The Polyranking Principle

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma *

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{arbrad,zm,sipma}@theory.stanford.edu

Abstract. Although every terminating loop has a ranking function, not every loop has a ranking function of a restricted form, such as a lexicographic tuple of polynomials over program variables. The *polyranking principle* is proposed as a generalization of polynomial ranking for analyzing termination of loops. We define *lexicographic polyranking functions* in the context of loops with parallel transitions consisting of polynomial assertions, including inequalities, over primed and unprimed variables. Next, we address *synthesis* of these functions with a complete and automatic method for synthesizing *lexicographic linear polyranking functions* with supporting linear invariants over linear loops.

1 Introduction

Guaranteed termination of program loops is necessary in many settings, such as embedded systems and safety critical software. Additionally, proving general temporal properties of infinite state programs requires termination proofs, for which automatic methods are welcome [13, 9, 11]. We propose a termination analysis based on lexicographic polynomial *polyranking* functions, which subsume lexicographic polynomial *ranking* functions.

Although every terminating loop has a ranking function, not every terminating loop has a ranking function of a restricted form, such as a lexicographic tuple of polynomials over program variables. In [2], we present a method for identifying bounded expressions that are eventually negative over loops with parallel transitions expressed as simultaneous assignments of polynomial expressions to variables. For showing termination, a function that *eventually* ranks is as good as a function that always ranks. That transitions are only assignments allows an efficient analysis based on *finite difference expressions* with respect to transitions. In this paper, we generalize our approach to loops with assertional transition relations, rather than just assignments. Including assertional transition relations in a loop abstraction language makes it more widely applicable for

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939. The first author was additionally supported by a Sang Samuel Wang Stanford Graduate Fellowship.

modeling infinite state systems. The main idea for analyzing these loops is to identify eventually decreasing expressions that upper bound differences.

In [1], we show how to synthesize lexicographic linear ranking functions with supporting linear invariants over loops with linear assertional transition relations. We extend the machinery in this paper to synthesize lexicographic linear polyranking functions with supporting linear invariants over linear loops. This extension requires solving complex structural constraints induced by the recursive definition of polyranking functions. The synthesis method solves efficiently a combination of structural and numerical constraints. It searches a space of *partial* structures, solving induced constraint systems to guide the search. These partial structures represent sets of complete structures so that an infeasible induced constraint system excludes entire sets. For our application, the induced constraint systems are numerical. We believe that the general strategy is applicable to other problem domains.

Termination of loops has received a fair amount of attention recently. Synthesis of linear ranking functions over linear loops with multiple paths and assertional transition relations is accomplished via polyhedral manipulation in [4, 5]. In [10], Podelski and Rybalchenko specialize the technique to single-path linear loops without an initial condition, providing an efficient and complete synthesis method for linear ranking functions based on linear programming. We generalize these results in [1]. Cousot shows how polynomial ranking functions can be synthesized over nonlinear loops using nonlinear convex optimization [6]. Tiwari proves that the termination of a class of single-path loops with linear guards and assignments is decidable [14]. Related efforts on *verification diagrams* [9] and *transition invariants* [11] (see also [7, 8, 3]) use automatic termination analysis. These frameworks either abstract the state-space or the transition relation to isolate a finite number of different forms of infinite behavior. Termination analysis can then find the reason for termination of each isolated behavior.

The rest of the paper is organized as follows. Section 2 introduces our loop abstraction and basic concepts. Section 3 describes the polyranking principle. Sections 4 and 5 describe our synthesis technique. Section 6 concludes.

2 Preliminaries

We define our abstraction of loops, present several basic concepts, and recall Farkas's Lemma, a mathematical result that we employ in synthesis.

Definition 1 (Polynomial Assertion) A *real variable* is a variable x that ranges over the reals, $\mathbb{R}$. A *polynomial term* has the form $c\prod_i x_i^{d_i}$ for constant $c \in \mathbb{R}$, real variables x_i, and degrees $d_i \in \mathbb{N}$ ranging over the nonnegative integers. A *polynomial expression* P is the sum of polynomial terms. A *polynomial atom* is the comparison $P_1 \bowtie P_2$ of two polynomial expressions, for $\bowtie \in \{<, \leq, =, \geq, >\}$. A *polynomial assertion* is the conjunction of polynomial atoms.

Definition 2 (Polynomial Loop) A *polynomial loop* $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ consists of variables $\mathcal{V}$, *initial condition* θ, and set of *transitions* $\mathcal{T}$. $\theta(\mathcal{V})$ is a polynomial assertion over $\mathcal{V}$ expressing what is true before entering the loop. A transition $\tau(\mathcal{V}, \mathcal{V}')$ over variables $\mathcal{V} \cup \mathcal{V}'$ is a polynomial assertion over unprimed and primed variables, where a primed variable x' represents the next-state value of x.

Definition 3 (Linear Loop) A *linear loop* is a polynomial loop in which all assertions and expressions are affine.

