Policy Iteration-Based Conditional Termination and Ranking Functions

Damien Massé

Univ. de Brest, UMR 6285, Lab-STICC, F-29200 Brest, France
damien.masse@univ-brest.fr

Abstract. Termination analyzers generally synthesize ranking functions or relations, which represent checkable proofs of their results. In [23], we proposed an approach for conditional termination analysis based on abstract fixpoint computation by policy iteration. This method is not based on ranking functions and does not directly provide a ranking relation, which makes the comparison with existing approaches difficult. In this paper we study the relationships between our approach and ranking functions and relations, focusing on extensions of linear ranking functions. We show that it can work on programs admitting a specific kind of segmented ranking functions, and that the results can be checked by the construction of a disjunctive ranking relation. Experimental results show the interest of this approach.

1 Introduction

Many approaches have been proposed to prove that a program terminates. Most techniques rely on the construction of ranking functions [13,24] or ranking relations as transition invariants [25]. Ranking functions and relations, like invariants in safety analysis, offer a checkable result, not directly related to the technique used to construct them. Similarly, a conditional termination analysis [7], i.e. an analysis which determines the set of terminating states, is expected to return not only the a set of terminating states, but an associated ranking function or relation.

In [23], we proposed to use policy iteration techniques in order to analyze conditional termination. Policy iteration (or strategy iteration) [10,19] has been developed as an alternative to the classical widening/narrowing techniques used for safety analysis by abstract interpretation on numerical programs. Applied on conditional termination, it can be used to compute, using linear optimization algorithms, an overapproximation of the (potentially) non-terminating states, hence proving the termination of the other states. However, it does not directly provide any ranking function or relation.

In this paper, we examine the relationships between our approach and ranking functions synthesis. We focus especially on disjunctive linear ranking relations and segmented linear ranking functions. We show the construction of a disjunctive ranking relation from the analysis, as well as the existence of a segmented

K.L. McMillan and X. Rival (Eds.): VMCAI 2014, LNCS 8318, pp. 453–471, 2014.

linear ranking function. Reciprocally, we show that programs admitting some restricted form of segmented ranking function can be analyzed by policy iteration. We complete these results by a practical experiments on a prototype analyzer.

2 Notations

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a tuple of n variables. We denote any element of $\mathbb{R}^{\boldsymbol{x}}$ either by $(x_1 = v_1; \ldots; x_n = v_n)$ or as a column vector $\boldsymbol{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$ of $\mathbb{R}^n$. When used as a matrix block, $\boldsymbol{x}$ represents the column vector $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$. Linear forms on $\mathbb{R}^{\boldsymbol{x}}$ are denoted either as a expression $(c_1x_1 + \ldots + c_nx_n)$ or as a row vector $(c_1 \ \ldots \ c_n)$. Similarly, $m \times n$-matrices $M \in \mathbb{R}^{m \times n}$ may be denoted as a m-tuple of linear forms on $\mathbb{R}^{\boldsymbol{x}}$. Given a matrix M, we denote by M^T its transpose.

3 Affine Programs and Semantics

3.1 Programs

For the sake of simplicity, we consider in this paper programs with only one program point. Hence, $\boldsymbol{x}$ being the set of (real) variables, a program is a pair $(I, \mathbf{T})$ where $I \subseteq \mathbb{R}^{\boldsymbol{x}}$ is the set of initial states, and $\mathbf{T} \subseteq \wp(\mathbb{R}^{\boldsymbol{x}} \times \mathbb{R}^{\boldsymbol{x}})$ describes the transitions, each transition defining a relation between the values of the variables before and after the transition.

Since we are interested in affine programs, I and $\mathbf{T}$ will be defined by linear constraints, that is:

1. I is a set of linear constraints on $\boldsymbol{x}$
2. Each transition of $\mathbf{T}$ is described as a set of linear constraints on $\boldsymbol{x}, \boldsymbol{x'}$ where $\boldsymbol{x'}$ represents the variables after the transition.

In our framework, I can be used to compute an overapproximation of the reachable states. However, for simplicity we will assume that all states are reachable, and I will not be used.

Example 1. We consider the program A of Fig 1. $\mathbf{T} = \{t_1, t_2\}$ is represented with:

$$t_1 : \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ a' \\ b' \end{pmatrix} \begin{matrix} \leq \\ \leq \\ = \\ = \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

Program A:

```
1: while a ≥ 0 do
2:     a ← a + b
3:     if b ≥ 0 then
4:         b ← −b − 1
5:     else
6:         b ← −b
7:     end if
8: end while
```

Program B:

```
1: while x ≤ 100 do
2:     if (*) then
3:         x ← −2x + 2
4:     else
5:         x ← −3x − 2
6:     end if
7: end while
```

Fig. 1. Two affine programs. a and b are integer variables, x is a real variable, and (*) is a non-deterministic choice.

$$t_2 : \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ a' \\ b' \end{pmatrix} \begin{matrix} \leq \\ \leq \\ = \\ = \end{matrix} \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

Note that we replace $b < 0$ by $b \leq -1$ in the representation, since b is an integer variable. Our approach deals mainly with real variables, and using strict constraints (or floating-point computations) requires technical considerations which are outside the scope of this paper.

In the following, we will represent a transition as a pair $(Q, \boldsymbol{q})$ where Q is a matrix and $\boldsymbol{q}$ is a vector such that the associated constraints are:

$$Q \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{pmatrix} \leq \boldsymbol{q}$$

When the program has only one transition, the program is called a *Linear Simple Loop* (LSL). Termination of Linear Simple Loop is well known to be a decidable problem (at least for linear assignments) [29]. However, deciding conditional termination of LSL is more complex [4], and not always possible [16].

