Linear Ranking with Reachability

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. We present a complete method for synthesizing *lexicographic linear ranking functions* supported by inductive linear invariants for loops with linear guards and transitions. Proving termination via linear ranking functions often requires invariants; yet invariant generation is expensive. Thus, we describe a technique that discovers just the invariants necessary for proving termination. Finally, we describe an implementation of the method and provide extensive experimental evidence of its effectiveness for proving termination of C 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 [19, 11, 15]. We propose a termination analysis of linear loops based on the synthesis of lexicographic linear ranking functions supported by linear invariants.

The method exploits the *constraint-based* approach to static analysis. In constraint-based analysis, a property *template* is constrained by high-level conditions that describe the desired property. These high-level conditions induce a constraint system; its solutions are valid instantiations of the template. For example, [3] describes how to generate linear invariants of linear transition systems via the constraint-based approach. In contrast to classical symbolic simulation, constraint-based analyses are not limited to invariants; for example, [4, 5] describes how to apply the constraint-based approach to generate linear ranking functions to prove loop termination.

Our approach to termination analysis has two distinct features over previous work on ranking function synthesis and constraint-based analysis. First, we combine the generation of ranking functions with the generation of invariants to form one constraint solving problem. This combination makes invariant generation *implicit*: the necessary *supporting* invariants for the ranking function are

* 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.

discovered on demand. In [4, 5], the generation of ranking functions was supported by independently generated invariants. Second, we take constraint-based analysis one step further in combining *structural* constraints with *numeric* constraints, which allows us to specify a set of templates for the ranking function components and solve for a *lexicographic* linear ranking function. The numeric constraints arise, as usual, from placing conditions on the template coefficients. The structural constraints arise from requiring a lexicographic ordering among the ranking function components, which constrains the conditions that should be applied. Consequently, our solving strategy alternates between adding or removing ordering constraints and solving the resulting numeric constraint systems. We propose two main components to the solving strategy: first, the numeric constraint solver quickly solves the special form of *parametric linear constraints* that arise; second, the higher-level solver searches for a lexicographic ranking function. The latter is theoretically complete, in that if a lexicographic ordering exists, it will be found; moreover, it is fast in practice.

Automatic synthesis of linear ranking functions has received a fair amount of attention. In [8], Katz and Manna show how to generate constraint systems over loops with linear assertional guards and linear assignments for which solutions are linear ranking functions. Synthesis of linear ranking functions over linear loops with multiple paths and assertional transition relations is accomplished via polyhedral manipulation in [4, 5]. In [14], Podelski and Rybalchenko specialize the technique to single-path linear loops without an initial condition, providing an efficient and complete synthesis method based on linear programming. We generalize these previous efforts.

Automatic synthesis of ranking functions is applicable in at least two ways. First, it can serve as the main component of an automated program analysis. We provide empirical evidence of its effectiveness at analyzing loops that arise in C programs. Second, it can support theoretically complete verification frameworks like *verification diagrams* [11] and *transition invariants* [15] (see also [6, 9, 1] for related ideas). In this context, the user guides the construction of the proof with support from automatic invariant generation and ranking 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 ranking functions.

The rest of the paper is organized as follows. Section 2 introduces basic concepts and mathematical tools. Section 3 presents the constraint generation

```
int gcd(int y1 > 0, int y2 > 0) :
  while y1 ≠ y2 do
    if y1 > y2 then
      y1 := y1 − y2
    else
      y2 := y2 − y1
  return y1
```
(a)

$$\Theta : \{y_1 \geq 1,\ y_2 \geq 1\}$$
$$\tau_1 : \{y_1 \geq y_2 + 1\} \Rightarrow \{y_1' = y_1 - y_2,\ y_2' = y_2\}$$
$$\tau_2 : \{y_2 \geq y_1 + 1\} \Rightarrow \{y_2' = y_2 - y_1,\ y_1' = y_1\}$$

(b)

Fig. 1. (a) Program GCD for finding the greatest common divisor of two positive integers. **(b)** Path-sensitive abstraction to set of guarded commands.

technique. Section 4 describes the method for synthesizing lexicographic functions in practice, while Section 5 discusses our approach to solving the generated numeric constraint systems. Section 6 presents empirical data from applying our analysis to C programs. Section 7 concludes.

2 Preliminaries

We first define our loop abstraction based on sets of linear transitions. Subsequently, we formalize ranking functions and inductive invariants in the context of loops. Finally, we present Farkas's Lemma.

Definition 1 (Linear Assertion) An *atom* over a set of variables $\mathcal{V} : \{x_1, \ldots, x_m\}$ is a linear inequality over $\mathcal{V}$: $a_1x_1 + a_2x_2 + \cdots + a_mx_m + b \geq 0$. Letting $\mathbf{x} = (x_1, x_2, \ldots, x_m, 1)^{\mathrm{T}}$, we also write $\mathbf{a}^{\mathrm{T}}\mathbf{x} \geq 0$, where $\mathbf{a}_{m+1} = b$. A linear assertion over $\mathcal{V}$ is a conjunction of linear inequalities over the program variables, written (as a matrix in homogenized form) $\mathbf{A}\mathbf{x} \geq 0$.