When discussing affine expressions, assertions, and loops, we use the notation $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq 0$ to mean a linear atom, where $\mathbf{x} = (x_1, x_2, \ldots, x_m, 1)^{\mathrm{T}}$ is a homogenized vector corresponding to the variables $\mathcal{V}$. A linear assertion is then $A\mathbf{x} \geq 0$, for homogenized matrix A. By $\tau(\mathbf{x}\mathbf{x}') \geq 0$, we mean the linear assertion corresponding to τ's transition relation, where $(\mathbf{x}\mathbf{x}')$ stands for the homogenized vector corresponding to $\mathcal{V} \cup \mathcal{V}'$: $(\mathbf{x}\mathbf{x}') = (x_1, x_2, \ldots, x_m, x'_1, x'_2, \ldots, x'_m, 1)^{\mathrm{T}}$.

Definition 4 (Loop Satisfaction) A loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ satisfies an assertion φ, written $L \models \varphi$, if φ holds in all reachable states of L.

Definition 5 (Infinitely Often) For infinite computation σ of loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$, $io(\sigma) \subseteq \mathcal{T}$ is the set of transitions that occur *infinitely often* in the computation.

Definition 6 (Lexicographic Polynomial Ranking Function) A *lexicographic polynomial ranking function* for a loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ is an n-tuple of polynomial expressions $\langle r_1(\mathcal{V}), \ \ldots, \ r_n(\mathcal{V}) \rangle$ such that for some $\epsilon > 0$ and for each $\tau \in \mathcal{T}$, for some $i \in \{1, \ldots, n\}$,

(Bounded) $L \models \tau(\mathcal{V}, \mathcal{V}') \rightarrow r_i(\mathcal{V}) \geq 0$,
(Ranking) $L \models \tau(\mathcal{V}, \mathcal{V}') \rightarrow r_i(\mathcal{V}') - r_i(\mathcal{V}) \leq -\epsilon$;
(Unaffecting) for $j < i$, $L \models \tau(\mathcal{V}, \mathcal{V}') \rightarrow r_j(\mathcal{V}') - r_j(\mathcal{V}) \leq 0$.

If $n = 1$, then the result is simply a polynomial ranking function.

For some loops, invariants are necessary to show that a ranking function (and, more generally, a polyranking function) is bounded in the loop; we say such invariants are *supporting* invariants.

Definition 7 (Polynomial Inductive Invariant) A formula φ is an invariant of a loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ if it satisfies the following conditions:

(Initiation) $\theta(\mathcal{V}) \rightarrow \varphi(\mathcal{V})$, and
(Consecution) for every $\tau \in \mathcal{T}$, $\varphi(\mathcal{V}) \wedge \tau(\mathcal{V}, \mathcal{V}') \rightarrow \varphi(\mathcal{V}')$.

Example 1. Consider the loop SIMPLE in Figure 1. Initially, $x+y$ is nonnegative. Transitions τ_1 and τ_2 are both guarded by $x \leq N$. Termination is relatively straightforward: τ_2 at least increments x, while τ_1 increases x if $x + y > 0$ and always increases y. Discovering the invariant $x + y \geq 0$ shows that incrementing y makes $x + y > 0$. However, a linear ranking function over $\{x, y, N\}$ cannot prove termination, as, for example, τ_1 does not change x when $x = -y < 0$.

$$\begin{array}{l} \theta: \ \{x + y \geq 0\} \\ \tau_1: \{x \leq N\} \Rightarrow \{x' \geq 2x + y,\ y' \geq y + 1,\ N' = N\} \\ \tau_2: \{x \leq N\} \Rightarrow \{x' \geq x + 1,\ y' = y,\ N' = N\} \end{array}$$

Fig. 1. Program SIMPLE, written as a loop.

$$\begin{array}{l} \theta: \ \{p \geq 0,\ q \geq 1,\ x = 0,\ y = 0\} \\ \tau_1: \{x + y \leq N\} \Rightarrow \left\{ \begin{array}{c} x + e - q \leq x' \leq x + e + q, \\ y + n - q \leq y' \leq y + n + q, \\ n + e + 1 \leq n' + e' \leq n + e + p, \\ p' = p,\ q' = q,\ N' = N \end{array} \right\} \\ \tau_2: \{x + y \leq N,\ n + e \geq 2q + 1\} \Rightarrow \left\{ \begin{array}{c} x + e - q \leq x' \leq x + e + q, \\ y + n - q \leq y' \leq y + n + q, \\ n' = n,\ e' = e,\ p' = p,\ q' = q,\ N' = N \end{array} \right\} \\ \tau_3: \{p \geq 0\} \Rightarrow \left\{ \begin{array}{c} n' + e' \leq -(n + e), \\ p' = p - 1,\ q' = \frac{1}{2}q, \\ x' = x,\ y' = y,\ N' = N \end{array} \right\} \end{array}$$

Fig. 2. Description of the erratic robot as the program ERRATIC.

Example 2. Consider the program in Figure 2, written as a loop. It defines the behavior of an erratic robot on a partially-bounded plane. Its current position is given by $(x,\ y)$; its tendency to move *north* is given by n (which may be negative), while its tendency to move *east* is given by e (which may also be negative). In τ_1 and τ_2, the robot's next position is determined by its current position and $(e,\ n)$, along with an error parameter q. In τ_1, n and e change so that their new sum is at least one greater than previously, but in τ_2, n and e remain constant. Finally, τ_3 adjusts parameters p and q and makes $n + e$ potentially negative (countering τ_1), but it does not change the robot's position.

Does the robot's program eventually halt? Intuitively, τ_3 may only be taken a finite number of times, while τ_1 and τ_2 *eventually* make $x + y$ only increase. However, a linear ranking function cannot suffice to show termination, as τ_1 does not make obvious progress on each iteration.

