Polyranking for Polynomial Loops

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. We propose *polyranking functions* as a generalization of ranking functions for analyzing termination of loops. We define *lexicographic polyranking functions* in a general context and then specialize their synthesis and application to loops with polynomial guards and assignments of polynomial expressions. Finally, we demonstrate that our prototype implementation for C source code scales to large software projects, proving termination for a high percentage of targeted loops.

Key words: Verification, Static analysis, Termination

1 Introduction

Proving termination of program loops is necessary for ensuring the correct behavior of software, especially embedded systems and safety critical applications. It is also required when proving general temporal properties of infinite state programs (*e.g.*, (15; 18; 22)). The traditional method for proving loop termination is by proving that some function of the program variables is well-founded within the loop (14). Such a function is called a ranking function.

Discovering ranking functions is thus one way of automating termination proofs. Synthesizing *linear ranking functions* for *linear loops* has received a fair amount of attention in recent years. Colón and Sipma describe the synthesis of linear ranking functions over linear loops with variables ranging over $\mathbb{R}$ using polyhedra (10; 11). Their loops allow multiple paths and assertional transition relations. In (21), Podelski and Rybalchenko specialize the technique to a restricted class of single-path imperative loops without initial conditions, providing an efficient and complete synthesis method based on linear programming. We generalize these results in (2) to general linear loops: loops that contain multiple paths and that have a nontrivial initial condition. The method is complete for lexicographic linear ranking functions. It also allows synthesizing supporting invariants simultaneously with the ranking function. These synthesis methods rely fundamentally on the *Farkas Lemma*, which dualizes a conjunctive linear constraint system. When a parametric template is encoded in the place of an explicit ranking function and its supporting invariants, the *dual* constraint system constrains the unknown coefficients. Solving the constraint system produces the ranking function and its supporting invariants.

Automatic ranking function synthesis for other classes of loops has been studied recently. Cousot generalizes the Farkas Lemma-based methods of (9; 21) for linear loops to polynomial loops, using Lagrangian relaxation (12). In the polynomial context, solving the dual constraint system is incomplete: an infeasible dual constraint system does not indicate the non-existence of a polynomial ranking function. In another direction, we study linear loops in which variables range over $\mathbb{Z}$, rather than $\mathbb{R}$ (4). In the integer case, there is no known duality to exploit.

The above methods all synthesize ranking functions of a particular form. Proving that a function is a ranking function requires proving the validity of a finite set of *verification conditions*. Thus, like inductive invariants, ranking functions allow proving a global property of a loop via proving local conditions. Synthesizing ranking functions thus requires solving a finite set of local constraints. Section 2 presents a negative result that motivates this style of termination analysis. Specifically, we show that no general and complete termination analysis exists for a restricted class of linear loops. This class of loops is a subset of any class of loops interesting enough to analyze in practice. Therefore, we cannot expect a general and complete termination analysis for any sufficiently expressive class of loops, but instead should search for principled techniques like linear ranking function synthesis.

One approach to extending termination analysis is to seek looser restrictions on ranking functions. In (5; 3), we introduce the idea of *polyranking*, in which a tree of ranking functions proves termination. A polyranking function need not decrease on each iteration; rather, it must eventually only decrease. Additionally, it must have a lower bound within the loop, as with ranking functions. While subsuming ranking in power for a given class of functions, polyranking shares with ranking the important property of requiring that a finite number of verification conditions are proven valid. Thus, polyranking functions, too, allow proving termination via proving local conditions. In (3), we present the general polyranking method for loops with initial conditions and assertional transition relations. We also provide a practically useful synthesis technique for the linear case, which expands on the techniques of (9; 2). In (5), we look at the special case of loops in which transitions have polynomial assertional guards, but updates are restricted to assignment of polynomials to variables, rather than arbitrary assertions. We call these loops *polynomial loops*. In this paper, we expand on polyranking functions in general and the special case of polynomial loops.

These research efforts mainly focus on the synthesis of *measures*, so the methods are presented over loops with simple control structure. However, it can support theoretically complete verification frameworks like *verification diagrams* (18) and *transition invariants* (22) (see also (13; 17; 6) for related ideas). In this context, the user guides the construction of the proof with support from automatic invariant generation and polyranking function synthesis. A *verification diagram* is a state-based abstraction of the reachable state-space of a program. A *transition invariant* is a relation-based abstraction of the transition relation of a program. Its form is a disjunctive set of relations. Ramsey's Theorem ensures that if each of the relations is well-founded, then the transition relation is well-founded. Both frameworks identify the semantic behavior of a loop, so that its infinite behavior can be expressed disjunctively. In a verification diagram, a loop might manifest itself in multiple strongly connected components of the diagram, each of which can have a different ranking function. Similarly, in a transition invariant, each disjunct can have a different ranking function. In many cases, these multiple ranking functions are simpler than a single global ranking function, so that it is reasonable to expect that automatic synthesis can usually replace the user in proposing polyranking functions.

In this paper, we discuss polyranking in the context of *polynomial loops*. We show that this class of loops is sufficiently expressive to represent sound abstractions for most loops appearing in ordinary C code. We implemented our method and, via CIL (19), applied it to several large open-source C programs, with size up to 75K lines of code. The timing results clearly demonstrate the practicality of the analysis.

Finite differences play a fundamental role in our application of polyranking to polynomial loops. Finite differences have a long history in program analysis (*e.g.*, (24; 16; 7)). These methods construct and solve difference equations and inequalities, producing loop invariants, running times, and termination proofs. While the equations and inequalities are often difficult to solve, we observe that for termination analysis explicit solutions are unnecessary. Rather, our method analyzes loops for qualitative behavior — specifically, that certain expressions *eventually* only decrease by at least some positive amount, yet are bounded from below. We address the challenge of characterizing such behavior in loops with multiple paths and nonlinear assignments and guards.

The rest of the paper is ordered as follows. Section 2 introduces polynomial loops and shows a fundamental restriction on the power of automatic termination analysis. Section 3 discusses the general polyranking method, while Section 4 specializes it for polynomial loops. Section 5 explains a stronger termination analysis than polyranking for polynomial loops. Section 6 describes our prototype implementation and empirical results. Section 7 concludes.

2 Preliminaries

In this section, we define our loop abstraction and associated terminology. We end by proving a general result about termination analysis.

