Deriving Numerical Abstract Domains via Principal Component Analysis

Gianluca Amato, Maurizio Parton, and Francesca Scozzari

Università di Chieti-Pescara – Dipartimento di Scienze

Abstract. We propose a new technique for developing ad-hoc numerical abstract domains by means of statistical analysis. We apply Principal Component Analysis to partial execution traces of programs, to find out a "best basis" in the vector space of program variables. This basis may be used to specialize numerical abstract domains, in order to enhance the precision of the analysis. As an example, we apply our technique to interval analysis of simple imperative programs.

1 Introduction

Numerical abstract domains are widely used to prove properties of program variables such as "all the array indexes are contained within the correct bounds" or "division by zero cannot happen". Moreover, numerical properties may help other kind of analyses, such as termination analyses [6], timing analyses [15], shape analyses [5], string cleanness analyses [10] and so on. Many numerical abstract domains strive to trade the accuracy of convex polyhedra [9] for higher speed (for instance, see the Octagon domain in [21]).

The precision of the analyses may often be improved with the use of special-purpose abstract domains, such as the domains for the analysis of digital filters [11], or the arithmetic-geometric progression abstract domain [12]. This idea may be pushed further by devising domains not just for a class of applications, but for a single program. For example, if we know the general form of the while-loop invariants which occur in a program, domains able to express these invariants should reach a higher precision than others.

In this paper we describe a family of ad-hoc domains and provide a fully automatic mechanism which, starting from an approximation of the concrete semantics of a program, selects the best domain in the family.

Consider the program in Figure 1, where the parameter x is the input and y is a local variable, and its partial execution trace for the input $x = 10$ which stops after 5 iterations of the `while` statement. Collecting the values for the variables x and y at different program points (after each assignment), we obtain the table in Figure 2. If we abstract this set of values using the box domain [7] of Cartesian product of intervals, we get the shaded area in Figure 3, given by

$$\begin{cases} 5 \leq x \leq 10 \\ -10 \leq y \leq -5 \end{cases}$$

R. Cousot and M. Martel (Eds.): SAS 2010, LNCS 6337, pp. 134–150, 2010.

```
xyline = function(x)
{
   assume(x>=0)
   y=-x
   while(x>y) {
      x= x-1
      y= y+1
   }
}
```

Fig. 1. The example program `xyline`

x	y
10	-10
9	-10
9	-9
8	-9
8	-8
7	-8
7	-7
6	-7
6	-6
5	-6
5	-5

Fig. 2. A partial execution trace of the example program

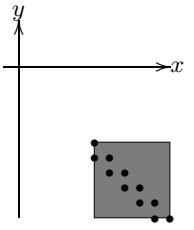

Fig. 3. Representation of the partial execution trace and relative box abstraction

The key point is that the abstraction in the box domain depends on the coordinate system we choose to draw the boxes. With the standard choice of (x, y) as coordinate system, the box in Figure 3 is a very rough approximation of the partial trace, but we can improve the precision by conveniently changing the axes. For instance, consider a different coordinate system whose axes x', y' are clockwise rotated by 30 degrees. The abstraction in this "rotated box domain" is depicted in Figure 4. The two boxes in Figure 4 are incomparable as sets of points, nonetheless the rotated box seems to fit better: for example, it has a smaller area. The question is how to find a "best rotation".