Finally, for synthesis, Farkas's Lemma [12] will serve both as a device for generating constraint systems and as the foundation for completeness claims.

Theorem 1 (Farkas's Lemma). *Consider the following system of linear inequalities over real variables $\mathcal{V} = \{x_1, \ldots, x_m\}$:*

$$S: \begin{bmatrix} A_{1,1}x_1 + \cdots + A_{1,m}x_m + A_{1,m+1} \geq 0 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ A_{n,1}x_1 + \cdots + A_{n,m}x_m + A_{n,m+1} \geq 0 \end{bmatrix}$$

If S is satisfiable, it entails linear inequality $c_1x_1 + \cdots + c_mx_m + c_{m+1} \geq 0$ iff there exist real numbers $\lambda_1, \ldots, \lambda_n \geq 0$ such that

$$c_1 = \sum_{i=1}^{n} \lambda_i A_{i,1} \quad \cdots \quad c_m = \sum_{i=1}^{n} \lambda_i A_{i,m} \quad c_{m+1} \geq \left(\sum_{i=1}^{n} \lambda_i A_{i,m+1} \right).$$

(Negative) $E(\mathcal{V})$ is bounded: $L \models E(\mathcal{V}) \leq -\epsilon$ for some $\epsilon > 0$
(Eventually Negative) For each $\tau \in \overline{\mathcal{T}}$, one of the following conditions holds:
  (Nonincreasing) $\tau \in \overline{\mathcal{T}} - \mathcal{A}$ and τ does not increase $E(\mathcal{V})$:
    1. $\tau \in \overline{\mathcal{T}} - \mathcal{A}$
    2. $L \models \tau(\mathcal{V}, \mathcal{V}') \rightarrow E(\mathcal{V}') - E(\mathcal{V}) \leq 0$
  (Eventually Decreasing) τ does not increase $E(\mathcal{V})$ by more than some eventually negative $F(\mathcal{V})$:
    1. $L \models \tau(\mathcal{V}, \mathcal{V}') \rightarrow E(\mathcal{V}') - E(\mathcal{V}) \leq F(\mathcal{V})$
    2. $F(\mathcal{V})$ is *eventually negative by* $\{\tau\} \subseteq \overline{\mathcal{T}}$

Fig. 3. $E(\mathcal{V})$ is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$ on loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$.

Furthermore, S is unsatisfiable iff S entails $-1 \geq 0$.

3 The Polyranking Principle

The polyranking principle is based on the following recursive definition of what it means for an expression $E(\mathcal{V})$ to be *eventually negative*.

Definition 8 (Eventually Negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$) Given loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ over $\mathcal{V}$, an expression $E(\mathcal{V})$ is *eventually negative by* $\mathcal{A} \subseteq \overline{\mathcal{T}} (\subseteq \mathcal{T})$, for $\mathcal{A} \neq \emptyset$, if either case **Negative** or case **Eventually Negative** holds in Figure 3. For case **Eventually Negative** to hold, the recursion must have finite depth.

Intuitively, transitions in $\mathcal{A}$ should make progress toward decreasing $E(\mathcal{V})$, while transitions in $\overline{\mathcal{T}}$ should not interfere counterproductively.

We represent an application of this definition as a tree of expressions, called an *EN-tree*. $E(\mathcal{V})$ is the root; the **Negative** case introduces the $-\epsilon$ expressions as leaves; the **Nonincreasing** case introduces 0-leaves; and the **Eventually Decreasing** case introduces $F(\mathcal{V})$ expressions as inner nodes. Branches are labeled by transitions.

Example 3. Figure 4 shows three examples of EN-trees. The tree in Figure 4**(a)** for SIMPLE represents the property that $N-x$ is eventually negative by $\{\tau_1, \tau_2\} \subseteq \{\tau_1, \tau_2\}$. The first difference of $N - x$ by τ_1 is at most $-x - y$, which is itself eventually negative. The supporting invariant $x + y \geq 0$ is required for proving that the first difference of $-x - y$ by τ_1 is at most -1.

Lemma 1. *If for loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$, $E(\mathcal{V})$ is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$ ($\subseteq \mathcal{T}$), then on any nonterminating computation of L, γ, such that $io(\gamma) \subseteq \overline{\mathcal{T}}$ and $io(\gamma) \cap \mathcal{A} \neq \emptyset$, eventually henceforth $E(\mathcal{V}) \leq -\epsilon$ for some $\epsilon > 0$.*

Proof. We proceed by induction on the structure of Definition 8 and the depth of the recursion, which we know is finite. As the base case, consider when **Negative** applies to $E(\mathcal{V})$; then the conclusion is immediate. For the inductive case, **Eventually Negative** applies. Consider $\tau \in \mathcal{T}$. If $\tau \notin io(\gamma)$, then we can, without loss of generality, assume that τ is never taken in γ — just skip a finite prefix — so that τ has no effect on $E(\mathcal{V})$. Otherwise, $\tau \in io(\gamma)$. There are two cases. If $\tau \notin \mathcal{A}$ and **Nonincreasing** applies to τ, then τ does not increase $E(\mathcal{V})$. Otherwise, **Eventually Decreasing** applies to τ, so that τ increases $E(\mathcal{V})$ by at most $F(\mathcal{V})$, while $F(\mathcal{V})$ is eventually negative by $\{\tau\} \subseteq \overline{\mathcal{T}}$. Since $io(\gamma) \cap \{\tau\} \neq \emptyset$, we have by induction that eventually henceforth $F(\mathcal{V}) \leq -\epsilon$ for some $\epsilon > 0$. By assumption, there is at least one $\tau \in \mathcal{A} \cap io(\gamma)$. Once each of these τ's $F(\mathcal{V})$ is henceforth negative, $E(\mathcal{V})$ is decreased by at least some $\epsilon > 0$ infinitely many times. Since no other transition can increase $E(\mathcal{V})$, eventually henceforth $E(\mathcal{V}) \leq -\epsilon$.