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 *$L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$ consists of variables $\overrightarrow{x}$,* initial condition *θ, and set of* transitions *$\mathcal{T}$. θ is a polynomial assertion over $\overrightarrow{x}$ expressing what is true before entering the loop. Each transition $\tau : G \Rightarrow \overrightarrow{x} := \overrightarrow{E}(\overrightarrow{x})$ consists of a* guard *G and an* update *$\overrightarrow{x} := \overrightarrow{E}(\overrightarrow{x})$. The guard is a polynomial assertion over $\overrightarrow{x}$, while the update is a simultaneous assignment of polynomial expressions to variables.*

Definition 3 (Transition Relation) *The* transition relation *ρ_τ of the transition $\tau : G \Rightarrow \overrightarrow{x} := \overrightarrow{E}(\overrightarrow{x})$ is $G \wedge \bigwedge_i x_i' = E_i(\overrightarrow{x})$, where the primed version of a variable indicates its value in the next state.*

Definition 4 (General Transition Relation) *A* general transition relation *ρ_τ is a conjunction of polynomial assertions over primed and unprimed variables $\overrightarrow{x} \cup \overrightarrow{x}'$, where the primed version of a variable indicates its value in the next state.*

Definition 5 (General Polynomial Loop) *A* general polynomial loop *$L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$ consists of variables $\overrightarrow{x}$,* initial condition *θ, and set of transitions $\mathcal{T}$ such that for each $\tau \in \mathcal{T}$, ρ_τ is a general transition relation. For general polynomial loops, we use the notation τ to indicate both the transition and its transition relation ρ_τ.*

Example 6 *Consider the polynomial loop* CHASE *in Figure 1. Formally, we*

$$\begin{array}{lllll} \tau_1\colon & \{x \geq y\} \Rightarrow & (x,\ y) & := & (x+1,\ y+x) \\ \tau_2\colon & \{x \geq y\} \Rightarrow & (x,\ y,\ z) & := & (x-z,\ y+z^2,\ z-1) \end{array}$$

Fig. 1. Polynomial loop CHASE.

write the loop $\langle\{x, y, z\}, \top, \{\tau_1, \tau_2\}\rangle$*, where the variables are* $\{x, y, z\}$*, the initial condition* $\top$ *imposes no restriction, and the transitions are* $\{\tau_1, \tau_2\}$*.* τ_1 *preserves the value of* z*.* x *and* y *may each increase or decrease, depending on current values. Further, while they both* eventually *increase, termination relies on* y *increasing more rapidly than* x*.*

Definition 7 (Linear Loop) *A* linear loop *is a polynomial loop in which all assertions and expressions are affine.*

Definition 8 (Computation) *A* computation $\langle\gamma = \tau_{j_0}\tau_{j_1}\tau_{j_2}\ldots,\ \sigma = s_0 s_1 s_2 \ldots\rangle$ *of a loop* $L : \langle\overrightarrow{x}, \theta, \mathcal{T}\rangle$ *consists of a* transition sequence γ *such that* $(\forall i)\tau_{j_i} \in \mathcal{T}$ *and a* state sequence σ*, where a* state *is a valuation of* $\overrightarrow{x}$*, such that* $(\forall i)\rho_{\tau_{j_i}}(s_i, s_{i+1})$*.*

Definition 9 (Loop Satisfaction) *A loop* $L : \langle\overrightarrow{x}, \theta, \mathcal{T}\rangle$ *satisfies an assertion* φ*, written* $L \models \varphi$*, if* φ *holds in all reachable states of* L*.*

Definition 10 (Infinitely Often) *For infinite computation* $\langle\gamma, \sigma\rangle$ *of loop* $L : \langle\overrightarrow{x}, \theta, \mathcal{T}\rangle$*,* $io(\gamma) \subseteq \mathcal{T}$ *is the set of transitions that occur* infinitely often.

In (2), we present a complete method for the synthesis of linear ranking functions: if a terminating loops has a linear ranking function, the method will find it. But not all terminating linear loops have linear ranking functions. Indeed, we show that no general and complete method exists to prove termination of linear loops, as stated by the following theorem.

Theorem 11 *Termination of linear loops is not semi-decidable.*

Proof. We proceed by reducing from Hilbert's 10^{th} problem, the existence of a nonnegative integer root of an arbitrary Diophantine equation, which is undecidable. First, we note that the existence of such a root is semi-decidable, via a proper enumeration of vectors of integers: if a root exists, the enumeration terminates with an affirmative answer. Thus, the nonexistence of nonnegative integer roots is not semi-decidable. Now, we reduce from the question of *nonexistence* of roots.

Instance: Given Diophantine equation $P(\overrightarrow{x}) = 0$ in variables $\overrightarrow{x} = \{x_1, \ldots, x_n\}$, determine if there does not exist a nonnegative integer solution.

Construction: Let $Q(\overrightarrow{x}) = P(\overrightarrow{x})^2$ so that always $Q(\overrightarrow{x}) \geq 0$. Construct a loop that mimics a random walk of $\mathbb{N}^n$. The loop has n transitions τ_i modeling the change in value of $Q(\overrightarrow{x})$ when x_i is incremented, and one transition τ_{n+1}

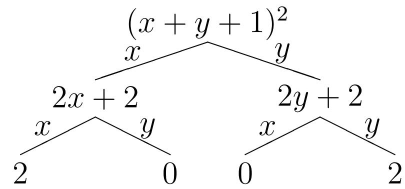


Fig. 2. The difference tree for $(x+y+1)^2$. Each edge is labeled by the variable to which the difference is applied.

that does not change any x_i. All $n+1$ transitions also decrement a counter, c, by $Q(\overrightarrow{x})$; the initial value of c is unrestricted. If $Q(\overrightarrow{x}) = 0$, then τ_{n+1} essentially does nothing. The loop terminates when c becomes negative. Then the only nonterminating case is when $P(\overrightarrow{x})$ has a nonnegative integer root.

It is easy to construct such a polynomial loop:

$$
\begin{array}{rlcll}
\theta\colon & \bigwedge_i x_i = 0 & & & \\
\tau_1\colon & \{c \geq 0\} \Rightarrow & (x_1,\ c) & := & (x_1+1,\ c - Q(\overrightarrow{x})) \\
 & & \cdots & & \\
\tau_n\colon & \{c \geq 0\} \Rightarrow & (x_n,\ c) & := & (x_n+1,\ c - Q(\overrightarrow{x})) \\
\tau_{n+1}\colon & \{c \geq 0\} \Rightarrow & (c) & := & (c - Q(\overrightarrow{x}))
\end{array}
$$

However, we want to construct a linear loop. The main difficulty is to track the value of the polynomial $Q(\overrightarrow{x})$ using only linear assignments. To this end, we construct a tree T of finite differences, with $Q(\overrightarrow{x})$ as the root. This tree is like a tree of partial derivatives. If $E(\overrightarrow{x})$ is a node, then its i^{th} of n children is $E_i(\overrightarrow{x}) = E(\overrightarrow{x})|_{x_i=x_i+1} - E(\overrightarrow{x})$. The degree of $E_i(\overrightarrow{x})$ in x_i is at most the degree of $E(\overrightarrow{x})$ in x_i minus one. The degrees in the other variables remain the same. Thus, letting the leaves be the first constants encountered during construction, the tree has finite height. For example, consider the tree for $(x+y+1)^2$ in Figure 2.