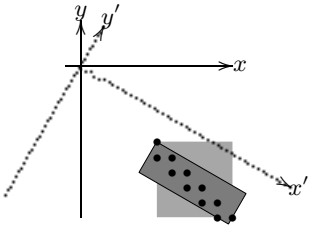

Fig. 4. Abstraction with boxes rotated by 30 degrees

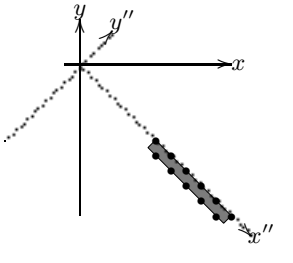

Fig. 5. Abstraction with boxes rotated by 45 degrees

To this aim, we use a statistical tool called *Principal Component Analysis* (PCA). The intuitive idea of PCA is to choose the axes maximizing the variance of the collected values. More explicitly, PCA finds a new orthonormal coordinate system such that the variance of the projection of the data points on the first axis is the maximum among all possible directions, the variance of the projection of the data points on the second axis is the maximum among all possible directions which are orthogonal to the first axis, and so on. In our example, the greatest variance is obtained by projecting the data points along the line $y = -x$. An orthogonal coordinate system (x', y') with the first axis corresponding to this

line may be obtained by a 45 degree clockwise rotation. The abstraction with respect to the box domain in (x'', y'') is depicted in Figure 5, and in the original coordinates it is given by:

$$\begin{cases} 10 \leq x - y \leq 20 \\ -1 \leq x + y \leq 0 \end{cases}$$

It is worth noting that the `while` invariant in our example is $x+y = 0, x-y \geq 0$, and it may be expressed only in the domain of 45 degree rotated boxes. This suggests that, using rotated boxes as abstract objects, an abstract interpretation based analyzer could infer this invariant. More generally, our intuition says that, if we consider well-known numerical abstract domains and adapt them to work with non-standard coordinate systems, we can improve the precision of the analysis without much degradation of performance. In this paper we develop the theoretical foundation and the implementation to validate this intuition, using the box domain as a case study. In Section 6, we will show that our analysis actually infers the invariant $x + y = 0, x - y \geq 0$.

The paper is structured as follows. Section 2 introduces some notations. Section 3 presents the abstract domains of parallelotopes, i.e. boxes w.r.t. non-standard coordinate systems, while Section 4 gives the abstract operators. Section 5 introduces PCA, used to automatically derive the "best coordinate system". Section 6 presents the prototype implementation we have developed in the R programming language [23], and shows some experimental results. Finally, in Section 8 we discuss ideas on future work.

2 Notations

Linear Algebra. We denote by $\bar{\mathbb{R}}$ the ordered field of real numbers extended with $+\infty$ and $-\infty$. Addition and multiplication are extended to $\bar{\mathbb{R}}$ in the obvious way, with the exception that 0 times $\pm\infty$ is 0. We use boldface for elements $\boldsymbol{v}$ of $\bar{\mathbb{R}}^n$. Given $\boldsymbol{u}, \boldsymbol{v} \in \bar{\mathbb{R}}^n$, and a relation $\bowtie \in \{<, >, \leq, \geq, =\}$, we write $\boldsymbol{u} \bowtie \boldsymbol{v}$ if and only if $u_i \bowtie v_i$ for each $i \in \{1, \ldots, n\}$. We denote by $\cdot$ the *dot product* on $\bar{\mathbb{R}}^n$, namely, $\boldsymbol{u} \cdot \boldsymbol{v} \stackrel{\text{def}}{=} u_1 v_1 + \cdots + u_n v_n$.

If $A = (a_{ij})$ is a matrix, we denote by A^T its *transpose*. If A is invertible, A^{-1} denotes its inverse, and $\mathrm{GL}(n)$ is the group of $n \times n$ invertible matrices. The identity matrix in $\mathrm{GL}(n)$ is denoted by I_n, and any $A \in \mathrm{GL}(n)$ such that $AA^T = I_n$ is called an *orthogonal* matrix. Clearly, any $1 \times n$-matrix can be viewed as a vector: in particular, we denote by $\boldsymbol{a}_{i*}$ (respectively $\boldsymbol{a}_{*j}$) the vector given by the i-th row (respectively the j-th column) of any $n \times n$-matrix A. If A is orthogonal, then the vectors $\boldsymbol{a}_{i*}$ are *orthonormal* (they have length 1 and are orthogonal), and thus are linearly independent. The same holds for vectors $\boldsymbol{a}_{*j}$. The standard orthonormal basis of $\mathbb{R}^n$ is denoted by $\{\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n\}$.