Definition 2 (Guarded Command) A *guarded command* $\tau(\mathcal{V}) : g(\mathcal{V}) \Rightarrow u(\mathcal{V}, \mathcal{V}')$ over a set of variables $\mathcal{V}$ consists of a guard $g(\mathcal{V})$, which is a linear assertion over $\mathcal{V}$, and an update $u(\mathcal{V}, \mathcal{V}')$, which is a linear assertion over primed and unprimed variables in $\mathcal{V}$. Letting $(\mathbf{x}\mathbf{x}') = (x_1, x_2, \ldots, x_m, x_1', x_2', \ldots, x_m', 1)^{\mathrm{T}}$, we also write $\tau(\mathbf{x}\mathbf{x}') \geq 0$.

Definition 3 (Loop) A *loop* $L : \langle G,\ \Theta\rangle$ over $\mathcal{V}$ consists of a set of guarded commands G over $\mathcal{V}$ and an initial condition Θ, a linear assertion over $\mathcal{V}$. We sometimes write $\Theta\mathbf{x} \geq 0$, viewing Θ as a homogenized matrix of coefficients.

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

Example 1. Consider the program GCD in Figure 1**(a)**, which computes the greatest common divisor of two positive integers [10]. Figure 1**(b)** presents the program as a *loop*. Strict inequalities are translated into weak inequalities based on the integer type of the variables.

To show that a loop is terminating, it is sufficient to exhibit a *ranking function*. In this paper we focus on lexicographic linear ranking functions.

Definition 5 (Linear Ranking Function) A *linear ranking function* for a loop $L : \langle G,\ \Theta \rangle$ over $\mathcal{V}$ is a linear expression $e :\ r_1 x_1 + \ldots + r_m x_m + r_{m+1}$, or $\mathbf{r}^{\mathrm{T}}\mathbf{x}$ (letting $\mathbf{x} = (x_1, x_2, \ldots, x_m, 1)^{\mathrm{T}}$ as usual), such that

(Bounded) $L \models\ \mathbf{r}^{\mathrm{T}}\mathbf{x} \geq 0$;
(Ranking) $L \models\ \tau(\mathbf{x}\mathbf{x}') \geq 0 \rightarrow \mathbf{r}^{\mathrm{T}}\mathbf{x} - \mathbf{r}^{\mathrm{T}}\mathbf{x}' \geq \epsilon$ for some $\epsilon > 0$ and all $\tau \in G$.

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

(Bounded) $L \models\ \tau(\mathbf{x}\mathbf{x}') \geq 0 \rightarrow \mathbf{r_i}^{\mathrm{T}}\mathbf{x} \geq 0$;
(Ranking) $L \models\ \tau(\mathbf{x}\mathbf{x}') \geq 0 \rightarrow \mathbf{r_i}^{\mathrm{T}}\mathbf{x} - \mathbf{r_i}^{\mathrm{T}}\mathbf{x}' \geq \epsilon$;
(Unaffecting) for $j < i$, $L \models\ \tau(\mathbf{x}\mathbf{x}') \geq 0 \rightarrow \mathbf{r_j}^{\mathrm{T}}\mathbf{x} - \mathbf{r_j}^{\mathrm{T}}\mathbf{x}' \geq 0$.

For some loops, invariants are necessary to prove that a function is actually a ranking function for the loop; we say such invariants are *supporting* invariants. While in theory, supporting invariants may be of any type, we focus on linear invariants in this paper.

Definition 7 (Linear Inductive Invariant) A (homogenized) linear assertion $\mathbf{I}\mathbf{x} \geq 0$ is an inductive invariant of a loop $L : \langle G,\ \Theta \rangle$ if it holds at the start of the loop and it is preserved by all guarded commands. Formally, a *linear inductive invariant* satisfies the following conditions:

(Initiation) $\Theta\mathbf{x} \geq 0 \rightarrow \mathbf{I}\mathbf{x} \geq 0$;
(Consecution) for every $\tau \in G$, $\mathbf{I}\mathbf{x} \geq 0 \wedge \tau(\mathbf{x}\mathbf{x}') \geq 0 \rightarrow \mathbf{I}\mathbf{x}' \geq 0$.

As we are working only with linear transitions, linear ranking functions, and linear invariants, Farkas's Lemma [18] will both justify completeness claims and serve as a tool for encoding and solving conditions for synthesizing ranking functions.

