Proof 3.5
The first step of proving this theorem is to exclude the linear subspaces that are irrelevant. By this, all subspaces, in which the infinite execution is bounded and therefore must have a fixed point and GNA of size zero, are meant.

We start by excluding the linear subspace that is orthogonal to the subspace containing the guard cone: $\mathcal{Y} = \{y \in \mathbb{R}^n \mid Gy \leq 0\}$. This is possible due to the decomposition theorem of the guard polyhedron, and we are only interested in the unbounded part, i.e. the cone. Another subspace that we can exclude is the eigenspace for eigenvalues that are smaller than one. This is possible since the projected execution is a contraction and therefore is bounded. Assume there is a fixed point for those excluded subspaces, then we can focus the rest of the proof on the remaining orthogonal space.

Part 1:

In this part, we use the existence of the JNF and two basis transformations to get the basis $y_1, \ldots, y_k$. In this basis, M turns into the desired speed matrix U.

Let $v_1, \ldots, v_n$ be a basis so that M has JNF with descending eigenvalues. We observe that we can fold the jordan blocks independently and that the rightmost column dominates each block eventually. Since larger eigenvalues come first and the execution is infinite, the basis vector v_i corresponding to the rightmost column must be included in the guard cone projected to $span(v_i, \ldots, v_n)$[1]. We can now iterate through the blocks and add the part that is in the generalized eigenspace of the previous block: $w_i = v_i + u_i$ with $u_i \in span(v_1, \ldots, v_{i-1})$, so that w_i is in the (unprojected) guard cone. The resulting w_i form a new basis. Note that $u_1 = \boldsymbol{0}$ and that this basis is smaller, in general $k \leq n$.

Viewing M in this new basis yields T, which has upper triangular form. To see this, consider $Mw_i = \lambda_i w_i + u_i$. By construction, w_i is an eigenvector of $span(v_i, \ldots, v_n)$ and $u_i \in span(v_1, \ldots, v_{i-1}) = span(w_1, \ldots, w_{i-1})$. Since we treated each block in turn, we only may have ones in the superdiagonal and zeros otherwise.

Now we use positive linear combinations of the w-basis and scaling to construct a basis $y_1, \ldots, y_k$ that satisfies

$$My_i = \lambda_i y_i + \mu_{i-1} y_{i-1}. \quad (1)$$

[1]It can be either v_i or the negative version $-v_i$

$$y_i = \beta_i \sum_{j=1}^{i} \alpha_{ij} w_j$$

With $\alpha_{ij} \geq 0, \alpha_{ii} > 0, \beta_i > 0$, as we will see later, we can set these factors in an order that prevents circular dependencies. Define $W = (w_1, \ldots, w_k)$ and $Y = (y_1, \ldots, y_k) = diag(\beta)\alpha W$. The matrix α is a lower triangular matrix with nonzero diagonal and therefore invertible, implying that Y is invertible. Since the w_i are in the guard cone and the guard cone is convex, it follows that the y_i are also in the guard cone:

$$GY \leq 0.$$

Now we have to choose α. First, split T into a diagonal and a nil-potent matrix: $T = D + N$. We have to ensure that (1) holds for all i. Write (1) in the basis of W:

$$(D+N)\beta_i \sum_{j \leq i} \alpha_{ij} e_j \stackrel{!}{=} \lambda_i \beta_i \sum_{j \leq i} \alpha_{ij} e_j + \mu_{i-1}\beta_{i-1} \sum_{j<i} \alpha_{ij} e_j$$

$$\Leftrightarrow \sum_{j \leq i} (\lambda_j - \lambda_i)\alpha_{ij} e_j + N \sum_{j \leq i} \alpha_{ij} e_j - \mu_{i-1} \frac{\beta_{i-1}}{\beta_i} \sum_{j<i} \alpha_{i-1j} e_j \stackrel{!}{=} \boldsymbol{0}.$$

note that this is not directly dependent on β, we can choose μ_{i-1} accordingly and leave $\beta > 0$ undefined for now.

We consider the two cases of the eigenvalues being unequal and equal:

$\lambda_i \neq \lambda_j$
$\lambda_j - \lambda_i > 0$, *due to the ordering of the eigenvalues. In this case, we can increase the subdiagonal entries in the ith column of α to increase the total expression and increase the entries in the $i-1$th column to decrease the total expression.*

$\lambda_i = \lambda_j$
In this case, we can make use of the structure of T: only the superdiagonal of N contains some ones. So it suffices to show:

$$\sum_{l<j \leq i} \alpha_{ij} T_{j-1j} e_{j-1} - \mu_{i-1} \sum_{l \leq j<i} \alpha_{i-1j} e_j \stackrel{!}{=} 0.$$

This can be a problem if $T_{j-1j} = 0$. But then, we can replace the corresponding basis vector y_i with $y_i + y_j$ without reducing the expressiveness of the GNA, since they have the same eigenvalue.

Part 2:

In this second part, we construct a GNA from Y and U and confirm that it meets the requirements.

L is nonterminating, so there must be some x inside the guard polyhedron. Let $x_1 = x + Y\gamma$ with $\gamma \geq 0$. We take the columns of Y and the entries from U and build a GNA. From the construction, we can directly derive (domain) and (ray). (init) is irrelevant since we consider loop programs as opposed to lasso programs.

It remains to show (point). The guard part is satisfied due to the definition of x_1:

$$Gx_1 = Gx + GY\gamma \leq g$$

The update part yields the following equation:

$$\begin{aligned} Mx_1 + m &= x_1 + Y\mathbf{1} \\ (M-I)(x+Y\gamma) + m &= Y\mathbf{1} \\ (M-I)x + m &= Y\mathbf{1} - (M-I)Y\gamma \\ Y^{-1}(M-I)x + Y^{-1}m &= \mathbf{1} - Y^{-1}(M-I)Y\gamma \\ (U-I)Y^{-1}x + Y^{-1}m &= \mathbf{1} - (U-I)\gamma \\ (U-I)Y^{-1}x + Y^{-1}m &= \mathbf{1} - (U-I)\gamma \end{aligned}$$

Now it remains to show that β, γ can be chosen such that the equation is satisfied. This is not trivial, but possible. For details refer to the paper [2].

4 Conclusion

The geometric nontermination argument is a finite representation of an infinite execution of a linear lasso program. We have proven soundness and, for special restrictions, we even have been able to prove completeness. While this restricted class of programs migh be considered useless on its own, it does appear useful in larger programs. With the nontermination argument, we are able to analyze these programs in an automated way that does not only give us an answer to the question of whether it terminates or not, but also an explicit way to reconstruct the infinite execution. In combination with other software that can find lasso programs in larger programs, this becomes a powerful tool in termination analysis.

References

[1] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R.-G. Xu. Proving non-termination. *ACM Sigplan Notices*, 43(1):147–158, 2008.

[2] J. Leike and M. Heizmann. Geometric nontermination arguments. *CoRR*, abs/1609.05207, 2016.

[3] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 239–251. Springer, 2004.

[4] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.