3.2 Concrete Semantics

To each program P is associated a transition relation $\tau \subseteq \mathbb{R}^{\boldsymbol{x}} \times \mathbb{R}^{\boldsymbol{x}}$. This relation can be used to construct a trace-based semantics to prove termination [12]. However, we propose to use here a state based backward semantics:

Proposition 1. *The set $\mathcal{S}$ of states starting an infinite execution trace is equal to:*

$$\mathcal{S} = \mathrm{gfp}\, \lambda Y.\mathrm{pre}(Y)$$

where $\mathrm{pre} \in \wp(\mathbb{R}^{\boldsymbol{x}}) \to \wp(\mathbb{R}^{\boldsymbol{x}})$ *is the predecessor predicate transformer:*

$$\begin{aligned} \mathrm{pre}(Y) &= \{\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{x}} \mid \exists \boldsymbol{v'} \in Y, (\boldsymbol{v}, \boldsymbol{v'}) \in \tau\} \\ &= \bigcup_{(Q,\boldsymbol{q}) \in \mathbf{T}} \{\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{x}} \mid \exists \boldsymbol{v'} \in Y, Q \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{v'} \end{pmatrix} \leq \boldsymbol{q}\} \end{aligned}$$

If Y is a polyhedron characterized with a set of linear constraints $A\boldsymbol{x} \leq \boldsymbol{c}$, $\mathrm{pre}(Y)$ is a union of polyhedra, which can be computed using libraries like Apron [21]. However, even if $\mathrm{pre}(Y)$ is computable, $\mathcal{S}$ is not computable, since the fixpoint computation may not terminate.

As seen in [12], it is well-known that a ranking function can be defined from the iterates of the pre operator:

Proposition 2. *Let $(S_i)_{i\in\mathbb{O}}$ be the iterates of $\mathrm{gfp}\,\lambda Y.\mathrm{pre}(Y)$ (i.e. $S_0 = \mathbb{R}^{\boldsymbol{x}}$, $S_{i+1} = \mathrm{pre}(S_i)$ and, for all limit ordinal l, $S_l = \cap_{i<l} S_i$). Then the function r defined on $\Sigma \setminus S$ by:*

$$\forall x \in \Sigma \setminus S, r(x) = \min\{i \in \mathbb{O} | r \notin S_i\}$$

is a ranking function over $\Sigma \setminus S$:

$$\forall x \in \Sigma \setminus S, \forall y \in \Sigma, x \to y \Longrightarrow (y \notin S \wedge r(y) < r(x))$$

However, computing (or approximating) $\mathcal{S}$ may not give the successive iterates.

3.3 Abstraction and Abstract Semantics

The *template polyhedral abstraction*[27] is a parametric sub-abstraction of the classical polyhedral abstraction, where the linear constraints used must belong to a template. In practice, the template is a matrix $T \in \mathbb{R}^{m\times n}$. Each row of T represents a linear form of program variables. The matrix T defines an abstraction of $\wp(\mathbb{R}^n)$, where an abstract element ρ is a element of $\mathcal{T}_T = \overline{\mathbb{R}}^m$ where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.. The concretization function is defined as:

$$\gamma_T(\rho) = \{\boldsymbol{v} \in \mathbb{R}^n \mid T\boldsymbol{v} \leq \rho\}$$

Note 1. In the following, we may consider T as a matrix or as a set of m linear forms. For any $\rho \in \mathcal{T}_T$, we will denote by $[\rho]_f$ the component of ρ associated to $f \in T$.

In the context of this abstraction, the best abstract transformer $\mathrm{pre}^\sharp = \alpha_T \circ \mathrm{pre} \circ \gamma_T$ is computable:

Lemma 1. *With $\rho \in \mathcal{T}_T$, the component of $\mathrm{pre}^\sharp(\rho)$ associated to f satisfies $[\mathrm{pre}^\sharp(\rho)]_f = \max_{(Q,\boldsymbol{q})\in\mathbf{T}}[[\![(Q,\boldsymbol{q})]\!]^\sharp]_f(\rho)$ with:*

- *If*
$$\{x | \exists x', \begin{pmatrix} Q & \\ 0 & T \end{pmatrix} \begin{pmatrix} x \\ x' \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{q} \\ \rho \end{pmatrix}\} = \emptyset,$$
then $[[\![(Q,\boldsymbol{q})]\!]^\sharp]_f = -\infty$.
- *otherwise,*
$$[[\![(Q,\boldsymbol{q})]\!]^\sharp]_f = \min\{\boldsymbol{\lambda}(\boldsymbol{q}^T\rho^T) | \boldsymbol{\lambda} \geq 0 \wedge \begin{pmatrix} Q^T & 0 \\ & T \end{pmatrix} \boldsymbol{\lambda} = \begin{pmatrix} f^T \\ 0 \end{pmatrix}\}$$

Hence $\mathrm{pre}^\sharp(\rho)$ can be computed by solving $m.|\mathbf{T}|$ linear programs. However, the classical Kleene iterations (with widening and narrowing operators) can only be used to approximate the abstract semantics $\mathcal{S}^\sharp = \mathrm{gfp}\,\mathrm{pre}^\sharp$ with the following restrictions:

1. Using a *dual* widening operator can only be used to find an *underapproximation* of the abstract semantics [11], which is not sound since our abstract semantics is an *overapproximation* of the concrete semantics[1].
2. Using n steps of a narrowing operator would only find states terminating after at most n iterations.

Thus we need to use *policy iteration* to compute $\mathcal{S}^\sharp$.

4 Policy Iteration

Policy or strategy iteration [10,18,19] is a method to compute the least (or greatest) fixpoint of specific classes of monotonic operators. It can be seen as an adaptation of the classical Newton's method in the sense that the operator σ is approximated (w.r.t. a *policy selection*) by a policy σ_0 where the fixpoint of σ_0 is computable. A new policy is selected to approximate σ around the fixpoint of σ_0, and the process iterates until a global fixpoint is reached.

The application of policy iteration to the computation of greatest fixpoints has been presented in [23]. Although this section extends slightly the framework by not restricting transitions to assignments, the results are similar.

4.1 Policy Selection

Theorem 1. *Given a transition $t = (Q, \boldsymbol{q})$, $[[\![t]\!]^\sharp]_f$ is the minimum of two function ψ^t and $[\phi^t]_f$ where:*

- *ψ^t is monotonic and its image is in $\{-\infty, +\infty\}$;*
- *$[\phi^t]_f$ is the minimum of a finite number of affine expressions (with positive coefficients) on ρ.*

Example 2. For t_1 in program A, and $T = (-a, -a - b)$, we have:

$$\begin{aligned}\psi^{t_1}(\rho) &= +\infty\\ [\phi^{t_1}]_{-a}(\rho) &= -1 + \rho_{-a-b}\\ [\phi^{t_1}]_{-a-b}(\rho) &= \min(-1 + \rho_{-a-b}, \rho_{-a})\end{aligned}$$

Theorem 2. *Let t be a transition, and $\rho \in \overline{\mathbb{R}}^m$. The value of $\psi^t(\rho)$ and, if $\psi^t(\rho) = +\infty$, the affine expression for which $\phi^t(\rho)$ is minimal can be computed by solving the following linear program:*

$$\max\{f.\boldsymbol{x} | \exists \boldsymbol{x}, \boldsymbol{x'}, \begin{pmatrix} Q \\ 0\ T \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{q} \\ \rho \end{pmatrix}\}$$

[1] The overapproximation being necessary to prove termination.