Define one variable $V(v)$ per non-leaf node v of T. Denote the variable associated with the root by r. Extend V to leaves by letting $V(v)$ be the constant expression of leaf v. Let $P(v)$ be the expression labeling vertex v. Now, the i^{th} transition, for $i \in \{1, \ldots, n\}$, is constructed as follows: for each non-leaf node v with i^{th} child u, assign $V(v) := V(v) + V(u)$, and assign $c := c - r$. The last transition is simply $c := c - r$. All transitions are guarded by $c \geq 0$. The initial condition is that $V(v) = P(v)(0, \ldots, 0)$ for each non-leaf node v, and $c = 0$. For example, consider the loop constructed for $(x+y+1)^2$ in Figure 3. The root variable is r; its left child's variable is q_1; and its right child's variable is q_2. τ_1 models the change to $(x+y+1)^2$ when x is incremented by 1, while τ_2 models the change corresponding to incrementing y.

$$
\begin{array}{rlcl}
\theta\!: & r = 1 \wedge q_1 = 2 \wedge q_2 = 2 & & \\
\tau_1\!: & \{c \geq 0\} \Rightarrow (r,\ q_1,\ q_2,\ c) & := & (r + q_1,\ q_1 + 2,\ q_2,\ c - r) \\
\tau_2\!: & \{c \geq 0\} \Rightarrow (r,\ q_1,\ q_2,\ c) & := & (r + q_2,\ q_1,\ q_2 + 2,\ c - r) \\
\tau_3\!: & \{c \geq 0\} \Rightarrow (c) & := & (c - r)
\end{array}
$$

Fig. 3. The constructed loop.

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

Fig. 4. Computing eventually negative.

The constructed loop does not always terminate iff $P(\overrightarrow{x})$ has a nonnegative integer root. For in that case, for some initial value of c and choice of transitions, eventually $r = 0$ and the last transition can be taken indefinitely. Otherwise, always $r \geq 1$ so that all transitions decrease c by at least 1. $\square$

3 The Polyranking Method

In this section, we consider the polyranking method for general polynomial loops. The polyranking method is based on the following recursive definition of what it means for an expression $E(\overrightarrow{x})$ to be *eventually negative*.

Definition 12 (Eventually Negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$) *Consider $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$. An expression $E(\overrightarrow{x})$ is* eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$*, for $\mathcal{A} \neq \emptyset$ and $\overline{\mathcal{T}} \subseteq \mathcal{T}$, if either case* **Negative** *or case* **Eventually Negative** *holds in Figure 4. For case* **Eventually Negative** *to hold, the recursion must have finite depth.*

Lemma 13 *If for loop $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$, $E(\overrightarrow{x})$ is eventually negative by $\mathcal{A} \subseteq \overline{\mathcal{T}}$, then on any nonterminating computation $\langle \gamma, \sigma \rangle$ of L such that $io(\gamma) \subseteq \overline{\mathcal{T}}$ and $io(\gamma) \cap \mathcal{A} \neq \emptyset$, eventually henceforth $E(\overrightarrow{x}) \leq -\epsilon$ for some $\epsilon > 0$.*

Proof. We proceed by induction on the structure of Definition 12 and the depth of the recursion, which we know is finite. As the base case, consider when **Negative** applies to $E(\overrightarrow{x})$; 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 $E(\overrightarrow{x})$. Otherwise,

$\tau \in io(\gamma)$. There are two cases. If $\tau \notin \mathcal{A}$ and **Nonincreasing** applies to τ, then τ does not increase $E(\overrightarrow{x})$. Otherwise, **Eventually Decreasing** applies to τ, so that τ increases $E(\overrightarrow{x})$ by at most $F(\overrightarrow{x})$, while $F(\overrightarrow{x})$ is eventually negative by $\{\tau\} \subseteq \overline{\mathcal{T}}$. Since $io(\gamma) \cap \{\tau\} \neq \emptyset$, we have by induction that eventually henceforth $F(\overrightarrow{x}) \leq -\epsilon$ for some $\epsilon > 0$. By assumption, there is at least one $\tau \in \mathcal{A} \cap io(\gamma)$. Once each of these τ's $F(\overrightarrow{x})$ is henceforth negative, $E(\overrightarrow{x})$ is decreased by at least some $\epsilon > 0$ infinitely many times. Since no other transition can increase $E(\overrightarrow{x})$, eventually henceforth $E(\overrightarrow{x}) \leq -\epsilon$. □

Consider, for example, how to define a non-lexicographic polyranking function using the eventually negative property. If an expression $E(\overrightarrow{x})$ is both bounded within a loop $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$, so that $E(\overrightarrow{x}) \geq 0$ if any transition is enabled, and *eventually negative by* $\mathcal{T} \subseteq \mathcal{T}$, then the loop always terminates. For on any infinite length computation $\langle \gamma, \sigma \rangle$, clearly $io(\gamma) \subseteq \mathcal{T}$ and $io(\gamma) \cap \mathcal{T} \neq \emptyset$. By the lemma, the expression is eventually negative, at which time no transition of $\mathcal{T}$ is enabled. If no $F(\overrightarrow{x})$ expressions are introduced, then $E(\overrightarrow{x})$ is a ranking function.

Definition 14 (Lexicographic Polyranking Function) *The ℓ-tuple of functions $\langle r_1(\overrightarrow{x}),\ r_2(\overrightarrow{x}),\ \ldots,\ r_\ell(\overrightarrow{x}) \rangle$ is a* lexicographic polyranking function *of loop $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$ if there exists map $\pi : \mathcal{T} \to \{1, \ldots, \ell\}$ such that*

(Bounded) *for each $\tau \in \mathcal{T}$,*

$$L \models \ \tau(\overrightarrow{x}, \overrightarrow{x}') \to r_{\pi(\tau)}(\overrightarrow{x}) \geq 0$$

(Polyranking) *for each $i \in \{1, \ldots, \ell\}$,*

$$r_i(\overrightarrow{x}) \textit{ is } \text{eventually negative by } \{\tau : \pi(\tau) = i\} \subseteq \{\tau : \pi(\tau) \geq i\}$$

Theorem 15 (Lexicographic Polyranking) *If loop $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$ has a lexicographic polyranking function, then it always terminates.*

Proof. Suppose that $\langle r_1(\overrightarrow{x}),\ r_2(\overrightarrow{x}),\ \ldots, r_\ell(\overrightarrow{x}) \rangle$ is a lexicographic polyranking function for L with with map $\pi : \mathcal{T} \to \{1, \ldots, \ell\}$, yet L does not always terminate. Let $\langle \gamma, \sigma \rangle$ be an infinite computation of L, and let $i \in \{1, \ldots, \ell\}$ be the lexicographic index such that $i = \min_{\tau \in io(\gamma)} \pi(\tau)$. By Definition 14, $r_i(\overrightarrow{x})$ 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 14, $io(\gamma) \subseteq \{\tau : \pi(\tau) \geq i\}$. Then by Lemma 13, $r_i(\overrightarrow{x})$ 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. □

```
let EN P(x⃗) 𝒜 𝒯̄ =
  let MAXIMUM Δ_{τ: G⇒x⃗:=E⃗(x⃗)} P(x⃗) =
    upper bound Δ_τ P(x⃗)
    subject to G
  in
  let rec FD_EN P(x⃗) max 𝒜 𝒯̄ =
    let CASE_NI τ = τ ∈ 𝒯̄ − 𝒜 ∧ MAXIMUM Δ_τ P(x⃗) ≤ 0 in
    let CASE_ED τ = FD_EN Δ_τ P(x⃗) (MAXIMUM Δ_τ P(x⃗)) {τ} 𝒯̄ in
    max < 0 ∨ ⋀_{τ∈𝒯̄} ((CASE_NI τ) ∨ (CASE_ED τ))
  in
  FD_EN P(x⃗) ∞ 𝒜 𝒯̄
```

Fig. 5. Finite difference-based implementation of Definition 12.