Abstract Interpretation. (See [8] for details). Given complete lattices $(C, \leq_C)$ and $(A, \leq_A)$, respectively called the *concrete domain* and the *abstract domain*,

a *Galois connection* is a pair (α, γ) of monotone maps $\alpha : C \to A$, $\gamma : A \to C$ such that $\alpha\gamma \leq_A \mathrm{id}_A$ and $\gamma\alpha \geq_C \mathrm{id}_C$. If $\alpha\gamma = \mathrm{id}_A$, then (α, γ) is called a *Galois insertion*. Given a monotone map $f : C \to C$, the map $\tilde{f} : A \to A$ is a *correct approximation* of f if $\alpha f \leq \tilde{f}\alpha$. The *best correct approximation* of f is the smallest correct approximation f^α of f. It is well-known that $f^\alpha = \alpha f \gamma$.

Boxes. A set $\mathcal{B} \subseteq \mathbb{R}^n$ is called a (closed) *box* if there are *bounds* $\boldsymbol{m}, \boldsymbol{M} \in \bar{\mathbb{R}}^n$ such that

$$\mathcal{B} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{m} \leq \boldsymbol{x} \leq \boldsymbol{M}\} \ .$$

We denote such a box with $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$. Boxes are used to abstract subsets of $\mathbb{R}^n$. If $\mathbb{B}\mathsf{ox}$ is the set of all the boxes, a Galois insertion $(\alpha^{\mathbb{B}}, \gamma^{\mathbb{B}}) : \wp(\mathbb{R}^n) \rightleftharpoons \mathbb{B}\mathsf{ox}$ may be defined by letting $\alpha^{\mathbb{B}}(\mathcal{C})$ be the smallest box enclosing $\mathcal{C}$ and $\gamma^{\mathbb{B}}(\mathcal{B}) = \mathcal{B}$.

Given two boxes $\langle \boldsymbol{m}, \boldsymbol{M} \rangle, \langle \boldsymbol{m}', \boldsymbol{M}' \rangle \in \mathbb{B}\mathsf{ox}$, we will use the following notation for the standard box operations (see [7]):

- The abstract union operation $\langle \boldsymbol{m}, \boldsymbol{M} \rangle \cup^{\mathbb{B}} \langle \boldsymbol{m}', \boldsymbol{M}' \rangle$ yields the smallest box containing both $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$ and $\langle \boldsymbol{m}', \boldsymbol{M}' \rangle$;
- The abstract intersection operation $\langle \boldsymbol{m}, \boldsymbol{M} \rangle \cap^{\mathbb{B}} \langle \boldsymbol{m}', \boldsymbol{M}' \rangle$ computes the greatest box contained in both $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$ and $\langle \boldsymbol{m}', \boldsymbol{M}' \rangle$;
- $\mathsf{assign}^{\mathbb{B}}(i, \boldsymbol{a}, b) : \mathbb{B}\mathsf{ox} \to \mathbb{B}\mathsf{ox}$ corresponds to the (linear) assignment "$x_i = \boldsymbol{a} \cdot \boldsymbol{x} + b$", where $\boldsymbol{x}$ is a vector of n variables, $i \in \{1, \ldots n\}$, $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. It is the best correct abstraction of the concrete operation $\mathsf{assign}(i, \boldsymbol{a}, b) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ defined as the pointwise extension of:

$$\mathsf{assign}(i, \boldsymbol{a}, b)(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{y} \quad \text{where} \quad y_j = \begin{cases} x_j & \text{if } j \neq i \ , \\ (\boldsymbol{a} \cdot \boldsymbol{x}) + b & \text{if } j = i \ . \end{cases}$$