$\sigma \leftarrow (+\infty)$ ▷ Initial policy
$\rho \leftarrow (+\infty)$ ▷ Initial fixpoint: $\top$
while $\Phi(\rho) \neq \rho$ **do** ▷ Stop if fixpoint is reached
  for all $f \in T$ **do**
    if $[\Phi(\rho)]_f < [\rho]_f$ **then** ▷ Change $[\sigma]_f$ only if it is not optimal for ρ
      $[\sigma]_f \leftarrow$ min-policy of $[\Phi]_f$ such that $[\sigma]_f(\rho) = [\Phi]_f(\rho)$
    end if
  end for
  $\rho \leftarrow \operatorname{gfp}_{\leq\rho}\sigma$ ▷ Compute the next fixpoint for the policy
end while

Fig. 2. Policy iteration algorithm for greatest fixpoint computation. This algorithm computes $\operatorname{gfp}\Phi$ using min-policies: each policy component $[\sigma]_f$ is given by a choice between the min-terms of $[\Phi]_f$.

If this linear program is infeasible, then $\psi^t = -\infty$, otherwise, $\psi^t = +\infty$ and ϕ^t_f can be directly constructed from the optimal dual solution.

Given a current policy σ and its fixpoint ρ, the new policy σ' is constructed as follows : $[\sigma']_f = \max_{t\in\mathbf{T}}[\sigma^t]_f$ where $[\sigma^t]_f = -\infty$ if $\psi^t(\rho_0) = -\infty$, and $[\sigma^t]_f$ is an affine expression of $[\phi^t]_f(\rho_0)$ which is minimal for ρ_0 otherwise.

Proposition 3. *The policy selection process constructs a new policy for which each component is the maximum of affine expressions.*

Example 3. For program A, if the current post-fixpoint is $(\rho_{-a} = 0, \rho_{-a-b} = 0)$, the policy selection process gives:

$$[\sigma_1]_{-a}(\rho) = \max(\rho_{-a-b} - 1, \rho_{-a} - 1)$$
$$[\sigma_1]_{-a-b}(\rho) = \max(\rho_{-a}, \rho_{-a-b} - 1)$$

In the program, this policy expresses the fact that, if after an iteration the constraints $-a \leq u \leq 0$ and $-a - b \leq v \leq 0$ are satisfied, then before the iteration, the constraints $-a \leq \max(u - 1, v - 1)$ and $-a - b \leq \max(u, v - 1)$ are satisfied.

In general, we will write the policy as a system of equations over the components of ρ, e.g.:

$$\rho_{-a} = \max(\rho_{-a-b} - 1, \rho_{-a} - 1)$$
$$\rho_{-a-b} = \max(\rho_{-a}, \rho_{-a-b} - 1)$$

4.2 Policy Iteration Result

Theorem 3 ([23]). *Following the algorithm of policy iteration presented Fig. 2, the new fixpoint ρ_l of each policy σ_l is computable by solving two linear programs. Furthermore, the algorithm terminates and gives the abstract fixpoint $\mathcal{S}^\sharp$.*

Note 2. Although the algorithm of Fig. 2 starts with $\sigma_0 = (+\infty)$, we can adapt it to start from any post-fixpoint.

Example 4. Figure 3 shows the results of the analyzes of programs A and B. The initial policy $((+\infty))$ is omitted. For both programs, Step 1 gives just the translation of the termination condition of the loop. Step 2 gives the equivalent of one more iteration. However, in Step 3, the relations between variables in the equation system enables to jump directly to the fixpoint. This jump is equivalent to ω iterations in the greatest fixpoint computation. Note that the analysis proves the termination of program A from every initial state. For program B, it proves that the programs terminates from $x > 1.6 \vee x < -1.2$, which is the best result we can get with a polyhedral abstraction ($x = 1.6$ and $x = -1.2$ being both non-terminating states). However, it does not give the exact set of terminating states (e.g. one can check that this program terminates from any integer).

Step	Policy	Fixpoint
1	$\rho_{-a} = 0$ $\rho_{-a-b} = +\infty$	$-a \leq 0$
2	$\rho_{-a} = \max(0, \rho_{-a} - 1)$ $\rho_{-a-b} = \rho_{-a}$	$-a \leq 0$ $-a - b \leq 0$
3	$\rho_{-a} = \max(\rho_{-a-b} - 1, \rho_{-a} - 1)$ $\rho_{-a-b} = \max(\rho_{-a}, \rho_{-a-b} - 1)$	$\emptyset$

(Prog. A)

Step	Policy	Fixpoint
1	$\rho_x = 100$ $\rho_{-r} = +\infty$	$x \leq 100$
2	$\rho_x = 100$ $\rho_{-x} = \max((\rho_x - 2)/2, (\rho_x + 2)/3))$	$x \leq 100$ $-x \leq 49$
3	$\rho_x = \max((\rho_{-x} + 2)/2, (\rho_{-x} - 2)/3)$ $\rho_{-x} = \max((\rho_x - 2)/2, (\rho_x + 2)/3))$	$x \leq 1.6$ $-x \leq 1.2$

(Prog. B)

Fig. 3. Results of the policy iteration process on program A with the template $T = (-a, -a - b)$ and program B with the template $T = (-x, x)$

5 Relationships with Ranking Functions

The policy iteration process computes the exact abstract semantics of the program. Proposition 2 states that this fixpoint entails the existence of a ranking function based on the iterates of the fixpoint, but does not give the form of the ranking function (or relation). In order to compare this approach with other existing methods, we need to make this ranking function explicit, or at least precise the kind of ranking relations a program must satisfy to be successfully analyzable by policy iteration. Since we use linear templates, linear ranking functions and their derivatives (piecewise linear ranking functions and disjunctive linear ranking relations) are the most interesting candidates to compare with our approach.