Example 4. For SIMPLE, the expression $N - x$ is bounded from below by 0 within the loop and eventually negative, as shown by the EN-tree in Figure 4**(a)**. Thus, SIMPLE always terminates.

Definition 9 (Lexicographic Polyranking Function) The ℓ-tuple of functions $\langle r_1(\mathcal{V}),\ r_2(\mathcal{V}),\ \ldots,\ r_\ell(\mathcal{V})\rangle$ is a *lexicographic polyranking function* of loop $L : \langle \mathcal{V}, \theta, \mathcal{T}\rangle$ if for some $\pi : \mathcal{T} \to \{1, \ldots, \ell\}$,

(Bounded) for each $\tau \in \mathcal{T}$, $L \models \tau(\mathcal{V}, \mathcal{V}') \to r_{\pi(\tau)}(\mathcal{V}) \geq 0$;
(Polyranking) $r_i(\mathcal{V})$ is *eventually negative by* $\{\tau : \pi(\tau) = i\} \subseteq \{\tau : \pi(\tau) \geq i\}$.

Theorem 2. *If loop $L : \langle \mathcal{V}, \theta, \mathcal{T}\rangle$ has a lexicographic polyranking function, then it always terminates.*

Proof. Suppose L does not always terminate, yet $\langle r_1(\mathcal{V}),\ r_2(\mathcal{V}),\ \ldots, r_\ell(\mathcal{V})\rangle$ with map $\pi : \mathcal{T} \to \{1, \ldots, \ell\}$ is a lexicographic polyranking function for L. Let γ be an infinite computation, and let $i \in \{1, \ldots, \ell\}$ be the lexicographic index such that $i = \min_{\tau \in io(\gamma)} \pi(\tau)$. By Definition 9, $r_i(\mathcal{V})$ is eventually negative by $\{\tau : \pi(\tau) = i\} \subseteq \{\tau : \pi(\tau) \geq i\}$. Moreover, by selection of i, $\{\tau : \pi(\tau) = i\} \cap io(\gamma) \neq \emptyset$; and by selection of i and Definition 9, $io(\gamma) \subseteq \{\tau : \pi(\tau) \geq i\}$. Then by Lemma 1, $r_i(\mathcal{V})$ eventually becomes negative and stays negative, disabling transitions in $\{\tau : \pi(\tau) = i\}$ and thus increasing i for the remaining computation. Repeating this argument at most ℓ times proves that all transitions are eventually disabled, a contradiction.

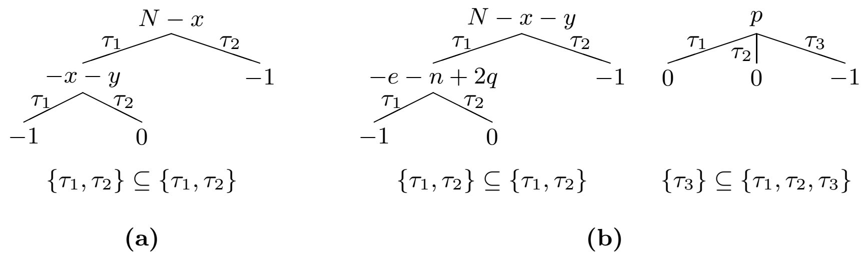


Fig. 4. EN-trees for **(a)** SIMPLE and **(b)** ERRATIC.

$\mathbb{I} : \dfrac{\Theta\mathbf{x} \geq \mathbf{0}}{\mathbf{Ix} \geq \mathbf{0}}$	$\mathbb{D}_i : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{-1 \geq \mathbf{0}}$	$\mathbb{C}_i : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{\mathbf{Ix}' \geq \mathbf{0}}$
$\mathbb{B}_{ij} : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{\mathbf{c_j}^{\mathrm{T}}\mathbf{x} \geq \mathbf{0}}$	$\mathbb{R}_{ij} : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{\mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' - \epsilon \geq \mathbf{0}}$	$\mathbb{U}_{ij} : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{\mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' \geq \mathbf{0}}$
$\mathbb{D}^{\star}_{ijk} : \dfrac{\begin{array}{ll}\mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0}\end{array}}{\mathbf{c_k}^{\mathrm{T}}\mathbf{x} + \mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' \geq \mathbf{0}}$		

Fig. 5. Farkas's Lemma specializations.