- $\mathsf{test}^{\mathbb{B}}(\boldsymbol{a}, b, \bowtie) : \mathbb{B}\mathsf{ox} \to \mathbb{B}\mathsf{ox}$ corresponds to the *then*-branch of the if-statement "`if (`$\boldsymbol{a} \cdot \boldsymbol{x} \bowtie b$`)`", where $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$. It is the best correct abstraction of the concrete operation $\mathsf{test}(\boldsymbol{a}, b, \bowtie) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ defined as:

$$\mathsf{test}(\boldsymbol{a}, b, \bowtie)(\mathcal{C}) \stackrel{\text{def}}{=} \mathcal{C} \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \bowtie b\} \ .$$

  The abstract operation $\mathsf{test}^{\mathbb{B}}(\boldsymbol{a}, b, \bowtie)(\langle \boldsymbol{m}, \boldsymbol{M} \rangle)$ computes the smallest box which contains the intersection of $\langle \boldsymbol{m}, \boldsymbol{M} \rangle$ and the set of points $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \bowtie b\}$.

3 The Parallelotope Domains

Every choice of $A \in \mathrm{GL}(n)$ gives new coordinates in $\mathbb{R}^n$, and boxes with respect to this transformed coordinates are called *parallelotopes*. Thus, a parallelotope is a box whose edges are parallel to the axes in the new coordinate system. Remark that we are not restricting to orthogonal change of basis. This means that we consider any invertible linear transformation, such as rotation, reflection, stretching, compression, shear or any combination of these. The aim of the change of coordinate system is to fit the original data with a higher precision than with standard boxes.

Example 1. Consider the set $\mathcal{C} = \{(u, -u) \mid u \geq 0\} \subseteq \mathbb{R}^2$ corresponding to the `while` invariant $x+y = 0, x-y \geq 0$ of program in Figure 1. If we directly abstract $\mathcal{C}$ in the box domain, we get $\alpha^{\mathbb{B}}(\mathcal{C}) = \langle (0, -\infty), (+\infty, 0)\rangle$, and $\gamma^{\mathbb{B}}(\alpha^{\mathbb{B}}(C)) = \mathbb{R}^+ \times \mathbb{R}^-$, with a sensible loss of precision. Let us consider a clockwise rotation of 45 degrees, centered on the origin, of the standard coordinate system. The matrix

$$A = \begin{bmatrix} \cos(-\frac{\pi}{4}) & -\sin(-\frac{\pi}{4}) \\ \sin(-\frac{\pi}{4}) & \cos(-\frac{\pi}{4}) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

transforms rotated coordinates into standard coordinates.

We want to abstract $\mathcal{C}$ with boxes on the rotated coordinate system. To this aim, we first compute the rotated coordinates of the points in $\mathcal{C}$, and then compute the smallest enclosing box. Since the rotated coordinates are given by $A^{-1}(x, y)^T$, we obtain:

$$\begin{aligned} \alpha^{\mathbb{B}}(A^{-1}\mathcal{C}) &= \alpha^{\mathbb{B}}(\{A^{-1}\boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{C}\}) \\ &= \alpha^{\mathbb{B}}\left(\left\{ \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} u \\ -u \end{bmatrix} \mid u \in \mathbb{R}^+ \right\}\right) \\ &= \alpha^{\mathbb{B}}\left(\left\{ \begin{bmatrix} u\sqrt{2} \\ 0 \end{bmatrix} \mid u \in \mathbb{R}^+ \right\}\right) = \langle (0,0), (+\infty, 0)\rangle \ . \end{aligned}$$

The axes in the rotated coordinate system are, respectively, the lines $y = -x$ and $y = x$ in the standard coordinate system. It means that the box $\langle (0,0), (+\infty, 0)\rangle$ computed above may be represented algebraically in the standard coordinate system as

$$\begin{cases} 0 \leq x + y \leq 0 \\ 0 \leq x - y \leq +\infty \end{cases}$$

More in general, using the matrix A, we may represent all the parallelotopes of the form