5.1 Ranking Functions and Relations

Linear Ranking Function. Linear ranking functions are commonly used to prove termination of simple programs.

Definition 1 (Ranking function). *If Σ is a set of states, S a subset of Σ, and $\tau \subseteq \Sigma \times \Sigma$ a transition relation, a* ranking function *over S is defined by an ordered set $(O, \prec)$ and a function $r : S \to O$ such that $\prec$ is a well-founded order and:*

$$\forall \sigma \in S, \sigma \xrightarrow{\tau} \sigma' \;\Rightarrow\; \sigma' \in S \text{ and } r(\sigma') \prec r(\sigma)$$

If τ represents the transition relation induced by a program P, the existence of a ranking function over S shows that the program terminates from any state in S.

The *ranking relation* $T(r) \subseteq S \times S$ generated by a ranking function r is the well-founded relation defined as $T(r) = \{(\sigma_1, \sigma_2) \mid r(\sigma_2) \prec r(\sigma_1)\}$. If r is a ranking function for τ over S, then $T(r)$ satisfies:

$$\tau \;\subseteq\; T(r) \cup (\Sigma \setminus S) \times \Sigma$$

We may use ordinals $(\mathbb{O}, <)$ as well-founded sets. However, since our approach deals with real values, we will denote by $\prec$ on a subset of $\mathbb{R}$, any well-founded suborder of $<$ on this subset.

Definition 2 (Linear ranking function). *A ranking function r on $(O, \prec)$ is* linear *if O is a subset of $\mathbb{R}$, $\prec$ is a sub-order of $<$ on O, and r is linear.*

Segmented Linear Ranking Functions. The domain of segmented ranking functions is presented in [30] to infer termination properties on programs. Its analysis produces piecewise-segmented ranking functions to infer sufficient conditions on programs. The domain is parametrized by two numerical abstract domains for the partitioning of the environment and for the values of the function. The prototypes used intervals for the partitioning and affine forms for the functions, but it should be possible to use other linear constraints (maybe templates) for the partitioning (which should be costly). We propose to call this instantiation of the domain *segmented linear ranking functions.*

Definition 3 (Segmented linear ranking function). *Let $S \subseteq \mathbb{R}^m$ and τ a transition relation on $\mathbb{R}^m$, a ranking function $r : S \to O$ on S is segmented linear if it can be defined by a n-uplet $(S_1, r_1), \ldots, (S_n, r_n)$ where:*

- *$\{S_i\}_{1 \le i \le n}$ is a partition of S, and each S_i is a polyhedron;*
- *for all i, r_i is defined on S_i and $r = r_i$ on S_i.*
- *all the r_i are linear.*

Disjunctive Linear Ranking Relations. An alternative to ranking functions are disjunctive ranking relations, defined as a finite union of ranking relations $T = T_1 \cup \ldots \cup T_n$. Although T may not be itself a ranking relation, a transition relation τ is well-founded if and only if its non-reflective transitive closure τ^+ is included in a disjunctive ranking relation (which is then called a *transition invariant*[25]). This approach is widely used to prove termination, using model-checking procedures to check the inclusion in transition invariants [22,8]. Of course, disjunctive ranking functions can be used to prove conditional termination:

Lemma 2. *Let P a program with a transition relation $\tau \subseteq \Sigma \times \Sigma$. Then P terminates from all states in S if and only if there exists a disjunctive ranking relation $T = T_1 \cup \ldots \cup T_n$ such that:*

$$\tau^+ \subseteq T \cup (\Sigma \setminus S) \times \Sigma$$

In [5], Chen *et al.* proposed to infer disjunctive ranking relations for linear simple loops (LSLs), where each ranking relation T_i is (by construction) based on a linear ranking function. More precisely, $T_i = T(r_i)$ such that there exists a polyhedral partition $(P_1, \ldots, P_k)$ of $\mathbb{R}^n$ where:

- r_i is linear over P_1;
- r_i is constant over P_2, ..., P_k and its values are always strictly lower than the elements of $r_i(P_1)$.

We will call these relations *disjunctive linear ranking relations.*

Before considering the ranking functions induced by the policy iteration algorithm, we examine the relationships between disjunctive ranking relations and segmented ranking functions. Disjunctive linear ranking relations are strictly more powerful than segmented linear ranking functions: any segmented linear ranking function induces a disjunctive linear ranking relation, but the converse is not true.

Theorem 4. *Let $\tau \subseteq \Sigma \times \Sigma$ and r a segmented linear ranking function for τ over S, defined by the n-uplet (S_1, r_1), ..., (S_n, r_n). For all $1 \leq i \leq n$, we define $T_i : S \times S$ as:*

$$T_i = T(r_i) \cup (S_i \times (S \setminus S_i))$$

Then $\tau^+ \subseteq T_1 \cup \ldots \cup T_n \cup (\Sigma \setminus S) \times \Sigma$.

Example 5. Program A admits a segmented linear ranking function r defined as:

$$r(a,b) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{if } a \geq 0 \text{ and } a + b < 0 \\ 2a + 2 & \text{if } a \geq 0 \text{ and } b \geq 0 \\ 2(a+b) + 3 & \text{if } a + b \geq 0 \text{ and } b < 0 \end{cases}$$

The partition contains four sets. The disjunctive ranking relation allows any transition between two different sets, but only decreasing transitions (w.r.t. the local ranking function) inside one set. While not well-founded, it is the union of well-founded relations and includes the transition closure of τ.

The disjunctive ranking relation does not give any information about the relations between the elements of the partition, therefore it can prove the termination of programs which do not admit a segmented linear ranking function.

Example 6. The program

```
1: while x ≥ 0 do
2:     x ← x + y
3:     if y ≥ 0 then
4:         y ← y − 1
5:     end if
6: end while
```

terminates and admits a disjunctive ranking relation defined by the ranking functions:

$$\rho_0 = \begin{cases} x & \text{if } y \leq -1 \text{ and } x \geq 0 \\ -1 & \text{if } x < 0, \text{ or } y > 0 \end{cases} \qquad \rho_1 = \begin{cases} y & \text{if } y > 0 \\ 0 & \text{if } y \leq 0 \end{cases}$$

However, a segmented ranking function would need to be quadratic when $y > 0$.

5.2 Policy Iteration and Ranking Relations

Our goal is to link the results of the policy iteration analysis with the existence of disjunctive ranking relations or segmented ranking functions. Since the policy iteration analysis gives an overapproximation of the non-terminating states, there exists a ranking function or relation on the complement, which should be related to the template used.