4 Polyranking with Finite Differences

We now specialize polyranking to the case of polynomial loops. To classify polynomial expressions as eventually negative, we use finite differences over transitions. We first recall the definition of a finite difference in the context of a polynomial loop.

Definition 16 (Finite Difference) *The* finite difference *of an expression $E(\overrightarrow{x})$ in $\overrightarrow{x}$ over assignment transition τ is*

$$\Delta_\tau E(\overrightarrow{x}) \stackrel{def}{=} E(\overrightarrow{x}') - E(\overrightarrow{x}),$$

where τ provides the value of $\overrightarrow{x}'$ in terms of $\overrightarrow{x}$. For convenience, we denote a chain of finite differences $\Delta_{\tau_{i_n}} \cdots \Delta_{\tau_{i_1}} E(\overrightarrow{x})$ by $\Delta_{\tau_{i_1},\ldots,\tau_{i_n}} E(\overrightarrow{x})$ or more simply by $\Delta_{i_1,\ldots,i_n} E(\overrightarrow{x})$. If $\tau_{i_n} = \cdots = \tau_{i_1}$, we denote the chain by $\Delta^n_{\tau_{i_1}} E(\overrightarrow{x})$ or more simply by $\Delta^n_{i_1} E(\overrightarrow{x})$. For list of transitions T with length n, we say that $\Delta_T E(\overrightarrow{x})$ is an n^{th} order *finite difference.*

Example 17 *For program* CHASE*, the first, second, and third order finite differences of $x - y$ over transition τ_1 are the following:*

$$\Delta_1(x-y) = (x+1) - (y+x) - (x-y) = 1 - x$$
$$\Delta_1^2(x-y) = \Delta_1(\Delta_1(x-y)) = \Delta_1(1-x) = 1 - (x+1) - (1-x) = -1$$
$$\Delta_1^3(x-y) = \Delta_1(\Delta_1^2(x-y)) = \Delta_1(-1) = (-1) - (-1) = 0.$$

Figure 5 presents the implemention of *eventually negative* using finite differences. Recursive function FD_EN parallels Figure 4: CASE_NI implements case **Nonincreasing**, while CASE_ED implements **Eventually Decreasing**. The parameter *max* is an upper bound on $P(\overrightarrow{x})$ so that $max < 0$ covers case **Negative**. The recursive call to FD_EN in CASE_ED computes the upper bound of $\Delta_\tau P(\overrightarrow{x})$ as the second argument, while the initial call to FD_EN in the last line passes ∞. Thus, *max* is always an upper bound on $P(\overrightarrow{x})$ in a call to FD_EN.

The function MAXIMUM can be implemented in several ways, as long as it returns an upper bound on $\Delta_\tau P(\overrightarrow{x})$ over G. For any finite difference $\Delta_{\tau:\ G \Rightarrow \overrightarrow{x} := \overrightarrow{E}(\overrightarrow{x})} P(\overrightarrow{x})$ arising in the analysis of a polynomial loop, the expression $\Delta_\tau P(\overrightarrow{x})$ is a polynomial and the guard $G(\overrightarrow{x})$ is a polynomial assertion. Thus, applying CAD (8) to simplify the formula $(\forall \overrightarrow{x})[G(\overrightarrow{x}) \rightarrow \Delta_\tau P(\overrightarrow{x}) = \epsilon]$ returns the exact ϵ (in general, a semi-algebraic set over ϵ) for which it is true. An upper bound on ϵ is an upper bound on $\Delta_\tau P(\overrightarrow{x})$. In special cases, the problem is a convex optimization problem, which can be solved efficiently (1). For example, if EN is applied to a linear loop, then all problem instances can be solved with linear programming.

Finding an upper bound of $\Delta_\tau P(\overrightarrow{x})$ over the domain given by the guard G allows EN to terminate on some cases on which it would not otherwise terminate. Invariants are useful here: strengthening the guard G can result in the existence of a constant upper bound that could not otherwise be found.

We represent executions of FD_EN with *finite difference trees.*

Definition 18 (Finite Difference Tree) *The* finite difference tree (FDT) *of an expression $E(\overrightarrow{x})$ with respect to transitions $\overline{\mathcal{T}} = \{\tau_1, \ldots, \tau_m\}$ has root $E(\overrightarrow{x})$ and branching factor m. Each node, indexed by its position T with respect to the root, represents finite difference $\Delta_T E(\overrightarrow{x})$. The first constant node along a path in the tree is a* leaf*; if its index is T, then it is an upper bound on the finite difference $\Delta_T E(\overrightarrow{x})$. The* height *of an FDT is the longest path to a leaf. A* finite FDT *is a finite difference tree with finite height.*

An FDT for an expression $E(\overrightarrow{x})$ over a transition set $\overline{\mathcal{T}}$ need not have finite height. In particular, assignments that result in exponential behavior (*e.g.*, $x := 2x$) can cause infinite-height FDTs. However, strengthening the loop with invariants, as described above, can result in a finite-height FDT.

Example 19 *Consider* CHASE. (EN $(x - y)$ $\{\tau_1, \tau_2\}$ $\{\tau_1, \tau_2\}$) *implicitly examines the FDT shown in Figure 6. Left (right) branches represent finite differences with respect to τ_1 (τ_2). Consider the leaves. -1, indexed by (τ_1, τ_1), satisfies case* **Negative***, as do (τ_1, τ_2, τ_2) and (τ_2, τ_2, τ_2). The leaves (τ_1, τ_2, τ_1), (τ_2, τ_1), and (τ_2, τ_2, τ_1) satisfy* **Nonincreasing**. *Going up through the tree, we see that $x - y$ is eventually decreasing by $\mathcal{T} \subseteq \mathcal{T}$. Therefore, because $x \geq y$ in the loop,* CHASE *must always terminate.*

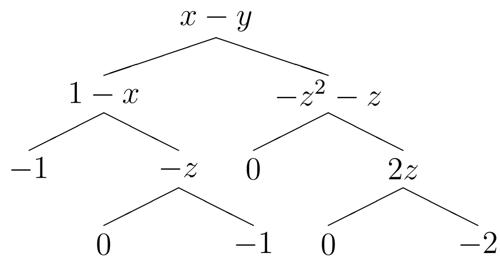


Fig. 6. Finite difference tree for CHASE of $x - y$ with respect to $\{\tau_1, \tau_2\}$.

```
let LEX L : ⟨x⃗, θ, 𝒯⟩ =
  let SAT o = ⋀_{τ∈𝒯} ⋁_{g∈guards(τ)} (EN g {τ} {τ' : τ = τ' ∨ (τ' > τ) ∈ o}) in
  let rec SEARCH o =
    SAT o and
      match CHOOSE_UNORDERED o with
      | None          →  true
      | Some (τ, τ')  →  [ (SEARCH (o ∪ (τ < τ')))
                                     or
                           (SEARCH (o ∪ (τ' < τ))) ]
  in
  SEARCH {}
```

Fig. 7. LEX L searches for a lexicographic linear polyranking function for L.