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 $c_1 x_1 + \cdots + c_m x_m + c_{m+1} \geq 0$ iff there exist real numbers $\lambda_1, \ldots, \lambda_n \geq 0$ such that

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

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

For notational convenience we represent applications of Farkas's Lemma by means of assertions in homogenized form as before, without explicitly mentioning the multipliers λ. Thus, for example, we write

$$\frac{\mathbf{A}\mathbf{x} \geq 0}{\mathbf{C}\mathbf{x} \geq 0} \qquad \text{where} \qquad \mathbf{x} = (x_1, x_2, \ldots, x_m, 1)^{\mathrm{T}}.$$

3 Constraint Generation

Our approach for finding (lexicographic) ranking functions is based on a *template* ranking function, consisting of a (set of) linear expression(s) over $\mathcal{V}$ with unknown coefficients, and a *template* supporting invariant, consisting of an n-conjunct linear assertion with $n(|\mathcal{V}|+1)$ unknown coefficients. Conditions on the templates ensure that any solution to the induced constraint system represents a ranking function and its supporting invariants. Farkas's Lemma provides a means of translating some high-level conditions to numeric constraints. Structural conditions (*i.e.*, ordering constraints among components) are enforced via the specific set of high-level conditions that are applied.

Definition 8 (Template Expression) A *template expression* over $\mathcal{V}$ is a linear expression $c_1 x_1 + \cdots + c_m x_m + c_{m+1}$, or $\mathbf{c}^{\mathrm{T}}\mathbf{x}$, with unknown coefficients $\mathbf{c}$. A *template assertion* is a linear assertion $\bigwedge_i \mathbf{c_i}^{\mathrm{T}}\mathbf{x} \geq 0$, or $\mathbf{C}\mathbf{x} \geq 0$.

The application of 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 $L : \langle G,\ \Theta\rangle$, the supporting invariant template $\mathbf{I}\mathbf{x} \geq 0$, and lexicographic ranking function template $\{\mathbf{c_1}^{\mathrm{T}}\mathbf{x},\ \ldots, \mathbf{c_n}^{\mathrm{T}}\mathbf{x}\}$, the following Farkas's Lemma *specializations* are applied to encode the appropriate conditions.

(Initiation) The supporting invariant includes the initial condition.

$$\mathbb{I} \stackrel{\text{def}}{=} \frac{\Theta\mathbf{x} \ \geq \mathbf{0}}{\mathbf{I}\mathbf{x} \ \ \geq \mathbf{0}}$$

(Disabled) Transition $\tau_i \in G$ and the invariant may contradict each other, indicating "dead code."

$$\mathbb{D}_i \stackrel{\text{def}}{=} \begin{array}{ll} \mathbf{I}\mathbf{x} & \geq \mathbf{0} \\ \tau_i(\mathbf{x}\mathbf{x}') & \geq \mathbf{0} \\ \hline -1 & \geq \mathbf{0} \end{array}$$

(Consecution) For transition $\tau_i \in G$, if the invariant holds and the transition is taken, then the invariant holds in the next state.

$$\mathbb{C}_i \stackrel{\text{def}}{=} \begin{array}{ll} \mathbf{I}\mathbf{x} & \geq \mathbf{0} \\ \tau_i(\mathbf{x}\mathbf{x}') & \geq \mathbf{0} \\ \hline \mathbf{I}\mathbf{x}' & \geq \mathbf{0} \end{array}$$

(Bounded) For transition $\tau_i \in G$ and ranking function component $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, the invariant and transition imply the nonnegativity of the ranking function component.

$$\mathbb{B}_{ij} \quad \stackrel{\text{def}}{=} \quad \frac{\begin{array}{ll} \mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0} \end{array}}{\begin{array}{ll} \mathbf{c_j}^{\mathrm{T}}\mathbf{x} & \geq \mathbf{0} \end{array}}$$

(Ranking) For transition $\tau_i \in G$ and ranking function component $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, taking the transition decreases the linear ranking function by at least some positive amount ($\epsilon > 0$).

$$\mathbb{R}_{ij} \quad \stackrel{\text{def}}{=} \quad \frac{\begin{array}{ll} \mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0} \end{array}}{\begin{array}{ll} \mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' - \epsilon & \geq \mathbf{0} \end{array}}$$

(Unaffecting) For transition $\tau_i \in G$ and ranking function component $\mathbf{c_j}^{\mathrm{T}}\mathbf{x}$, taking the transition does not increase the linear ranking function.

$$\mathbb{U}_{ij} \quad \stackrel{\text{def}}{=} \quad \frac{\begin{array}{ll} \mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0} \end{array}}{\begin{array}{ll} \mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' & \geq \mathbf{0} \end{array}}$$

This specialization is used only if $n > 1$ — *i.e.*, if the ranking function template is lexicographic.

The specializations are applied according to the definitions of inductive invariants and ranking functions discussed in Section 2. In Section 4, we describe an effective algorithm for finding lexicographic ranking functions in practice.