Example 7. For program B, the policy iteration analysis (with $T = (x, -x)$) proves that the program terminates from $x > 1.6$ or $x < -1.2$. A ranking function r should be defined on $]-\infty, -1.2[\cup]1.6, +\infty[$. Since $|x|$ increases at each iteration, r should be increasing on $]-\infty, -1.2[$ (with $\lim_{x \to -1.2} r(x) = \omega$) and decreasing on $]1.6, +\infty[$ (with $\lim_{x \to 1.6} r(x) = \omega$). Hence, on $]-\infty, -1.2[$, x is a ranking function, whereas $-x$ is a ranking function on $]1.6, +\infty[$. Note that the value -1.2 is given by ρ_{-x} and 1.6 by ρ_x. Therefore, it appears that the partial ranking functions are related to the negation of the template elements.

In general, we shall prove that if the policy iteration analysis shows that A is a set of terminating states:

1. A disjunctive linear ranking relation can be defined on A, of which the ranking relations are directly related to the template (Theorem 5).
2. We can also find on A a segmented linear ranking function r, with template-related restrictions of the domains of the subfunctions. To represent these restrictions, we shall describe r as a min-defined segmented ranking function, i.e. as a minimum of functions with overlapping domains (Definition 4). Furthermore, we prove the converse of this result, i.e. the existence of a

segmented linear ranking function satisfying these restrictions on A implies that the policy iteration analysis can prove conditional termination on A (Theorem 6).

By Theorem 3, we know that the policy iteration process returns the exact abstract semantics of the program, defined as $\mathcal{S}^\sharp = \text{gfp}\ \alpha_T \circ \text{pre} \circ \gamma_T$. The iterates of this fixpoint are elements of the template abstract domain. Hence we can expect a potential ranking relation to be closely related to the template linear forms. Theorem 5 formalizes this idea and shows that the programs directly admits a disjunctive linear ranking relation on the terminating part:

Theorem 5. *Let T be a template with m linear forms $f_1, \ldots, f_m$ over $\mathbb{R}^{\boldsymbol{x}}$, and $A^\sharp$ an abstract element of $\mathcal{T}_T$. If $\text{gfp}\,\lambda X.(\text{pre}_T^\sharp(X)) = A^\sharp$, then there exists m ranking relations R_1, R_2, ..., R_m on $\mathbb{R}^{\boldsymbol{x}} \setminus \gamma_T(A^\sharp)$ satisfying the conditions (C1) and (C2) defined as follows:*

(C1) *For all i, there exists a well-founded suborder $\prec^i$ of $<$ on $\mathbb{R}$ such that:*

$$\forall(\boldsymbol{v}_1, \boldsymbol{v}_2) \in \mathbb{R}^n \setminus \gamma_T(A^\sharp), (\boldsymbol{v}_1, \boldsymbol{v}_2) \in R_i \iff -f_i\boldsymbol{v}_2 \prec^i -f_i\boldsymbol{v}_1$$

(C2) *$R_1 \cup \ldots \cup R_m$ is a disjunctive ranking relation for τ over $\mathbb{R}^n \setminus \gamma_T(A^\sharp)$:*

$$\tau^+ \subseteq (R_1 \cup \ldots \cup R_m) \cup \gamma_T(A^\sharp) \times \mathbb{R}^n$$

Furthermore, with $(\text{pre}_T^\sharp)^k$ denoting the k-th iterate of the operator $\text{pre}_T^\sharp$ starting from $\mathbb{R}^n$, we can construct $\prec^i$ as:

$$u \prec^i v \iff \exists k \in \mathbb{O}, u < -[(\text{pre}_T^\sharp)^k]_{f_i} \leq v$$

This theorem proposes well-founded orders $\prec^i$ based on sets of iterates, which are not easy to use. We shall examine the problem of finding simpler orders in Sect. 5.3.

Example 8. We can prove the termination of program A with the template $T = (-a, -a-b)$. The associated relations R_1 and R_2 can be defined on $\mathbb{R}^2$ as follows:

$$((a,b),(a',b')) \in R_1 \text{ iff } a' \prec a$$
$$((a,b),(a',b')) \in R_2 \text{ iff } a'+b' \prec a+b$$

where

$$u \prec u' \iff a < 0 \leq b \vee 0 \leq a+1 \leq b$$

We can see that R_1 and R_2 are generated by the functions a and $a+b$.

Example 9 shows that the converse of Theorem 5 does not hold:

Example 9. The program seen in Example 6 admits a disjunctive ranking relation

$$T(\rho_0) \cup T(\rho_1) \text{ with } \rho_0 = x \text{ and } \rho_1 = y$$

with the well-founded order $\prec$ defined as $a \prec b \Leftrightarrow a < 0 \leq b \vee a + 1 \leq b$. However, analyzing the program with the template $T = (-x, -y)$ gives just the condition $x \geq 0$ for potential non-termination, because the next iterate in the concrete domain gives the constraint $x + y \geq 0$ which is not translated in the abstract domain.

Theorem 5 presents a disjunctive linear ranking relation of which the components are based on the policy iteration analysis. As we saw on Example 6, this does not prove the existence of a segmented linear ranking function. Two difficulties arise when we try to construct a segmented ranking function from the policy iteration analysis.

First, the domains of the subfunctions must partition the terminating states, whereas each well-founded order of Theorem 5 is defined on the whole domain. We circumvent the problem by including the possibility of overlapping domains. In this case, the value of the r is defined as the minimum of the values of the underlying functions.

Definition 4 (Min-defined segmented ranking functions). *Let Σ be a set of states, S a subset of Σ and τ a transition relation on Σ, a ranking function $r : S \to O$ on S (where $O \subseteq \mathbb{R}$) is min-defined segmented if it can be defined by a n-uplet $(S_1, r_1), \ldots, (S_n, r_n)$ where:*

1. $\bigcup_{1 \leq i \leq n} S_i = S$,
2. *and* $\forall \sigma \in S, r(\sigma) = \min_{\sigma \in S_i} r_i(\sigma)$ *(where* min *is the minimum with respect to the total order $<$).*

Furthermore, r is min-defined segmented linear if all r_i are linear.