The finite difference method thus provides an efficient means of applying the concept of eventually negative. To finish, we need an efficient way of finding a lexicographic polyranking function. Definition 14 requires finding a π mapping transition indices to lexical component indices. While the number of possible π's is finite — as only at most $|\mathcal{T}|$ lexicographic components are required — it is exponentially large in the number of transitions.

Figure 7 presents LEX, which adapts the lexicographic search algorithm of (2) for polyranking functions. It searches for a lexicographic polyranking function with $|\mathcal{T}|$ components, where each transition $\tau : G \Rightarrow \vec{x} := \vec{E}(\vec{x})$ is associated with a unique component. That component is an expression taken from G. Specifically, let each atom of G be expressed in the form $P(\vec{x})\ \{\geq, >\}\ 0$, and represent the set of such $P(\vec{x})$ by $guards(\tau)$. Then the lexicographic component for τ is an expression from $guards(\tau)$.

SAT takes a partial order o among $\mathcal{T}$, and for each $\tau \in \mathcal{T}$, produces the sets $\mathcal{A} = \{\tau\}$ and $\overline{\mathcal{T}} = \{\tau' : \tau = \tau' \vee (\tau' > \tau) \in o\}$ of Definition 12. It checks if for each τ, some expression of $guards(\tau)$ is eventually negative. SEARCH incrementally constructs a linear order o among $\mathcal{T}$. One level of the search first checks the feasibility of the current partial order o. If it is not feasible,

$$\begin{array}{llcl}\tau_1\colon & \{x \geq 0,\ y \leq z^3\} & \Rightarrow & (x,\ y)\ :=\ (x-1,\ y-1)\\ \tau_2\colon & \{x \geq 0,\ y \leq z^3\} & \Rightarrow & (y,\ z)\ :=\ (y-1,\ z+y)\end{array}$$

Fig. 8. Loop CUBE.

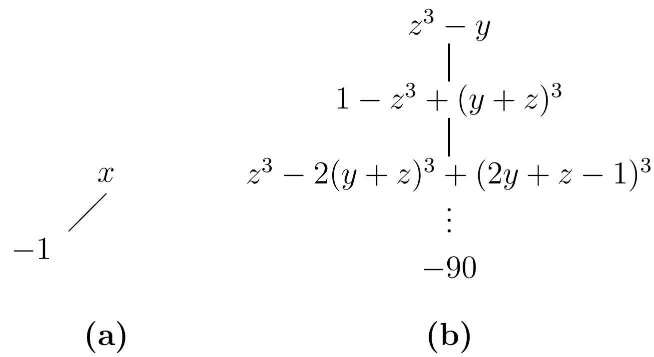


Fig. 9. FDTs in the proof of termination of CUBE.

then no linear extension of this order can induce a satisfiable constraint system, so the current branch terminates. If it is feasible and o is a linear order, then a solution has been found. Otherwise, two unordered components are selected and the function is called recursively on the two alternate orderings of these components. The $\cup$ operation returns the transitive closure.

Example 20 *Consider the polynomial loop in Figure 8. Calling* LEX *returns the lexicographic polyranking function* $\langle x,\ z^3 - y\rangle$ *with map* $\pi : \{\tau_1 \mapsto 1,\ \tau_2 \mapsto 2\}$. *Figure 9 presents the FDTs showing that* **(a)** x *is eventually negative by* $\{\tau_1\} \subseteq \{\tau_1, \tau_2\}$ *and* **(b)** $z^3 - y$ *is eventually negative by* $\{\tau_2\} \subseteq \{\tau_2\}$.

5 A Stronger Analysis

In this section, we define a stronger analysis technique than polyranking for polynomial loops. We now view finite difference trees as objects to be analyzed, rather than as representations of executions of FD_EN. Therefore, we define the construction of the FDT of at most height *max_height* of $P(\overrightarrow{x})$ with respect to $\overline{\mathcal{T}}$ in Figure 10. Finite differences are computed except when the maximum height has been reached or when the given expression is constant with respect to the set of transitions. An expression is constant if the finite difference with respect to every transition is 0.

Example 21 *Consider polynomial loop* INTERACTION *in Figure 11 and the FDT of* x *in Figure 12***(a)***, which is generated by* (FDT x $\{\tau_1, \tau_2\}$ h)*, for* $h \geq 2$. *The leaf at* (τ_2, τ_1) *is the expression* 1*, so that, seen as a trace of* EN, x *is clearly not* eventually negative by $\mathcal{T} \subseteq \mathcal{T}$ *as defined in Definition 12. However, we show that on all nonterminating computations,* x *is eventually negative.*

```
let rec FDT P(x⃗) 𝒯 max_height =
  let MAXIMUM Δ_{τ: G⇒x⃗:=E⃗(x⃗)} P(x⃗) =
    upper bound Δ_τ P(x⃗)
    subject to G
  in
  if max_height = 0 ∨ P(x⃗) is constant with respect to 𝒯 then
  Leaf (MAXIMUM Δ_τ P(x⃗)) else
  Tree (P(x⃗), {FDT Δ_τ P(x⃗) 𝒯 (max_height − 1)}_{τ∈𝒯})
```

Fig. 10. Generation of at most *max_height*-height FDT.

$$\begin{array}{llcl} \tau_1\colon & \{x \geq 0\} \Rightarrow (x,\ y,\ z) & := & (x+z,\ y+1,\ z-2) \\ \tau_2\colon & \{x \geq 0\} \Rightarrow (x,\ y) & := & (x+y,\ y-2) \end{array}$$

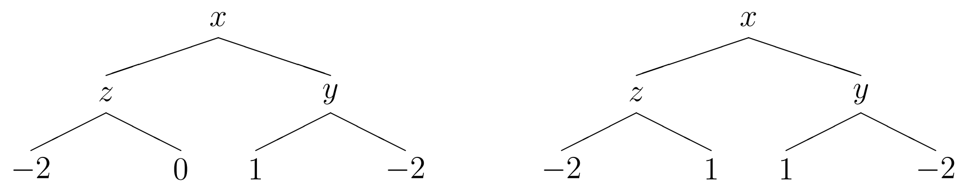
Fig. 11. Polynomial loop INTERACTION.

An FDT t is computed symbolically, but given a computation $\langle \gamma = \tau_{j_0}\tau_{j_1}\tau_{j_2}\ldots,\ \sigma = s_0 s_1 s_2 \ldots\rangle$, we can evaluate the symbolic nodes over each state s_i so that t has value $t(s_i)$ at state s_i. Alternately, we can compute a sequence of *instance trees* from the initial state s_0 and the transition sequence γ. Initially, $t_0 = t(s_0)$. Then the value of each node in instance tree t_{i+1} is given by applying transition τ_{j_i} to tree t_i, where an application of a transition τ_{j_i} to t_i increases each node by the value of its τ_{j_i}-child (applied to each node simultaneously, or starting from the root). For FDT t, always $t_i = t(s_i)$, but this equation does not hold for the variations on FDTs that we discuss below.