Example 5. ERRATIC has lexicographic linear polyranking function $\langle p,\ N-x-y \rangle$ where $\pi : \{\tau_1 \mapsto 2,\ \tau_2 \mapsto 2,\ \tau_3 \mapsto 1\}$. Clearly, $N - x - y$ is bounded from below by 0 by both τ_1 and τ_2, while p is bounded from below by 0 by τ_3. Figure 4**(b)** presents the EN-trees representing the proofs that $N - x - y$ is eventually negative by $\{\tau_1, \tau_2\} \subseteq \{\tau_1, \tau_2\}$ and p is eventually negative by $\{\tau_3\} \subseteq \{\tau_1, \tau_2, \tau_3\}$. No supporting invariants are required.

4 Constraint Generation

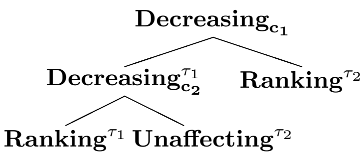
The remainder of this paper focuses on a complete method of synthesizing lexicographic linear polyranking functions with supporting linear invariants for linear loops. We adapt the machinery of [1] to this more general setting. The main idea is to invoke a set of Farkas's Lemma *specializations* on a loop and a set of *template* ranking functions and invariants.

Definition 10 (Template Expression) A *template expression* over $\mathcal{V}$ is a linear expression $\mathbf{c}^{\mathrm{T}}\mathbf{x}$, with unknown coefficients $\mathbf{c}$. A *template assertion* is a linear assertion $\mathbf{Cx} \geq 0$ with matrix of unknown coefficients $\mathbf{C}$.

Farkas's Lemma takes a system of linear assertions and templates and returns a *dual* numeric constraint system over the λ-multipliers and the unknown template coefficients. Given a loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$, the supporting invariant template $\mathbf{Ix} \geq 0$, and template expressions $\{\mathbf{c_1}^{\mathrm{T}}\mathbf{x},\ \ldots, \mathbf{c_n}^{\mathrm{T}}\mathbf{x}\}$, the following Farkas's Lemma *specializations* are applied to encode the appropriate conditions.

Definition 11 (Farkas's Lemma Specializations) See Figure 5.

- $\mathbb{I}$ (*Initiation*): The supporting invariant includes the initial condition.
- $\mathbb{D}_i$ (*Disabled*): Transition $\tau_i \in \mathcal{T}$ and the invariant may contradict each other, indicating "dead code."
- $\mathbb{C}_i$ (*Consecution*): For transition $\tau_i \in \mathcal{T}$, if the invariant holds and the transition is taken, then the invariant holds in the next state.
- $\mathbb{B}_{ij}$ (*Bounded*): For transition $\tau_i \in \mathcal{T}$ and template expression $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, the invariant and transition imply the nonnegativity of the expression.



(a)

$$\begin{aligned}
\textbf{Decreasing}_{\mathbf{c_1}} &\Rightarrow (\mathbb{D}_1 \vee \mathbb{B}_{1,1}) \wedge (\mathbb{D}_2 \vee \mathbb{B}_{2,1}) \\
\textbf{Decreasing}^{\tau_1}_{\mathbf{c_2}} &\Rightarrow \mathbb{D}_1 \vee \mathbb{D}^{\star}_{1,1,2} \\
\textbf{Ranking}^{\tau_1} &\Rightarrow \mathbb{D}_1 \vee \mathbb{R}_{1,2} \\
\textbf{Unaffecting}^{\tau_2} &\Rightarrow \mathbb{D}_2 \vee \mathbb{U}_{2,2} \\
\textbf{Ranking}^{\tau_2} &\Rightarrow \mathbb{D}_2 \vee \mathbb{R}_{2,1}
\end{aligned}$$

(b)

Fig. 6. **(a)** Template tree. **(b)** Generating tree constraints.

- $\mathbb{R}_{ij}$ (*Ranking*): For transition $\tau_i \in \mathcal{T}$ and template expression $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, taking the transition decreases the value of the expression by at least some positive amount ($\epsilon > 0$).
- $\mathbb{U}_{ij}$ (*Unaffecting*): For transition $\tau_i \in \mathcal{T}$ and template expression $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, taking the transition does not increase the value of the expression.
- $\mathbb{D}^{\star}_{ijk}$ (*Decreasing*): For transition $\tau_i \in \mathcal{T}$ and template expressions $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$ and $\mathbf{c_k}^{\mathrm{T}}\mathbf{x}$, the first difference of $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$ over τ_i is upper bounded by $\mathbf{c_k}^{\mathrm{T}}$.

All specializations except $\mathbb{D}^{\star}_{ijk}$ are used in [1] for synthesizing linear ranking functions, and thus explained in greater depth there. $\mathbb{D}^{\star}_{ijk}$ is the key specialization for polyranking function synthesis. It allows us to express the condition that the difference $\mathbf{c_j x'} - \mathbf{c_j x}$ is upper bounded by the expression $\mathbf{c_k x}$, which in turn is constrained to be eventually negative.

Applying a set of specializations to a loop and a set of template expressions induces a numeric constraint system. However, we must define how to apply the specializations. For now, we assume that we are given a tuple of *template trees* and a map π mapping transitions to tuple components. The i^{th} template gives the form of an EN-tree for the property that the root expression is eventually negative by $\{\tau : \pi(\tau) = i\} \subseteq \{\tau' : \pi(\tau') \geq i\}$.