Theorem 2. *Loop $L : \langle G,\ \Theta\rangle$ has a ℓ-lexicographic linear ranking function supported by an n-conjunct inductive linear invariant iff the constraint system generated by*

$$\mathbb{I} \ \wedge \ \bigwedge_{\tau_i \in G} \big(\mathbb{D}_i \vee \big(\mathbb{C}_i \wedge \mathbb{B}_{i,\sigma(i)} \wedge \mathbb{R}_{i,\sigma(i)}\big)\big) \ \wedge \ \bigwedge_{\tau_i \in G, j < \sigma(i)} (\mathbb{D}_i \vee \mathbb{U}_{ij})$$

is satisfiable for some σ mapping transition indices to lexical component indices $\{1, \ldots, \ell\}$.

Corollary 1. *Loop $L : \langle G,\ \Theta\rangle$ has a linear ranking function supported by an n-conjunct inductive linear invariant iff the constraint system generated by*

$$\mathbb{I} \ \wedge \ \bigwedge_{\tau_i \in G} (\mathbb{D}_i \vee (\mathbb{C}_i \wedge \mathbb{B}_i \wedge \mathbb{R}_i))$$

is satisfiable.

These claims follow from the completeness of Farkas's Lemma and the definitions of inductive invariants and ranking functions.

The above formulation of the constraint system assumes a single n-conjunct invariant supporting all lexical components. An alternate formulation allows each component to have its own n-conjunct invariant. Specifically, the constraint system may be split into ℓ constraint systems, one for each component j:

$$\mathbb{I} \ \wedge \bigwedge_{\tau_i \in G} (\mathbb{D}_i \vee \mathbb{C}_i) \ \wedge \bigwedge_{\tau_i \in G : \sigma(i) = j} (\mathbb{D}_i \vee (\mathbb{B}_{ij} \wedge \mathbb{R}_{ij})) \ \wedge \bigwedge_{\tau_i \in G : \sigma(i) > j} (\mathbb{D}_i \vee \mathbb{U}_{ij})$$

This separation has two advantages. First, in practice, the multiple smaller systems are easier to solve than the equivalent large system. Second, the inductive invariant template may be instantiated differently for each system, so that a smaller template is sometimes possible. Consequently, each condition contributes constraints over fewer variables to the generated constraint systems.

Example 2. Consider finding a two-lexicographic linear ranking function for GCD, with a two-conjunct supporting invariant. Choosing the mapping σ determines the form of the conditions on the ranking function template $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ and the invariant template $\mathbf{I}\mathbf{x} \geq 0$. For $\sigma(1) = 2$, $\sigma(2) = 1$, we obtain $\mathbb{I} \wedge \mathbb{C}_1 \wedge \mathbb{C}_2 \wedge \mathbb{B}_{1,2} \wedge \mathbb{B}_{2,1} \wedge \mathbb{R}_{1,2} \wedge \mathbb{R}_{2,1} \wedge \mathbb{U}_{1,1}$, where we omit the *disabled* cases, as we often do in practice, so that the dual constraint system is conjunctive. $\mathbb{C}_1$ and $\mathbb{R}_{1,2}$ expand to the following:

$$\mathbb{C}_1 : \begin{array}{l} i_{1,1}y_1 + i_{1,2}y_2 + i_{1,3} \geq 0 \\ i_{2,1}y_1 + i_{2,2}y_2 + i_{2,3} \geq 0 \\ y_1 \geq y_2 + 1 \\ y_1' = y_1 - y_2 \\ y_2' = y_2 \\ \hline i_{1,1}y_1' + i_{1,2}y_2' + i_{1,3} \geq 0 \\ i_{2,1}y_1' + i_{2,2}y_2' + i_{2,3} \geq 0 \end{array} \qquad \mathbb{R}_{1,2} : \begin{array}{l} i_{1,1}y_1 + i_{1,2}y_2 + i_{1,3} \geq 0 \\ i_{2,1}y_1 + i_{2,2}y_2 + i_{2,3} \geq 0 \\ y_1 \geq y_2 + 1 \\ y_1' = y_1 - y_2 \\ y_2' = y_2 \\ \hline c_{2,1}y_1 + c_{2,2}y_2 \geq c_{2,1}y_1' + c_{2,2}y_2' + \epsilon \end{array}$$

Farkas's Lemma dualizes each component of the condition to produce one numerical constraint system over the λ-multipliers and the unknown template coefficients. See Section 5 for details on the form and solution of such constraint systems. Solving the system that arises here reveals the two-component ranking function $\langle y_2 - 2,\ y_1 - 2\rangle$, supported by the invariant $y_1 \geq 1 \ \wedge \ y_2 \geq 1$, which proves termination for GCD.