$$\begin{cases} m_1 \leq x + y \leq M_1 \\ m_2 \leq x - y \leq M_2 \end{cases}$$

Thus, we have transformed a non-relational analysis into a relational one, where the form of the relationships is given by the matrix A. If we concretize the box by applying $\gamma^{\mathbb{B}}$ and using the matrix A to convert the result to the standard coordinate system, we obtain $A\gamma^{\mathbb{B}}\alpha^{\mathbb{B}}(A^{-1}\mathcal{C}) = \mathcal{C}$. Thus, we get a much better precision than using standard boxes. We stress out that we need to choose A cleverly, on the base of the specific data set, otherwise we may loose precision: for example, if $\mathcal{D} = \{(u, 0) \mid u \in \mathbb{R}\}$, then $\gamma^{\mathbb{B}}(\alpha^{\mathbb{B}}(\mathcal{D})) = \mathcal{D}$ but $A\gamma^{\mathbb{B}}\alpha^{\mathbb{B}}(A^{-1}\mathcal{D}) = \mathbb{R}^2$.

It is worth noting that, if we prefer not to deal with irrational numbers, we may choose the transformation matrix

$$A' = \sqrt{2}\ A = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

This corresponds to a 45 degree clockwise rotation followed by a scaling by $\sqrt{2}$ in all directions.

In order to define the abstract domains of parallelotopes, we use the same complete lattice $\mathbb{B}\mathsf{ox}$ we used for the box domain, but equipped with a different abstraction function, and different abstract operations.

Definition 1 (The Parallelotope Domains). *Given $A \in \mathrm{GL}(n)$, we define the maps $\gamma^A : \mathbb{B}\mathsf{ox} \to \wp(\mathbb{R}^n)$ and $\alpha^A : \wp(\mathbb{R}^n) \to \mathbb{B}\mathsf{ox}$ as*

$$\gamma^A(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) \stackrel{\text{def}}{=} A\gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) \ ,$$
$$\alpha^A(\mathcal{C}) \stackrel{\text{def}}{=} \alpha^{\mathbb{B}}(A^{-1}\mathcal{C}) \ .$$

Using the above definition, it is easy to check that (α^A, γ^A) is a Galois insertion. Intuitively, the abstraction α^A first projects the points into the new coordinate system, then computes the standard box abstraction. The concretization map γ^A performs the opposite process. Remark that, as a particular case, we have $\alpha^{I_n} = \alpha^{\mathbb{B}}$ and $\gamma^{I_n} = \gamma^{\mathbb{B}}$.

4 Abstract Operations on Parallelotopes

In this section we illustrate the main abstract operations on the Parallelotope domains. We show that, in most cases, the abstract operators can be easily recovered by the corresponding operators on boxes. In all the operations, we ignore the computational cost of computing the inverse of the matrix A. If A is orthogonal, the cost may be considered constant since $A^{-1} = A^T$ and we do not need to compute the transpose: it is enough to consider specific algorithms which performs transposition "on the fly" when needed. If A is not orthogonal, the inverse may be computed with standard algorithms which have complexities between quadratic and cubic. However, A^{-1} needs to be computed only once for the entire execution of the abstract interpretation procedure, hence its computational cost is much less relevant then the cost of the abstract operations.

4.1 Union and Intersection

Given $B_1, B_2 \in \mathbb{B}\mathsf{ox}$, the best correct approximation of the concrete union is:

$$B_1 \cup^A B_2 \stackrel{\text{def}}{=} \alpha^A(\gamma^A(B_1) \cup \gamma^A(B_2)) \ .$$

By replacing α^A and γ^A with their definitions, A and A^{-1} cancel out and we have that $\cup^A$ is the same as $\cup^{\mathbb{B}}$. The same holds for intersection.

Proposition 1 (Union and intersection). *Given $B_1, B_2 \in \mathbb{B}\mathsf{ox}$, we have that:*

$$B_1 \cup^A B_2 = B_1 \cup^{\mathbb{B}} B_2 \qquad B_1 \cap^A B_2 = B_1 \cap^{\mathbb{B}} B_2$$

The computational complexity of both operations is $O(n)$.