Definition 12 (Template Tree) A *template tree* for loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ and $\mathcal{A} \subseteq \overline{\mathcal{T}}$ gives the form of an EN-tree for the property that the root expression is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$. The root node and inner nodes are **Decreasing** nodes (corresponding to the root expression and **Eventually Decreasing** case of Definition 8, respectively) labeled with template expressions, while leaves are **Ranking** nodes (the **Negative** case) or **Unaffecting** nodes (the **Nonincreasing** case). Non-root nodes are also labeled with some $\tau \in \overline{\mathcal{T}}$. Each inner node has $|\overline{\mathcal{T}}|$ children.

Definition 8 indicates when a leaf node must be **Ranking**$^{\tau}$ or **Unaffecting**$^{\tau}$. Specifically, if the node is a child of the root and $\tau \in \mathcal{A}$, or if its parent is a node labeled by τ, then the node is **Ranking**$^{\tau}$; otherwise, it is **Unaffecting**$^{\tau}$.

Example 6. The template tree for $\{\tau_1, \tau_2\} \subseteq \{\tau_1, \tau_2\}$ of SIMPLE in Figure 6**(a)** includes the EN-tree in Figure 4**(a)**.

Definition 13 (Tree Constraints) Given a loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ and a template tree for $\mathcal{A} \subseteq \overline{\mathcal{T}}$, the *tree constraints* are generated by applications of the Farkas's Lemma specializations as follows:

Parent is $\mathbf{Decreasing}^{*}_{\mathbf{c_j}}$ and node is ...

$$
\begin{array}{lcl}
\ldots\ \mathbf{Ranking}^{\tau_i} & \Rightarrow & \mathbb{D}_i \vee \mathbb{R}_{ij} \\
\ldots\ \mathbf{Unaffecting}^{\tau_i} & \Rightarrow & \mathbb{D}_i \vee \mathbb{U}_{ij} \\
\ldots\ \mathbf{Decreasing}^{\tau_i}_{\mathbf{c_k}} & \Rightarrow & \mathbb{D}_i \vee \mathbb{D}^{\star}_{ijk}
\end{array}
$$

Node is *root* and $\mathbf{Decreasing}_{\mathbf{c_j}} \Rightarrow \mathbb{D}_i \vee \mathbb{B}_{ij}$ for each $\tau_i \in \mathcal{A}$

Example 7. The tree constraints of the template tree in Figure 6(**a**) are generated as in Figure 6(**b**). The *disabled* cases are often ignored in practice, so the template tree induces numerical constraints corresponding to $\mathbb{B}_{1,1} \wedge \mathbb{B}_{2,1} \wedge \mathbb{D}^{\star}_{1,1,2} \wedge \mathbb{R}_{1,2} \wedge \mathbb{U}_{2,2} \wedge \mathbb{R}_{2,1}$.

We use template trees and tree constraints to formalize the synthesis of lexicographic linear polyranking functions. This theorem mimics Definition 9 in the context of synthesis.

Theorem 3. *Loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$ has an ℓ-lexicographic linear polyranking function supported by an n-conjunct linear inductive invariant iff there exists a mapping $\pi : \mathcal{T} \to \{1, \ldots, |\mathcal{T}|\}$ and an ℓ-tuple of template trees $\langle T_1,\ T_2,\ \ldots,\ T_\ell \rangle$ such that (1) the i^{th} template tree is for $\{\tau : \pi(\tau) = i\} \subseteq \{\tau' : \pi(\tau') \geq i\}$, and (2) the ℓ numeric constraint systems induced by*

$$\mathbb{I}\ \wedge \bigwedge_{\tau_i \in \mathcal{T}} (\mathbb{D}_i \vee \mathbb{C}_i)\ \wedge\ \mathit{tree\_constraints}(T_j) \quad \mathit{for} \quad j \in \{1, \ldots, \ell\}$$

are satisfiable.

5 Synthesis

Theorem 3 applies to a given tuple of template trees. But in practice, we must find this tuple of templates. This section addresses this issue with an effective search, which seeks a tuple of trees such that the root expressions form a lexicographic linear polyranking function for a loop $L : \langle \mathcal{V}, \theta, \mathcal{T} \rangle$. Three different forms of constraints are handled by the full algorithm. The *lexicographic constraints* induce the transition sets $\mathcal{A}$ and $\overline{\mathcal{T}}$ for each component tree. These transition sets plus the *well-formedness constraints* from the definition of EN-trees induce a *numeric constraint system* via the tree constraints. Only when the two sets of structural constraints and the induced numeric constraints are satisfied is a solution found.

The first algorithm searches for EN-trees. Its input is a loop; an n-conjunct invariant template; the two sets of transitions, $\mathcal{A}$ and $\overline{\mathcal{T}}$; and a maximum tree height. It returns whether an EN-tree exists of at most the maximum height, possibly supported by an n-conjunct invariant. Its search strategy consists of incrementally building partial template trees. A partial template tree induces a numeric constraint system that is satisfiable if some completion of the tree induces a satisfiable constraint system. Thus, solving partial constraint systems guides the search. If a complete template tree is found that induces a satisfiable constraint system, the algorithm reports success.