The value of $E(\overrightarrow{x})$ at any point in a computation depends not only on the number of times each transition was taken, but also on the *order* in which the transitions were taken. This dependence on order makes direct analysis difficult. Instead, we approximate t by another tree t^+ such that its root expression $E(\overrightarrow{x})^+$ is always an upper bound on $E(\overrightarrow{x})$ over any computation. Specifically, we define *Taylor FDTs* and *partial Taylor FDTs*, which eliminate the dependence on the order in which the transitions are taken. We then show how every FDT can be conservatively approximated by a partial Taylor FDT. To start, we define the *critical nodes* of an FDT, which satisfy a particular property in a partial Taylor FDT.

Definition 22 (Critical Nodes) *In an FDT for $E(\overrightarrow{x})$, the* critical nodes *are nodes $\Delta_T E(\overrightarrow{x})$ such that for all permutations π, $\Delta_{\pi(T)} E(\overrightarrow{x})$ is a constant.*

Definition 23 (Taylor FDT and Partial Taylor FDT) *An FDT of $E(\overrightarrow{x})$ is a* Taylor FDT *if for each sequence of transitions T and every permutation π of T, $\Delta_T E(\overrightarrow{x}) = \Delta_{\pi(T)} E(\overrightarrow{x})$. That is, all n^{th} order finite differences sharing the same multiset of transitions have the same value. An FDT of $E(\overrightarrow{x})$ is a* partial Taylor FDT *if for each critical leaf $\Delta_T E(\overrightarrow{x})$ and permutation π, $\Delta_T E(\overrightarrow{x}) = \Delta_{\pi(T)} E(\overrightarrow{x})$.*



(a) **(b)**

Fig. 12. **(a)** The FDT of INTERACTION and its **(b)** Taylored FDT.

Even if an FDT is not a Taylor or partial Taylor FDT, it is associated with a partial Taylor FDT.

Definition 24 (Taylored FDT) *Given FDT t of $E(\overrightarrow{x})$, the* Taylored FDT *t^+ is a partial Taylor FDT. Each critical leaf $\Delta_T E(\overrightarrow{x})$ of t is given value $\max_\pi \Delta_{\pi(T)} E(\overrightarrow{x})$ in t^+; the rest of t^+ is identical to t.*

For computation $\langle \gamma, \sigma \rangle$ and FDT t, the value of a node in the instance tree t_i^+ is at least that of its counterpart in $t_i = t(s_i)$, for each i. Then always $t_i \leq t_i^+$, where $\leq$ is extended to nodewise comparison, and thus, for the root, always $E(s_i) = E_i \leq E_i^+$.

Example 25 *The Taylored FDT for x over* INTERACTION *is shown in Figure 12***(b)***. Node (τ_1, τ_2) becomes 1 because $\max\{\Delta_{1,2}x, \Delta_{2,1}x\} = \max\{0, 1\} = 1$.*

We extend the definition of a Taylored FDT so that the result is a Taylor FDT; however, the extension can only be computed with respect to the initial value of a computation.

Definition 26 (Fully Taylored FDT) *Given FDT t of $E(\overrightarrow{x})$ and initial value s_0 of a computation $\langle \gamma, \sigma \rangle$, the* fully Taylored FDT *t_f^+ is a Taylor FDT. Each node n at index T in t_f^+ has value $\max_\pi((\Delta_{\pi(T)} E(\overrightarrow{x}))[\overrightarrow{x} \mapsto s_0])$.*

For computation $\langle \gamma, \sigma \rangle$, always $t_i \leq t_i^+ \leq (t_f^+)_i$.

A fully Taylored FDT has the property that for any multiset of transitions T, all finite difference nodes $\Delta_{\pi(T)} E(\overrightarrow{x})$, for permutation π, have the same value. Consequently, the FDT may be analyzed in a way parallel to the analysis of polynomials of multiple variables that vary continuously with time. Specifically, we look at the discrete Taylor expansion around "time" 0 — the beginning of the computation. We will then see that only certain terms are interesting, which will allow us to analyze partial Taylor FDTs.

Suppose that the FDT t for $E(\overrightarrow{x})$ over $\overline{\mathcal{T}}$ is a Taylor FDT, a strong assumption. Introduce one variable z_τ for each transition $\tau \in \overline{\mathcal{T}}$, representing the number of times that τ has been taken. Then the discrete Taylor series expansion expresses the value of $E(\overrightarrow{x})$ at each iteration in terms of the z_τ variables and the initial tree $t_0 = t(s_0)$. A discrete Taylor series is similar to a continuous Taylor series, except that exponentiation a^n is replaced by the *falling factorial* $(a)_n = a(a-1)\cdots(a-n+1)$. For sequence of transitions T, let $S(T)$ be the set of transitions occurring in T and $C(T,\tau)$ be the number of occurrences of τ in T. Then the discrete Taylor series expansion is

$$E(s_i) \;=\; \sum_{\Delta_T E(\overrightarrow{x}) \in t} \frac{\prod_{\tau \in S(T)} (z_\tau)_{C(T,\tau)}}{|T|!} (\Delta_T E(\overrightarrow{x}))[\overrightarrow{x} \mapsto \overrightarrow{s_0}]$$

where each z_τ is equal to the number of times τ has been taken up to time i.

For termination analysis, we care only about eventual trends. Thus, only the *dominant terms* are important. In a polynomial, a term $c_1 \prod x_k^{d_k}$ dominates term $c_2 \prod x_k^{e_k}$ if $d_k \geq e_k$ for each k and $d_{k'} > e_{k'}$ for at least one k'. Examining the expansion, we see that the dominant terms are those of the critical leaves, which are constants. Then for any finite-height FDT t, not just Taylor FDTs, we need only analyze terms corresponding to the critical leaves of the Taylored FDT t^+. These terms comprise the *dominant Taylor expression*.

Definition 27 (Dominant Taylor Expression) *The* dominant Taylor expression *of FDT t is*

$$\sum_{\Delta_T E(\overrightarrow{x}) \in critical\_nodes(t^+)} \frac{\prod_{\tau \in S(T)} z_\tau^{C(T,\tau)}}{|T|!} \Delta_T E(\overrightarrow{x})$$

which is a polynomial in z_τ, since each critical node $\Delta_T E(\overrightarrow{x})$ is a constant.

Example 28 *Consider polynomial loop* INTERACTION. *The dominant Taylor expression of the Taylored FDT for the expression x is*

$$-2 \cdot \frac{1}{2!} z_1 z_1 + \frac{1}{2!} z_1 z_2 + \frac{1}{2!} z_2 z_1 - 2 \cdot \frac{1}{2!} z_2 z_2 = -z_1^2 + z_1 z_2 - z_2^2.$$

The nonnegative term $z_1 z_2$ arises from the adverse interaction of τ_1 and τ_2.

Combining the result $t_i \leq t_i^+ \leq (t_f^+)_i$ with the dominant Taylor expression admits analysis of the evolution of $E(\overrightarrow{x})$.