If we split the constraint system, as described above, into two constraint systems induced by $\mathbb{I} \wedge \mathbb{C}_1 \wedge \mathbb{C}_2 \wedge \mathbb{B}_{1,2} \wedge \mathbb{R}_{1,2} \wedge \mathbb{U}_{1,1}$ and $\mathbb{I} \wedge \mathbb{C}_1 \wedge \mathbb{C}_2 \wedge \mathbb{B}_{2,1} \wedge \mathbb{R}_{2,1}$, then the solution produces the same ranking function $\langle y_2 - 2,\ y_1 - 2\rangle$. However, only a one-conjunct template invariant is needed, as the first component only requires the invariant $y_1 \geq 1$, while the second only requires $y_2 \geq 1$.

4 Lexicographic Linear Ranking Functions

Theorem 2 requires finding a σ mapping transition indices to lexical component indices. While the number of possible σ's is finite — as only at most $|G|$ lexicographic components are required — it is exponentially large in the number of

```
bool has_llrf(L, n) :          bool rec llrf_{L,n}(o) :
  llrf_{L,n}({})                 if sat_{L,n}(o) then
                                   if complete(o) then return true
                                   (i, j) := choose_unordered(o)
                                   return llrf_{L,n}(o ∪ (i < j)) or llrf_{L,n}(o ∪ (i > j))
                                 else return false
```

Fig. 2. Search strategy for finding a lexicographic linear ranking function, supported by an n-conjunct inductive linear invariant, for loop L.

transitions. An implementation could enumerate the possible σ's to find one that induces a satisfiable constraint system, but this option is not practically feasible, nor is it theoretically satisfying. Not only does this approach take too long on loops for which a ranking function exists, but its worst-case input — loops for which no solution exists — occurs frequently in real code. In this section, we describe an effective procedure for deciding whether a loop has a lexicographic linear ranking function.

Each transition is assigned its own template component in the lexicographic function so that the discovered function has $|G|$ components, ensuring completeness (although the final function may be compressible). Each transition and its component induces a *bounded* condition and a *ranking* condition. Furthermore, the transitions and proposed invariant template induce *initiation* and *consecution* (and possibly *disabled*) conditions. All of these conditions are independent of the order of the lexicographic components; only the *unaffecting* conditions remain to be determined.

Figure 2 outlines the algorithm. Function *has_llrf* indicates if a loop $L : \langle G,\ \Theta \rangle$ has a lexicographic linear ranking function supported by an n-conjunct inductive linear invariant. The function $llrf_{L,n}$ takes as input an *order relation* o, which describes the known ordering constraints between components. Initially, this relation is the empty relation. First, $llrf_{L,n}$ checks the feasibility of the current ordering o. The function $sat_{L,n}$ generates the constraint system induced by L, n, and o, returning whether the system is satisfiable. If it is not, then no possible completion of this ordering can induce a satisfiable constraint system, so the search on the current branch terminates. If the constraint system is satisfiable and the order relation is complete, a solution is feasible and the function returns. If the order is not complete, 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.

Proposition 1. *Assuming that* $sat_{L,n}$ *is a decision procedure, has_llrf*$(L,\ n)$ *returns* **true** *iff loop* $L : \langle G,\ \Theta \rangle$ *has a lexicographic linear ranking function supported by an* n*-conjunct inductive linear invariant.*

Example 3. Since each transition is assigned its own template component, we drop the second index of $\mathbb{B}$ and $\mathbb{R}$. Returning to the GCD program of Example 2, the search first generates the constraint systems induced by $\mathbb{I} \wedge \mathbb{C}_1 \wedge \mathbb{C}_2 \wedge \mathbb{B}_1 \wedge \mathbb{R}_1$ and $\mathbb{I} \wedge \mathbb{C}_1 \wedge \mathbb{C}_2 \wedge \mathbb{B}_2 \wedge \mathbb{R}_2$ and checks their feasibility. Finding both systems satisfiable, it adds (arbitrarily) the ordering constraint $1 > 2$ between indices 1 and 2, which results in the constraint systems induced by $\mathbb{I}\wedge\mathbb{C}_1\wedge\mathbb{C}_2\wedge\mathbb{B}_1\wedge\mathbb{R}_1\wedge\mathbb{U}_{1,2}$ and $\mathbb{I}\wedge\mathbb{C}_1\wedge\mathbb{C}_2\wedge\mathbb{B}_2\wedge\mathbb{R}_2$. Both systems are found satisfiable and, since $\{1 > 2\}$ is a total ordering, the search terminates with ranking function $\langle y_2 - 2,\ y_1 - 2\rangle$, supported by the invariants $y_1 \geq 1$ and $y_2 \geq 1$, respectively, where $\sigma(1) = 2,\ \sigma(2) = 1$ (*i.e.*, τ_1 maps to the second component, τ_2 to the first).