Of course, any min-defined segmented ranking function can be transformed to a segmented ranking function by restricting the domain of the subfunctions. However, using them makes the following theorem much easier to present, as the domain of each subfunction becomes independent of the others subfunctions.

The second difficulty is the relationships between the values of different subfunctions. The easiest approach is to consider intermediate functions φ:

Theorem 6. *Let $A^\sharp \in \mathcal{T}_T$. Then* $\operatorname{gfp} \lambda X.(\operatorname{pre}^\sharp_T(X)) \sqsubseteq^\sharp A^\sharp$ if and only if *there exists a min-defined segmented ranking function $r = \min_{i \in \{1,\ldots,m\}} r_i : \mathbb{R}^n \setminus \gamma_T(A^\sharp) \to O$ such that for all i:*

(C3) $\operatorname{Dom}(r_i) = \{\boldsymbol{v} \in \mathbb{R}^n \mid [A^\sharp]_{f_i} < f_i \boldsymbol{v}\}$
(C4) $\forall i, r_i(\boldsymbol{v}) = \varphi_i(-f_i \boldsymbol{v})$ *where φ_i is a monotonic function from $]-\infty, -[A^\sharp]_{f_i}[$ to O.*

Proof (sketch). Let's consider the iterates $(A^\sharp_k)$ of $pre^\sharp_T$ starting from $(+\infty)$. We define $r_i(\boldsymbol{v})$ as the maximal ordinal k such that $f_i \boldsymbol{v} \leq [A^\sharp_k]_{f_i}$. One can easily check that this ordinal exists (if $\boldsymbol{v} \in \operatorname{Dom}(r_i)$), that r_i is monotonic w.r.t. $-f_i \boldsymbol{v}$, and that r is a ranking function. Reciprocally, if a ranking function r satisfy (C3) and (C4), then we prove by transfinite induction that for all $\boldsymbol{v} \in \gamma_T(A^\sharp_k)$, $r(\boldsymbol{v}) \geq k$. The limit case is a consequence of the co-continuity of γ_T. For the successor case,

let us suppose that there exists $\boldsymbol{v} \in \gamma_T(A^\sharp_{k+1}) = \gamma_T \circ \alpha_T(\mathrm{pre}(\gamma_T(A^\sharp_k)))$ such that $r(\boldsymbol{v}) < k+1$ (for example, $r_i(\boldsymbol{v}) \leq k$). Then there exists $\boldsymbol{v'} \in \mathrm{pre}(\gamma_T(A^\sharp_k))$ such that $f_i\boldsymbol{v'} \geq f_i\boldsymbol{v}$, hence $r_i(\boldsymbol{v'})$ is defined by (C3) and $r(\boldsymbol{v'}) \leq r_i(\boldsymbol{v'}) \leq r_i(\boldsymbol{v}) \leq k$ by (C4). Since $\boldsymbol{v'} \in \mathrm{pre}(\gamma_T(A^\sharp_k))$, there must be a successor $\boldsymbol{v''}$ of $\boldsymbol{v'}$ in $\gamma_T(A^\sharp_k))$, which by induction hypothesis must satisfy $r(\boldsymbol{v''}) \geq k$, which contradicts the fact that r is a ranking function.

Example 10. Example 5 gives a segmented ranking r function for program A, which can also be min-defined:

$$r(a,b) = \min(r_{-a}(a,b), r_{-a-b}(a,b))$$

with r_{-a} (resp. r_{-a-b}) depending only on a (resp. $a+b$):

$$r_{-a}(a,b) = \begin{cases} 0 & \text{if } a < 0 \\ 2a+2 & \text{if } a \geq 0 \end{cases}$$

$$r_{-a-b}(a,b) = \begin{cases} 1 & \text{if } a+b < 0 \\ 2(a+b)+3 & \text{if } a+b \geq 0 \end{cases}$$

Theorem 6 shows the form of a potential ranking function for the set of terminating states found by the policy iteration algorithm. Also, it gives a completeness result by describing the programs analyzable with a specific template as any program admitting a min-defined segmented ranking function satisfying these conditions. Since linear ranking functions satisfy them, we can deduce that programs admitting linear ranking functions can be proved to terminate by policy iteration (with an appropriate template):

Corollary 1. *If a program admits a linear ranking function r, then a policy iteration analysis with a template T including $-r$ can prove its termination.*

Conditional termination with a linear ranking function, however, is more complicated since condition (C3) make strong assumptions on the domain of the ranking function.

Example 11. The program

```
1: while x ≥ 0 do
2:     x ← x + y
3: end while
```

admits x as a linear ranking function when $y < -1$. However, an analysis with the template $(-x, -y)$ cannot give any result of the form $A^\sharp = (x >= 0, y >= -1)$ because the domain of the ranking function associated to x should be all the states satisfying $x < 0$. More generally, while the set of non-terminating states is $(x >= 0, y >= 0)$, we can prove that no template can give this result.

On the other hand, with an initial constraint of the form $y \leq -\epsilon < 0$, the analysis proves the termination of the program.

5.3 On the Well-founded Relations

Theorem 5 proposes well-founded relations $\prec^i$ defined from (infinite) sets of iterates of the abstract computation. However, relations found in the literature are generally defined directly. For example, a common order used in linear ranking functions is $\prec_\epsilon$ (with $\epsilon > 0$) defined as:

$$a \prec_\epsilon b \iff a + \epsilon \leq b$$

Such an order is useful because checking $a \prec_\epsilon b$ is easier, and because it shows the evolution of the states towards termination.

In this section we study the possibility of constructing theses kinds of orders (not based on infinite sets) from the analysis. Our first step is consider the sequence of policies. This sequence constructs a finite and decreasing chain of $p+1$ fixpoints $\rho_0 = (+\infty), \ldots, \rho_p$. Projecting this chain to the i-th component gives a decreasing sequence $\rho_0^i = +\infty, \ldots, \rho_p^i$. As a result, any couple (a, b) where $a < -\rho_k^i \leq b$ can be included in $\prec^i$. Hence we can construct $\prec^i$ as:

$$\begin{aligned} \prec^i = \prec^{i,1} \cup \prec^{i,2} \cup \ldots \cup \prec^{i,p} \\ \cup\,]-\infty, -\rho_1^i[\times [-\rho_1^i, +\infty[\cup \ldots \cup\,]-\infty, -\rho_p^i[\times [-\rho_p^i, +\infty[\end{aligned} \tag{1}$$

where each $\prec^{i,k}$ is associated to the k-th policy and only defined on $[-\rho_{k-1}^i, -\rho_k^i[$.