Theorem 29 (Taylor Eventually Negative) *Consider loop $L : \langle \overrightarrow{x}, \theta, \mathcal{T} \rangle$, expression $E(\overrightarrow{x})$, and sets of transitions $\mathcal{A} \subseteq \overline{\mathcal{T}}$, where $\mathcal{A} \neq \emptyset$ and $\overline{\mathcal{T}} \subseteq \mathcal{T}$. Suppose that for each $\mathcal{I} \subseteq \overline{\mathcal{T}}$ such that $\mathcal{I} \cap \mathcal{A} \neq \emptyset$, the dominant Taylor expression of the FDT t of some height of $E(\overrightarrow{x})$ with respect to $\mathcal{I}$ decreases without bound as the number of iterations increases. Then on any infinite* *computation $\langle\gamma,\sigma\rangle$ for which $io(\gamma) \subseteq \overline{\mathcal{T}}$ and $io(\gamma) \cap \mathcal{A} \neq \emptyset$, $E(\overrightarrow{x})$ eventually becomes negative and stays negative.*

Proof. Each set $\mathcal{I}$ represents a possible set of transitions that are taken infinitely often, and thus also a set of infinite computations $S(\mathcal{I}) \stackrel{\text{def}}{=} \{\langle\gamma,\sigma\rangle : io(\gamma) = \mathcal{I}\}$. Consider one $\mathcal{I}$. Suppose the dominant Taylor expression of t with respect to $\mathcal{I}$ decreases without bound as the number of iterations increases. Consider a computation $\langle\gamma,\sigma\rangle \in S(\mathcal{I})$. A fully Taylored FDT t_f^+ is determined by s_0 of σ. By selection, $io(\gamma) = \mathcal{I}$, so that the dominant Taylor expression really dominates the Taylor series expansion of the root of t_f^+. Therefore, the root $(E_f^+)_i$ decreases without bound. But $(E_f^+)_i \geq E_i^+ \geq E_i$, so E_i also decreases without bound. Since this conclusion holds for all $\mathcal{I}$, and $\cup_{\mathcal{I}:\ \mathcal{I}\subseteq\overline{\mathcal{T}}\wedge\mathcal{I}\cap\mathcal{A}\neq\emptyset} S(\mathcal{I}) = \{\langle\gamma,\sigma\rangle : io(\gamma) \subseteq \overline{\mathcal{T}} \wedge io(\gamma) \cap \mathcal{A} \neq \emptyset\}$, then $E(\overrightarrow{x})$ must eventually become negative. □

The following example introduces a technique for showing that a dominant Taylor expression is decreasing. Changing to polar coordinates allows an expression that increases with each iteration to appear explicitly in the dominant Taylor expression.

Example 30 *Recall that the dominant Taylor expression for x with respect to $\{\tau_1,\tau_2\}$ in* INTERACTION *is $-z_1^2 + z_1z_2 - z_2^2$. Call the expression $\sqrt{z_1^2 + z_2^2}$ the* absolute length *of a computation (in which both transitions are taken infinitely often). Since z_1 and z_2 express the number of times τ_1 and τ_2 have been taken, the absolute length is initially 0 and grows with each iteration. If the dominant Taylor expression decreases without bound as the absolute length of the computation increases, then the assumption of the Theorem is satisfied for $\mathcal{I} = \{\tau_1,\tau_2\}$.*

Let $z_1 = r\cos\theta$ and $z_2 = r\sin\theta$. r corresponds to the absolute length, while $\theta \in [0,\frac{\pi}{2}]$ expresses the ratio of z_2 to z_1. Then after a change of variables,

$$-z_1^2 + z_1z_2 - z_2^2 = -r^2\cos^2\theta + r^2\cos\theta\sin\theta - r^2\sin^2\theta = r^2(\cos\theta\sin\theta - 1).$$

Call this expression $Q(r,\theta)$. Differentiating, we find

$$\frac{\partial Q}{\partial r} = 2r(\cos\theta\sin\theta - 1) \quad \textit{and} \quad \frac{\partial^2 Q}{\partial r^2} = 2(\cos\theta\sin\theta - 1).$$

The relevant domain of θ is $[0,\frac{\pi}{2}]$, over which the maximum of $\frac{\partial^2 Q}{\partial r^2}$ is -1, occurring at $\theta = \frac{\pi}{4}$. Therefore, independently of θ, as r increases, $Q(r,\theta)$ eventually decreases without bound; thus, the dominant Taylor expression also eventually decreases without bound.

Finally, considering the case where only τ_1 (τ_2) is taken after a certain point, we note that the dominant Taylor expression is $-z_1^2$ ($-z_2^2$), which decreases

(a)

```
while(i < a.n || j < b.n) {
  if (i >= a.n)              c.e[c.n++] = b.e[j++];
  else if (j >= b.n)         c.e[c.n++] = a.e[i++];
  else if (a.e[i] <= b.e[j]) c.e[c.n++] = a.e[i++];
  else                       c.e[c.n++] = b.e[j++];
}
```

(b)

$$\begin{array}{llcll}
\tau_1\colon & \{j < b.n,\ i < a.n\} & \Rightarrow & (c.n,\ j) & := (c.n+1,\ j+1) \\
\tau_2\colon & \{j < b.n,\ i < a.n\} & \Rightarrow & (c.n,\ i) & := (c.n+1,\ i+1) \\
\tau_3\colon & \{j \geq b.n,\ i < a.n\} & \Rightarrow & (c.n,\ i) & := (c.n+1,\ i+1) \\
\tau_4\colon & \{j < b.n,\ i \geq a.n\} & \Rightarrow & (c.n,\ j) & := (c.n+1,\ j+1)
\end{array}$$

Fig. 13. **(a)** Imperative loop in C and **(b)** the corresponding polynomial loop.

without bound. Thus, INTERACTION *terminates on all input, as x is Taylor eventually negative by $\{\tau_1, \tau_2\} \subseteq \{\tau_1, \tau_2\}$.*

We can apply the trick of using the absolute length and changing to polar coordinates in general, via the usual extension of polar coordinates to higher dimensions. For m transitions and expression $Q(r, \theta_1, \ldots, \theta_{m-1})$, we check if $\frac{\partial^n Q}{\partial r^n}$ is everywhere at most some negative constant over $\theta_i \in [0, \frac{\pi}{2}]$, $i \in [1..m-1]$, where $\frac{\partial^n Q}{\partial r^n}$ is the first derivative with respect to r that is constant with respect to r.

Compared to the polyranking technique, the FDT analysis is more robust against negative interference between transitions, but at greater computational cost.

6 Experimental Results

To test the applicability of our termination analysis, we implemented a C loop abstracter in CIL (19) and the polyranking-based termination analysis in O'Caml. The purpose of the loop abstracter is to extract a set of polynomial loops from a C loop with arbitrary control flow, including embedded loops. The termination analysis is then applied to the extracted loops. The analysis is sound up to alias analysis, modification of variables by called functions, and issues like unsigned casts and overflow. We chose to ignore these factors in our experimentation, as they have no bearing on the scalability of the actual termination analysis.