5 Solving the Numerical Constraints

Consider the specialization $\mathbb{R}_{ij}$, and expand the transition relation $\tau_i(\mathbf{x}\mathbf{x}')$ to the guard $\mathbf{G_i}\mathbf{x} + \mathbf{g_i} \geq \mathbf{0}$ and the update $\mathbf{U_i}\mathbf{x} + \mathbf{V_i}\mathbf{x}' + \mathbf{u_i} \geq \mathbf{0}$, where $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}}$. By Farkas's Lemma, $\mathbb{R}_{i,j}$ expands as follows:

$$\mathbb{R}_{ij}:\ \begin{array}{ll} \mathbf{Ix} & \geq \mathbf{0} \\ \tau_i(\mathbf{xx}') & \geq \mathbf{0} \\ \hline \mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' - \epsilon & \geq \mathbf{0} \end{array} \quad \Rightarrow \quad \begin{array}{r|l} \lambda_I & \mathbf{Ix} \qquad\qquad + \ \mathbf{i} \geq \mathbf{0} \\ \lambda_G & \mathbf{G_i x} \qquad\quad\ + \mathbf{g_i} \geq \mathbf{0} \\ \lambda_U & \mathbf{U_i x} + \ \mathbf{V_i x}' + \mathbf{u_i} \geq \mathbf{0} \\ \hline & \mathbf{c_j}^{\mathrm{T}}\mathbf{x} - \mathbf{c_j}^{\mathrm{T}}\mathbf{x}' - \ \epsilon \geq \mathbf{0} \end{array}$$

The second table corresponds to the constraints

$$\begin{aligned} \lambda_I{}^{\mathrm{T}}\mathbf{I} + \lambda_G{}^{\mathrm{T}}\mathbf{G_i} + \lambda_U{}^{\mathrm{T}}\mathbf{U_i} &= \mathbf{c_j} \\ \lambda_U{}^{\mathrm{T}}\mathbf{V_i} &= -\mathbf{c_j} \\ \lambda_I{}^{\mathrm{T}}\mathbf{i} + \lambda_G{}^{\mathrm{T}}\mathbf{g_i} + \lambda_U{}^{\mathrm{T}}\mathbf{u_i} &\leq -\epsilon \end{aligned} \qquad \begin{aligned} \lambda_I, \lambda_G, \lambda_U &\geq \mathbf{0} \\ \epsilon &> 0 \end{aligned}$$

where $\mathbf{I}$, $\mathbf{i}$, and $\mathbf{c_j}$ are the unknown coefficients. The template invariant coefficients $\mathbf{I}$ and $\mathbf{i}$ appear in nonlinear terms, while the template ranking function coefficients $\mathbf{c_j}$ occur only linearly. This observation holds in general. Thus, the number of template ranking function coefficients has little impact on the complexity of the constraint solving problem. Indeed, if no invariant template is proposed, the constraint solving problem is linear and thus trivial to solve. In this section, we focus on the nonlinear case that arises with a template invariant.

In principle, Tarski's decision procedure for polynomials [21] or CAD [2] can be used to produce solutions of these systems. Sankaranarayanan *et al.*'s work on constraint-based linear invariant generation [3, 16] explores several techniques for solving similar constraint problems that arise in invariant generation; the main practical solution is partly heuristic. We present a principled technique that combines bisection search over the invariant template coefficients with linear constraint solving. It guarantees finding a solution if one exists with integer coefficients of at most some given absolute value.

Our method searches among linear assertions with integer coefficients in a preset range, looking for a linear invariant that makes the constraint system satisfiable. Rather than explicitly enumerating all linear assertions and checking whether they result in a satisfiable constraint system, the method is guided in its enumeration by considering *regions* of coefficients. A *feasibility check* forms a linear overapproximation of the constraint system based on the considered region and checks it for satisfiability. If the overapproximation is found unsatisfiable,

```
bool sat(sys, params) :
  queue := {(1, [−1, 1]^|params|)}
  while |queue| > 0 do
    (depth, r) := choose(queue)
    if feasible(sys, r) then
      if corner(sys, r) then return true
      if depth ≤ D · |params| then
        (r_1, r_2) := bisect(r)
        queue := queue ∪ {(depth + 1, r_1), (depth + 1, r_2)}
  return false
```

Fig. 3. Satisfiability check for numerical constraint systems.

it concludes that no assertion in that region is a supporting invariant, thus excluding the entire region from further search. If the region is found feasible, an assertion in the *corner* of the region is chosen and substituted in the constraint system, thus linearizing the constraint system. If the resulting constraint system is satisfiable, the search terminates, as a solution has been found. Otherwise, the region is bisected, and each subregion is handled recursively until a solution is found or the preset depth is reached. An outline of the algorithm is given in Figure 3.