Example 12. With program B, we need two well-founded orders $\prec^x$ and $\prec^{-x}$ on $\mathbb{R}$. Since the successive fixpoints (for the linear form x) gives $x \leq 100$ and $x \leq 1.6$ (cf. Fig. 3), we may construct $\prec^x$ as:

$$\begin{aligned} a \prec^x b \Leftrightarrow\ & a < -1.6 \leq b \\ & \text{or } a < -100 \leq b \\ & \text{or } (a, b) \in [-100, -1.6[\wedge a \prec^{x,3} b \\ & \text{or } (a, b) \in]-\infty, -100[\wedge a \prec^{x,1} b \end{aligned}$$

where $\prec^{x,1}$ (resp. $\prec^{x,3}$) is a well-founded suborder of $<$ on $]-\infty, -100[$ (resp. $[-100, -1.6[$).

Finding an order for each policy is difficult. Let's restrict ourselves to the case of a LSL. The policy is described as a system of affine equations, and the next fixpoint as the limit of a sequence $(\boldsymbol{\nu}_k)$ of the form:

$$\begin{aligned} \boldsymbol{\nu}_0 &= \rho_j \\ \boldsymbol{\nu}_{k+1} &= A\boldsymbol{\nu}_k + B \end{aligned}$$

where A is a nonnegative matrix. Then we have $\boldsymbol{\nu}_{k-1} - \boldsymbol{\nu}_k = A^k(\boldsymbol{\nu}_0 - \boldsymbol{\nu}_1)$ (where $\boldsymbol{\nu}_0 - \boldsymbol{\nu}_1$ is also nonnegative). Using [20, Sect. 9.3], we get the following proposition:

Proposition 4. *Let P be a LSL, and $\rho_0, \ldots, \rho_p$ the sequence constructed by policy iteration on P, ρ_{k-1}^i and ρ_k^i the i-th component of ρ_{k-1} and ρ_k. Then there exists a well-founded order $\prec^{i,k}$ for equation (1) and a integer $d > 0$ such that:*

– *if* $\rho_k = -\infty$,

$$\exists \epsilon > 0, (\prec^{i,k})^d \subseteq \prec_\epsilon \quad \textit{where } a \prec_\epsilon b \Leftrightarrow a + \epsilon \leq b$$

– *otherwise:*

$$\exists h > 1, (\prec^{i,k})^d \subseteq \prec_{h,-\rho^i_k} \quad \textit{where } a \prec_{h,-\rho^i_k} b \Leftrightarrow (-\rho^i_k - a) \geq h(-\rho^i_k - b)$$

In this proposition, $(\prec^{i,k})^d = \prec^{i,k} \circ \ldots \circ \prec^{i,k}$ is the d-th power of $\prec^{i,k}$. Note that $\prec_\epsilon$ or $\prec_{h,-\rho^i_k}$ are well-founded orders on $[-\rho^i_{k-1}, -\rho^i_k[$, which implies that $\prec^{i,k}$ is a well-founded order.

This proposition does not directly give $\prec^{i,k}$, but it shows that decreasing sequences decreases (at least) on average linearly when $\rho_k = -\infty$ and geometrically (from the upper bound) when ρ_k is finite. Although this results is not proven on the general case, we expect the progressions to be similar in most cases.

Example 13. Continuing the previous example, we consider $\prec^{x,1}$ on $]-\infty, -100[$ and $\prec^{x,3}$ on $[-100, -1.6[$. The first policy (associated to $\prec^{x,1}$) returns the next fixpoint after one iteration. Hence we can just define $\prec^{x,1} = \emptyset$.

The third policy (associated to $\prec^{x,3}$) converges with a geometric rate. The order $\prec^{x,3}$ can be defined as:

$$u \prec^{x,3} u' \iff (-1.6 - u) \geq 4(-1.6 - u')$$

Concretely, this results shows that from an initial value $x > 1.6$, the value of x (when $x > 0$) diverges from 1.6 at a geometric rate.

6 Experiments

6.1 Template Selection

A prototype analyzer implementing the policy iteration algorithm was developed, using VPL[2] to handle exact polyhedral and linear programming operations. Since our analysis use the template polyhedral domain, selecting a correct template was an issue. However, overapproximating the greatest fixpoint enables a progressive refinement of the template: any fixpoint computed with a template can be used as a starting post-fixpoint with another template. Based on this idea, the following heuristics was implemented:

1. first, a few backwards steps in the general polyhedral domain is computed;
2. then the actual linear constraints are used as a basis for the template and the abstract fixpoint is computed for this template;
3. the process can be iterated from the new post-fixpoint, alternating between backward direct iterations and policy iterations. Since every intermediate result is a safe approximation, we can stop anytime, or if the fixpoint is reached.

[2] `http://verasco.imag.fr/wiki/VPL`

This heuristics can be compared to existing techniques for invariant analyses which uses partial traces to specialize the abstract domain [3,28]. In the worst case, the analysis gives the same result as a classical narrowing, returning an approximation of the states which do not terminate after n loop iterations. In the best case, it gives the abstract greatest fixpoint in the (general) polyhedral domain, which may not be the exact set of non-terminating states.

Example 14. In the terminating program (from [9]):

```
1: while x ≠ 0 do
2:     if x > 0 then
3:         x ← x − 1
4:     else
5:         x ← x + 1
6:     end if
7: end while
```

the decreasing iterations in the polyhedral domain stabilizes immediately at $]-\infty, +\infty[$. The problem can be solved by using several program points (or, similarly, state partitioning) to separate the cases $x > 0$ and $x < 0$.

6.2 Results

LSL Test Suite. First, we used the LSL test suite proposed by Chen *et al.*[5]. This suite has 38 LSL loops, of which 12 are non-terminating, 7 are terminating with linear ranking functions, and 19 are terminating with non-linear ranking functions. Our prototype analyzed the whole test suite in 0.5 second. The results are summarized on Table 1. Terminating LSLs with a linear ranking function are all proved to terminate (although our heuristics does not guarantee this). Terminating LSLs without a linear ranking function are proved to terminate most of the time, yet our analyzer failed in 6 cases whereas Chen *et al.*'s algorithm, which is specifically designed to prove termination on LSLs, failed only twice. Interpreting the results of non-terminating LSLs is harder since this test suite was not designed for conditional termination analysis. For 2 LSLs, our approach managed to find the exact set of terminating states, something which was not possible with only narrowings. For 3 LSLs, the greatest fixpoint is directly reached by a few iterations in the polyhedral domain. Finally, for the other programs, the PI techniques does not refine the decreasing iteration sequence.