$$
\begin{array}{l}
\texttt{let}\ \mathit{exists\_tree}\ \mathcal{A}\ \overline{\mathcal{T}}\ =\\
\quad \texttt{let}\ \mathit{initial\_tree}\ =\\
\qquad \mathbf{Decreasing_{c_1}}\left(\begin{array}{l}\{\mathbf{ShouldRank}^{\tau}\ :\ \tau\in\mathcal{A}\}\cup\\ \quad\{\mathbf{ShouldUnaffect}^{\tau}\ :\ \tau\in\overline{\mathcal{T}}-\mathcal{A}\}\end{array}\right)\\
\quad \texttt{in}\\
\quad \texttt{let}\ \mathit{sat}\ \mathit{en}\ =\ \mathit{num\_sat}\ (\mathit{inv\_cts}\ \wedge\ (\mathit{tree\_cts}\ \mathit{en}))\ \texttt{in}\\
\quad \texttt{let}\ \mathit{next\_tree}\ \mathit{en}\ \mathit{extend}\ =\ \ldots\ \texttt{in}\\
\quad \texttt{let rec}\ \mathit{search}\ \mathit{en}\ =\\
\qquad\quad \mathit{height}\ \mathit{en}\ \leq\ \mathit{max\_height}\\
\qquad \texttt{and}\ \mathit{sat}\ \mathit{en}\\
\qquad \texttt{and}\left(\begin{array}{l}\quad\ \mathit{complete}\ (\mathit{en})\\ \texttt{or}\ \mathit{search}\ (\mathit{next\_tree}\ \mathit{en}\ \texttt{false})\\ \texttt{or}\ \mathit{search}\ (\mathit{next\_tree}\ \mathit{en}\ \texttt{true})\end{array}\right)\\
\quad \texttt{in}\\
\quad \mathit{search}\ \mathit{initial\_tree}
\end{array}
$$

Fig. 7. ($\mathit{exists\_tree}\ \mathcal{A}\ \overline{\mathcal{T}}$) searches for an EN-tree proving that there is a bounded expression $E(\mathbf{x})$ that is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$.

Partial template trees are represented as in Section 4, except that two additional nodes are introduced. **ShouldRank** and **ShouldUnaffect** nodes are directions for making a partial template tree complete. In a complete template tree, all such nodes are replaced by one of the three original nodes. Indeed, complete trees must have a form obeying Definition 12. The definition of tree constraints is extended to the new nodes by simply ignoring them: **ShouldRank** and **ShouldUnaffect** nodes do not induce constraints. In this section, we represent the children of **Decreasing** nodes as a set labeling the node, rather than as figures like Figure 6**(a)**.

Figure 7 describes the procedure as the function *exists_tree*. It searches for an EN-tree of at most the specified maximum height, *max_height*, corresponding to the property that the root expression is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$. It defines the initial tree, *initial_tree*, as a **Decreasing** node with **ShouldRank** $\mathcal{A}$ children and **ShouldUnaffect** $\overline{\mathcal{T}} - \mathcal{A}$ children. The function *sat* builds the numerical constraint system induced by the given tree *en* and returns its satisfiability. The constraints include those on the template invariant (*inv_cts*).

The search proceeds by exploring the space of partial template trees via *next_tree*. For current tree *en* and boolean *extend*, (*next_tree en extend*) returns the next tree in the search space. The next tree is constructed by replacing some $\mathbf{ShouldRank}^{\tau}$ ($\mathbf{ShouldUnaffect}^{\tau}$) node with a $\mathbf{Ranking}^{\tau}$ ($\mathbf{Unaffecting}^{\tau}$) or a $\mathbf{Decreasing}^{\tau}_{\mathbf{c_k}}$ node. The choice is controlled by *extend*. When *extend* is `true`, the tree is "extended" by replacing the node with a $\mathbf{Decreasing}^{\tau}_{\mathbf{c_k}}$ node, which has a fresh template expression $\mathbf{c_k}$ and a set of $\mathbf{ShouldRank}^{\tau'}$ and $\mathbf{ShouldUnaffect}^{\tau'}$ children, where a child is $\mathbf{ShouldRank}^{\tau'}$ only if $\tau = \tau'$.

Example 8. Consider ERRATIC with a linear transition order $\tau_3 \prec \tau_1 \prec \tau_2$. The initial partial template trees for the lexicographic components of τ_1, τ_2, and τ_3

are the following:

τ_1 : $\mathbf{Decreasing_{c_1}}\{\mathbf{ShouldRank}^{\tau_1}, \mathbf{ShouldUnaffect}^{\tau_2}\}$
τ_2 : $\mathbf{Decreasing_{c_2}}\{\mathbf{ShouldRank}^{\tau_2}\}$
τ_3 : $\mathbf{Decreasing_{c_3}}\{\mathbf{ShouldUnaffect}^{\tau_1}, \mathbf{ShouldUnaffect}^{\tau_2}, \mathbf{ShouldRank}^{\tau_3}\}$

The next tree for τ_1, without extending the tree, could be

$$\mathbf{Decreasing_{c_1}}\{\mathbf{Ranking}^{\tau_1}, \mathbf{ShouldUnaffect}^{\tau_2}\};$$

however, the corresponding numeric constraint system is unsatisfiable. Therefore, with extension, the next tree is

$$\mathbf{Decreasing_{c_1}}\left\{\begin{array}{l}\mathbf{Decreasing}^{\tau_1}_{\mathbf{c_4}}\{\mathbf{ShouldRank}^{\tau_1}, \mathbf{ShouldUnaffect}^{\tau_2}\}, \\ \mathbf{ShouldUnaffect}^{\tau_2}\end{array}\right\}.$$

Finally, the search runs as follows. It checks that the given template tree, *en*, has a height at most *max_height* and that the numeric constraint system induced by *en* is feasible. If the system is infeasible, then no extension of *en* can induce a feasible constraint system, so the search halts on the branch. If both checks are satisfied and the tree is *complete*, then a satisfying tree has been found, so the search returns `true`. If the tree is still incomplete, it recurses on the next tree, first without extension and then with extension. Each of the two calls to *next_tree* should modify the same node.