The feasibility check is computed as follows. Consider a constraint system Φ and a region r consisting of one interval $[\ell_i, u_i]$ for each invariant template coefficient c_i. Each constraint φ in Φ of the form $\cdots \pm c_i\lambda_j + \cdots \geq 0$ is replaced by a constraint φ' in which each c_i is replaced by ℓ_i if $c_i\lambda_j$ occurs negatively and by u_i if $c_i\lambda_j$ occurs positively (recall that $\lambda_j \geq 0$). The resulting linear constraint system Φ' is checked for satisfiability.

Lemma 1. *If Φ is satisfiable for some value of the c_i's in r, then Φ' is satisfiable.*

Consequently, if Φ' is found unsatisfiable, then no solution exists for Φ with coefficients from r; the region r is deemed infeasible.

The linear systems formed during the feasibility check and the check for a solution contain the positive ϵ introduced by the *ranking* specialization. This ϵ may be maximized, resulting in a linear program. The resulting maximum must be positive for the constraint system to be satisfiable. Alternately, ϵ may be set to a constant value, resulting in a linear feasibility check.

The satisfiability check may terminate with a solution, terminate because it hits the maximum depth, or terminate because all regions are found infeasible. If the search terminates only because regions are found infeasible, then no (lexicographic) linear ranking function exists supported by an invariant of the given size. If the maximum depth is ever reached but no solution is found, the result is inconclusive. Assuming a fair *bisect* criterion and that the "lower-left" corner of the region is always chosen when checking a particular solution, we have the following guarantee.

Proposition 2. *sat(sys, params) returns* **true** *iff a supporting invariant exists with integer coefficients in the range $[-2^{D-1}, 2^{D-1})$, for maximum depth $D \geq 1$.*

The forward direction is immediate. For the other direction, first, every linear invariant with rational coefficients may be represented such that all coefficients lie in $[-1, 1]$ and the denominators of coefficients are powers of 2. Second, if *bisect* and *corner* satisfy the stated restrictions, then every combination of such coefficients, with coefficient denominators up to 2^{D-1} and numerators in $[-2^{D-1}, 2^{D-1})$, appears as a corner. Converting these rational coefficients to integers results in integer coefficients in $[-2^{D-1}, 2^{D-1})$. Finally, Lemma 1 ensures that no region containing a solution is pruned.

Invariants from other sources can be used directly by adding them "above the line" of each specialization. For example, polynomial methods exist to generate all invariant linear equalities [7] and many linear inequalities [17]. These invariants strengthen the constraint system without introducing nonlinearities. An invariant template may still be required to enable solving for the harder-to-find supporting invariants. However, one may expect that a smaller template will be sufficient for finding the ranking function, thus reducing the complexity of solving the constraint system.

6 Experimental Results

We implemented our method in O'Caml as a standalone tool. Its input is a set of guarded commands, the initial condition, known invariants, and the size of the supporting invariant template. When successful, it returns a ranking function and its supporting invariants as a witness to termination. By itself, this tool is suitable for analyzing interesting loops like GCD and McCarthy's 91 function. For example, it finds a single ranking function for GCD in about five seconds and a lexicographic ranking function for GCD in about one second. The computation of the lexicographic function is faster because it requires only a one-conjunct template to produce two invariants, while the single ranking function needs two conjuncts. For McCarthy's 91 function, our tool finds the lexicographic ranking function in about a second.

To test our method's general applicability, we implemented a prototype C loop abstracter in CIL [12]. Its input is C source code; its output is a set of abstracted loops, represented as sets of guarded commands. This set of tests measures the performance of our lexicographic ranking function synthesis method without supporting invariants[1]. Thus, all numeric constraint systems are linear.

We implemented three *lexicographic strategies* to guide the synthesis. *One* tries to find a non-lexicographic linear ranking function. *Search* tries to find a non-lexicographic ranking function first, then applies *has_llrf*$(L,\ n)$ if that fails. *Set* tries to find a non-lexicographic ranking function first; if that fails, it tries to find a $|G|$-component lexicographic function *with a fixed initial ordering*. The third strategy was implemented for comparison purposes. One important feature of a scalable analysis is that when it fails on a common class of input, it fails

[1] Our prototype abstracter does not produce initial conditions for the loops, so no assertions can be proved invariant.