Example 31 *The loop in Figure 13*(**a**) *merges two lists. The abstracted polynomial loop is shown in Figure 13*(**b**)*; four transitions with infeasible guards were pruned. The analysis proves that the loop terminates.*

Table 1
Results of analysis. Legend: **LOC**: lines of code *of files successfully parsed and containing loops*, as measured by `wc`; **# L**: total number of analyzed loops; **# A**: number of loops successfully abstracted; **# P**: number of (abstracted) loops proved terminating; **% P/A**: percentage of abstracted loops proved terminating; **% P/L**: percentage of total loops proved terminating; **Time**: total time in seconds required to analyze the program. `small1` requires a maximum FDT height of 4; data for all others are for a maximum height of 1.

Name	LOC	# L	# A	# P	% P/A	% P/L	Time (s)
`small1`	310	8	6	4	66	50	4
`vector`	361	13	13	12	92	92	3
`serv`	457	9	6	5	83	55	4
`dcg`	1K	55	53	53	100	96	4
`bcc`	4K	70	18	18	100	25	6
`sarg`	7K	122	26	25	96	20	102
`spin`	19K	652	132	119	90	18	29
`meschach`	28K	896	803	770	95	85	40
`f2c`	30K	434	114	96	84	22	41
`ffmpeg1`	33K	453	270	214	79	47	45
`gnuplot`	50K	825	329	298	90	36	106
`gaim`	57K	605	60	52	86	8	97
`ffmpeg2`	75K	2216	1945	1856	95	83	112

We applied the analysis to several open-source projects from Netlib (20) and Sourceforge (23). The results of the analysis are summarized in Table 1. These programs span a range of applications: for example, `f2c` converts FORTRAN source to C source; `spin` is a model checker; and `meschach` is a package of numerical algorithms.

Many of the loops are obviously terminating. In some sense, this observation is disappointing: of what value is our analysis when the reasons are trivial? Three points come to mind. First, applying any analysis at all is useful. Programmers regularly write loops with complicated control structure that span several editor pages. Verifying manually that all paths, say, increment a counter (and the right counter) is thus tedious and ineffective. An automated analysis filters out correct cases; remaining loops warrant a second look. Second, our analysis scales well to triviality: the FDTs are shallow (of depth one for the counter case), thus requiring an insignificant amount of time. Finally, compared to a naive syntactic analysis, our approach has two advantages. First, a syntactic analysis might be sensitive to the presentation of the loop. For example, a syntactic analysis could stumble on a `while` loop that terminates using `break` or `goto` statements. Our abstraction and analysis approach not only is insensitive to such presentations of loops, it can also identify other loop guards than the one explicitly provided by the `for` or `while` statement. Second, termination of a minority of proofs is less than trivial. For example, the `meschach` source contains loops with terminating behavior similar to that in Figure 13. Our analysis easily handles such cases.

Reasons for failed proofs are numerous. A failure to abstract a nontrivial guarded set indicates possible nontermination, but usually arises because the termination behavior is not number-related. Even "successful" abstractions may present only incidental information; termination may rest on other criteria, such as the appearance of a special character in a string. In several cases, we noted that the lack of invariant analysis weakens the termination analysis.

7 Conclusion

We described a termination analysis based on finding polyranking functions, which subsume ranking functions. A tuple of functions is considered a lexicographic polyranking function if its component functions are proved to be eventually negative with respect to certain transition sets. While the concept of polyranking is general, we specialized it to the case of polynomial loops, for which finite difference arithmetic provides an efficient implementation of checking if an expression is eventually negative. Finally, we discussed a second analysis that is stronger than polyranking, but computationally more expensive.

Polynomial loops provide an expressive language for abstracting real code. Although termination for this class of loops is not semi-decidable — even when restricted to affine expressions — we show that the polyranking-based analysis is effective in practice. This analysis is notable for two reasons. First, it is applicable to polynomial, rather than just linear, expressions and assignments. Second, our analysis naturally scales to the difficulty of the problem, which enables our prototype implementation to analyze tens of thousands of lines of C in seconds.

References

[1] Boyd, S., and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2005.

[2] Bradley, A. R., Manna, Z., and Sipma, H. B. Linear ranking with reachability. In *CAV* (2005). To appear.

[3] Bradley, A. R., Manna, Z., and Sipma, H. B. The polyranking principle. In *ICALP* (2005). To appear.

[4] Bradley, A. R., Manna, Z., and Sipma, H. B. Termination analysis of integer linear loops. In *CONCUR* (2005). To appear.

[5] Bradley, A. R., Manna, Z., and Sipma, H. B. Termination of polynomial programs. In *VMCAI* (2005), pp. 113–129.

[6] Codish, M., Genaim, S., Bruynooghe, M., Gallagher, J., and Vanhoof, W. One lop at a time. In *WST* (2003).

[7] Cohen, J. Computer-assisted microanalysis of programs. *Comm. ACM 25*, 10 (1982).

[8] Collins, G. E. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *GI Conf. Automata Theory and Formal Languages* (1975), pp. 515–532.

[9] Colón, M., Sankaranarayanan, S., and Sipma, H. Linear invariant generation using non-linear constraint solving. In *CAV* (2003), pp. 420–433.

[10] Colón, M., and Sipma, H. Synthesis of linear ranking functions. In *TACAS* (2001), pp. 67–81.

[11] Colón, M., and Sipma, H. Practical methods for proving program termination. In *CAV* (2002), pp. 442–454.

[12] Cousot, P. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *VMCAI* (2005), pp. 1–24.

[13] 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.

[14] Floyd, R. W. Assigning meaning to programs. In *Proc. of a Symposium in Applied Mathematics* (1967), vol. 19, A.M.S., pp. 19–32.

[15] H. B. Sipma, T. E. Uribe, and Z. Manna. Deductive model checking. In *CAV* (1996), pp. 209–219.

[16] Katz, S., and Manna, Z. Logical analysis of programs. *Comm. ACM 19*, 4 (1976).

[17] Lee, C. S., Jones, N. D., and Ben-Amram, A. M. The size-change principle for program termination. In *POPL* (2001).

[18] Manna, Z., Browne, A., Sipma, H., and Uribe, T. E. Visual abstractions for temporal verification. In *Algebraic Methodology and Software Technology* (1998), pp. 28–41.

[19] Necula, G. C., McPeak, S., Rahul, S. P., and Weimer, W. CIL: Intermediate language and tools for analysis and transformation of C programs. In *Proceedings of Conf. on Compiler Construction* (2002).

[20] *Netlib Repository*, 2004. (`http://www.netlib.org`).

[21] Podelski, A., and Rybalchenko, A. A complete method for the synthesis of linear ranking functions. In *VMCAI* (2004), pp. 239–251.

[22] Podelski, A., and Rybalchenko, A. Transition invariants. In *LICS* (2004), pp. 32–41.

[23] *SourceForge*, 2004. (`http://sourceforge.net`).

[24] Wegbreit, B. Mechanical program analysis. *Comm. ACM 18*, 9 (1975).