Lemma 2. *If num_sat is a decision procedure,* ($exists\_tree\ \mathcal{A}\ \overline{\mathcal{T}}$) *returns* `true` *iff there is an expression* $E(\mathbf{x})$ *and an n-conjunct supporting linear invariant* $I\mathbf{x} \geq 0$ *such that (1)* $E(\mathbf{x})$ *is lower bounded by* 0 *by each* $\tau \in \mathcal{A}$ *relative to* $I\mathbf{x} \geq 0$*, and (2)* $E(\mathbf{x})$ *is eventually negative by* $\mathcal{A} \subseteq \overline{\mathcal{T}}$ *relative to* $I\mathbf{x} \geq 0$*, and the associated EN-tree has height at most max_height.*

Example 9. After several more iterations for each template tree in Example 8, each **ShouldRank** (**ShouldUnaffect**) node is replaced by a **Ranking** (**Unaffecting**) node. One possible solution is discussed in Example 4, where the trees for τ_1 and τ_2 have been merged into one tree.

The second algorithm, *lex*, adapted from [1], works on top of the first to find a lexicographic function. It associates with each transition $\tau \in \mathcal{T}$ a template expression $T(\tau)$. It seeks a linear order among the transitions by incrementally searching over partial orders. A partial order $\prec$ over $\mathcal{T}$ induces a set of constraints: for each τ, $T(\tau)$ must be eventually negative by $\{\tau\} \subseteq \{\tau' : \tau' \prec \tau \vee \tau' = \tau\}$; and $T(\tau)$ must be lower bounded by 0 when τ is enabled. These constraints are checked by the first algorithm. If $\prec$ induces an unsatisfiable constraint system, then no linear extension can induce a satisfiable constraint system. Thus, solving partial constraint systems guides this search, as well. If a linear order is found that induces a satisfiable constraint system, the second algorithm has found a lexicographic polyranking function.

Proposition 1. *Consider loop* $L : \langle \mathcal{V}, \theta, \mathcal{T}\rangle$*, an n-conjunct template invariant, and maximum tree height max_height.* ($lex\ L$) *returns* `true` *iff* L *has a lexicographic linear polyranking function such that the associated EN-trees have height at most max_height and are each supported by an n-conjunct linear invariant.*

6 Conclusion

We implemented the lexicographic linear polyranking approach to proving termination of linear loops in our tool, `linsys`. The implementation found the polyranking function and supporting invariant described in Example 4 in about 2 seconds and the lexicographic polyranking function described in Example 5 in about 1 second. The primary challenge in scaling our synthesis method to assertional polynomial loops is solving the polynomial constraint systems. Approximation methods like in [6] may address scalability.

We believe that two ideas from this paper are useful in domains other than termination analysis. First, the recursive definition of an eventually negative expression may be of interest for hybrid system analysis and control theory. Second, the form of our synthesis method for solving structural and numeric constraints should be applicable to other constraint solving and synthesis tasks.

Acknowledgments We thank the reviewers for their insightful comments.

References

1. Bradley, A. R., Manna, Z., and Sipma, H. B. Linear ranking with reachability. In *CAV* (2005).
2. Bradley, A. R., Manna, Z., and Sipma, H. B. Termination of polynomial programs. In *VMCAI* (2005), pp. 113–129.
3. Codish, M., Genaim, S., Bruynooghe, M., Gallagher, J., and Vanhoof, W. One lop at a time. In *WST* (2003).
4. Colón, M., and Sipma, H. B. Synthesis of linear ranking functions. In *TACAS* (2001), pp. 67–81.
5. Colón, M. A., and Sipma, H. B. Practical methods for proving program termination. In *CAV* (2002), pp. 442–454.
6. Cousot, P. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *VMCAI* (2005), pp. 1–24.
7. Dershowitz, N., Lindenstrauss, N., Sagiv, Y., and Serebrenik, A. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering, Communication and Computing 12* (2001), 117–156.
8. Lee, C. S., Jones, N. D., and Ben-Amram, A. M. The size-change principle for program termination. In *POPL* (2001), pp. 81–92.
9. Manna, Z., Browne, A., Sipma, H. B., and Uribe, T. E. Visual abstractions for temporal verification. In *Algebraic Methodology and Software Technology* (1998), pp. 28–41.
10. Podelski, A., and Rybalchenko, A. A complete method for the synthesis of linear ranking functions. In *VMCAI* (2004), pp. 239–251.
11. Podelski, A., and Rybalchenko, A. Transition invariants. In *LICS* (2004), pp. 32–41.
12. Schrijver, A. *Theory of Linear and Integer Programming*. Wiley, 1986.
13. Sipma, H. B., Uribe, T. E., and Manna, Z. Deductive model checking. In *CAV* (1996), pp. 209–219.
14. Tiwari, A. Termination of linear programs. In *CAV* (2004), pp. 70–82.