Table 1. Experiment results

LSL test suite[5]			
Programs			Results
38 LSLs	Terminating : 26	Linear r.f.: 7 Non-linear r.f.: 19	Termination proved: 7 Termination proved: 13
	Non-terminating : 12		Exact semantics with PI: 2 Exact semantics with narrowing: 3 No improvement: 7

These results show that our approach is quite fast and can find complex termination properties, but not as efficient as a technique specifically designed for linear simple loops.

Other Programs. To our knowledge, no test suite exists for termination analysis (or conditional termination) on general programs. Hence we tested some examples given on previous works. The analyzes were fast and sometimes successful. However, several cases failed to give interesting results. We identified two main causes.

1. The iterates in the polyhedral domain stabilize after a few iterations, as in Example 14. This problem is directly related to the use of polyhedral abstractions.
2. Termination (or interesting conditional termination) requires the use of lexicographic ordering [9]. Our approach seems to be more suited to prove termination when the ranking relations are interlinked than when they form a lexicographic order. This is especially interesting as other approaches (limited to termination analysis) are specifically designed for lexicographic ordering [9,2]. Hence our approach can be used in complement to those.

For both problems, partitioning the set of states should improve the results.

7 Conclusion

This paper has described how policy iteration can be used to find conditional termination properties. The analysis is fast and the result can be precise, although it relies heavily on the abstract domain used. To improve the analysis, we plan to investigate the application of dynamic trace partitioning [26] for conditional termination. Another possibility would be to extend the policy iteration framework to other abstract domains such as the generalized template domain [6] or quadratic templates [1].

Acknowledgements. The author thanks D. Monniaux, L. Gonnord and S. Putot as well as the anonymous referees for their comments and suggestions.

References

1. Adjé, A., Gaubert, S., Goubault, E.: Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In: Gordon, A.D. (ed.) ESOP 2010. LNCS, vol. 6012, pp. 23–42. Springer, Heidelberg (2010)
2. Alias, C., Darte, A., Feautrier, P., Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In: Cousot, Martel (eds.) [15], pp. 117–133
3. Amato, G., Parton, M., Scozzari, F.: Deriving numerical abstract domains via principal component analysis. In: Cousot, Martel (eds.) [15], pp. 134–150

4. Bozga, M., Iosif, R., Konečný, F.: Deciding conditional termination. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 252–266. Springer, Heidelberg (2012)
5. Chen, H.Y., Flur, S., Mukhopadhyay, S.: Termination proofs for linear simple loops. In: Miné, A., Schmidt, D. (eds.) SAS 2012. LNCS, vol. 7460, pp. 422–438. Springer, Heidelberg (2012)
6. Colón, M.A., Sankaranarayanan, S.: Generalizing the template polyhedral domain. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 176–195. Springer, Heidelberg (2011)
7. Cook, B., Gulwani, S., Lev-Ami, T., Rybalchenko, A., Sagiv, M.: Proving conditional termination. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 328–340. Springer, Heidelberg (2008)
8. Cook, B., Podelski, A., Rybalchenko, A.: Abstraction refinement for termination. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 87–101. Springer, Heidelberg (2005)
9. Cook, B., See, A., Zuleger, F.: Ramsey vs. lexicographic termination proving. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 47–61. Springer, Heidelberg (2013)
10. Costan, A., Gaubert, S., Goubault, É., Martel, M., Putot, S.: A policy iteration algorithm for computing fixed points in static analysis of programs. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 462–475. Springer, Heidelberg (2005)
11. Cousot, P.: Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Thèse d'État ès sciences mathématiques, Université Joseph Fourier, Grenoble, France (March 21, 1978)
12. Cousot, P., Cousot, R.: An abstract interpretation framework for termination. In: Conference Record of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Philadelphia, PA, January 25-27, pp. 245–258. ACM Press, New York (2012)
13. Cousot, P.: Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In: Cousot (ed.) [14], pp. 1–24
14. Cousot, R. (ed.): VMCAI 2005. LNCS, vol. 3385. Springer, Heidelberg (2005)
15. Cousot, R., Martel, M. (eds.): SAS 2010. LNCS, vol. 6337. Springer, Heidelberg (2010)
16. Dai, L., Xia, B.: Non-termination sets of simple linear loops. In: Roychoudhury, A., D'Souza, M. (eds.) ICTAC 2012. LNCS, vol. 7521, pp. 61–73. Springer, Heidelberg (2012)
17. De Nicola, R. (ed.): ESOP 2007. LNCS, vol. 4421. Springer, Heidelberg (2007)
18. Gaubert, S., Goubault, E., Taly, A., Zennou, S.: Static analysis by policy iteration on relational domains. In: De Nicola (ed.) [17], pp. 237–252
19. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: De Nicola (ed.) [17], pp. 300–315
20. Hogben, L.: Handbook of Linear Algebra, 1st edn. Discrete Mathematics and Its Applications. Chapman & Hall/CRC (2007)
21. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
22. Kroening, D., Sharygina, N., Tsitovich, A., Wintersteiger, C.M.: Termination analysis with compositional transition invariants. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 89–103. Springer, Heidelberg (2010)

23. Massé, D.: Proving termination by policy iteration. Electr. Notes Theor. Comput. Sci. 287, 77–88 (2012)
24. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
25. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS, pp. 32–41. IEEE Computer Society (2004)
26. Rival, X., Mauborgne, L.: The trace partitioning abstract domain. ACM Trans. Program. Lang. Syst. 29(5) (2007)
27. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Cousot (ed.) [14], pp. 25–41
28. Seladji, Y., Bouissou, O.: Fixpoint computation in the polyhedra abstract domain using convex and numerical analysis tools. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 149–168. Springer, Heidelberg (2013)
29. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004)
30. Urban, C.: The abstract domain of segmented ranking functions. In: Logozzo, F., Fähndrich, M. (eds.) SAS 2013. LNCS, vol. 7935, pp. 43–62. Springer, Heidelberg (2013)