				One		*Set*		*Search*			
Name	**LOC**	**#L**	**#A**	**#P**	**Tm**	**#P**	**Tm**	**#P**	**P/A**	**P/L**	**Tm**
`small1`	310	8	6	2	1	2	1	2	33	25	2
`vector`	361	13	13	12	1	12	1	12	92	92	1
`serv`	457	9	6	5	1	5	1	5	83	55	1
`dcg`	1K	55	54	53	2	53	3	53	98	96	2
`bcc`	4K	70	23	18	3	18	4	18	78	25	4
`sarg`	7K	108	25	25	100	25	100	25	100	23	100
`spin`	19K	652	157	119	73	119	73	119	75	18	73
`meschach`	28K	911	805	758	73	763	81	763	94	83	82
`f2c`	30K	436	130	96	77	96	87	96	73	22	87
`ffmpeg/libavformat`	33K	451	268	217	133	217	158	217	80	48	158
`gnuplot`	50K	828	365	297	157	300	170	300	82	36	173
`gaim`	57K	594	71	52	103	52	103	52	73	8	101
`ffmpeg/libavcodec`	75K	2223	1936	1867	263	1868	276	1869	96	84	278

Table 1. Results of tests. Interpret column headings as follows: **Name**: name of program; **LOC**: lines of code *of files successfully parsed and containing loops*, as measured by `wc`; **#L**: number of analyzed loops; **#A**: number of *nontrivial* abstracted loops; **#P**: number of loops proved terminating; **Tm**: time in seconds required to analyze the program; **P/A**: percent of abstracted loops proved terminating; **P/L**: percent of all loops proved terminating. The experimental results are subdivided by the strategies *one*, *set*, and *search*.

quickly. Comparing *Search* with *Set* allows us to assess whether *Search* fails quickly on loops that do not have lexicographic linear ranking functions.

The implementation omits the *disabled* case when generating the constraint systems to avoid introducing disjunctions, which produce multiple numeric constraint systems. In practice, however, the sets of guarded commands extracted by our C loop abstracter contain inconsistent transitions: each guarded command represents a path through the loop, but some of these paths are logically infeasible. Pruning the guarded command set by checking satisfiability of the guards before starting the main analysis leads to significant time and memory savings. This pruning is essentially the *disabled* case without invariants.

We applied our C loop abstracter and termination tool to a set of thirteen C projects downloaded from NETLIB [13] and SOURCEFORGE [20]. Table 1 displays the results. The timing data are obtained from running the tests on a 3GHz dual-Pentium processor with 2GB of memory. Loops are counted as successfully abstracted if they do not have any trivial guards or updates. The data indicate that our method provides good coverage of successfully abstracted loops (**P/A**) at low cost in time (**Tm**). Moreover, the timing data indicate that the extra power of lexicographic function synthesis comes at almost no cost. Although *Search* invokes the search method on every loop for which *One* fails, its speed is competitive with *One* while proving 10 more loops terminating. Moreover, comparing *Search* with *Set* indicates that the search method fails quickly when it must fail: the feasibility check controls an otherwise expensive search.

7 Conclusion

Our method of ranking function synthesis demonstrates two new ideas. First, the constraint-based analysis style of separating the encoding of requirements from the synthesis itself [4] is extended by incorporating structural and numeric constraints. The constraint solving process involves an interplay between satisfying structural requirements and the induced numeric constraints. Second, supporting invariants are found implicitly. This choice avoids the full cost of invariant generation. Our numeric constraint solver exploits the implicit nature of the supporting invariants, using a feasibility check to exclude sets of irrelevant assertions and invariants. In a real sense, the ranking conditions guide the search for supporting invariants.

Acknowledgments We thank the reviewers for their insightful comments.

References

1. Codish, M., Genaim, S., Bruynooghe, M., Gallagher, J., and Vanhoof, W. One lop at a time. In *WST* (2003).
2. 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.
3. Colón, M. A., Sankaranarayanan, S., and Sipma, H. B. Linear invariant generation using non-linear constraint solving. In *CAV* (2003), pp. 420–433.
4. Colón, M. A., 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. 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.
7. Karr, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976).
8. Katz, S. M., and Manna, Z. A closer look at termination. *Acta Informatica 5*, 4 (1975), 333–352.
9. Lee, C. S., Jones, N. D., and Ben-Amram, A. M. The size-change principle for program termination. In *POPL* (2001), pp. 81–92.
10. Manna, Z. *Mathematical Theory of Computation.* McGraw-Hill, 1974.
11. 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.
12. 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).
13. *Netlib Repository*, 2004. (`http://www.netlib.org`).
14. Podelski, A., and Rybalchenko, A. A complete method for the synthesis of linear ranking functions. In *VMCAI* (2004), pp. 239–251.
15. Podelski, A., and Rybalchenko, A. Transition invariants. In *LICS* (2004), pp. 32–41.

16. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Constraint-based linear-relations analysis. In *SAS* (2004), pp. 53–68.
17. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Scalable analysis of linear systems using mathematical programming. In *VMCAI* (2005), pp. 25–41.
18. Schrijver, A. *Theory of Linear and Integer Programming*. Wiley, 1986.
19. Sipma, H. B., Uribe, T. E., and Manna, Z. Deductive model checking. In *CAV* (1996), pp. 209–219.
20. *SourceForge*, 2004. (`http://sourceforge.net`).
21. Tarski, A. *A Decision Method for Elementary Algebra and Geometry*, 2 ed. University of California Press, Berkeley, CA, 1951.