4.2 Assignment

The abstract operation $\mathsf{assign}^A(i, \boldsymbol{a}, b)$ corresponds to the (linear) assignment "$x_i = \boldsymbol{a} \cdot \boldsymbol{x} + b$", where $\boldsymbol{x}$ is a vector of n variables, $i \in \{1, \ldots n\}$, $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We look for a constructive characterization of the best correct approximation, defined as:

$$\mathsf{assign}^A(i, \boldsymbol{a}, b) \stackrel{\text{def}}{=} \alpha^A \, \mathsf{assign}(i, \boldsymbol{a}, b) \, \gamma^A \ .$$

Let us note that the concrete operation may be rewritten using matrix algebra as $\mathsf{assign}(i, \boldsymbol{a}, b)(\boldsymbol{x}) = Z_{i,\boldsymbol{a}} \boldsymbol{x} + b\boldsymbol{e}^i$ where

$$Z_{i,\boldsymbol{a}} = I + \boldsymbol{e}^i \cdot \boldsymbol{a}' \quad \text{with} \quad a'_j = \begin{cases} a_j & \text{if } j \neq i, \\ a_i - 1 & \text{if } j = i. \end{cases}$$

This allows us to prove the following:

Theorem 1 (Assignment). *Given $\langle \boldsymbol{m}, \boldsymbol{M} \rangle \in \mathbb{B}\mathsf{ox}$, we have that*

$$\mathsf{assign}^A(i, \boldsymbol{a}, b)(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) = \langle \boldsymbol{m}' + A^{-1} b\boldsymbol{e}^i, \boldsymbol{M}' + A^{-1} b\boldsymbol{e}^i \rangle$$

where

$$\boldsymbol{m}' = \inf_{\boldsymbol{x} \in \langle \boldsymbol{m}, \boldsymbol{M} \rangle} (H^T \boldsymbol{e}^i) \cdot \boldsymbol{x} \qquad \boldsymbol{M}' = \sup_{\boldsymbol{x} \in \langle \boldsymbol{m}, \boldsymbol{M} \rangle} (H^T \boldsymbol{e}^i) \cdot \boldsymbol{x} \ ,$$

and $H = A^{-1} Z_{i,\boldsymbol{a}} A$. The complexity is $O(n^2)$.

Proof (Sketch). We may rewrite the abstract operator as:

$$\begin{aligned} & \alpha^A(\mathsf{assign}(i, \boldsymbol{a}, b)(\gamma^A(\langle \boldsymbol{m}, \boldsymbol{M} \rangle))) \\ = \ & \alpha^{\mathbb{B}}(A^{-1} \mathsf{assign}(i, \boldsymbol{a}, b)(A \gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle))) \\ = \ & \alpha^{\mathbb{B}}(A^{-1} Z_{i,\boldsymbol{a}} A \ \langle \boldsymbol{m}, \boldsymbol{M} \rangle + A^{-1} b\boldsymbol{e}^i) \ . \end{aligned}$$

Let $H = A^{-1} Z_{i,\boldsymbol{a}} A$ and $H \ \langle \boldsymbol{m}, \boldsymbol{M} \rangle = \langle \boldsymbol{m}', \boldsymbol{M}' \rangle$. For each $i \in \{1, \ldots, n\}$, $m'_i = \inf_{\boldsymbol{x} \in \langle \boldsymbol{m}, \boldsymbol{M} \rangle} (H\boldsymbol{x}) \cdot \boldsymbol{e}^i = \inf_{\boldsymbol{x} \in \langle \boldsymbol{m}, \boldsymbol{M} \rangle} (H^T \boldsymbol{e}^i) \cdot \boldsymbol{x}$. Since $H^T \boldsymbol{e}^i$ is the transpose of the i-th row of H, we may compute $\inf_{\boldsymbol{x} \in \langle \boldsymbol{m}, \boldsymbol{M} \rangle} (H^T \boldsymbol{e}^i) \cdot \boldsymbol{x}$ using interval arithmetic.

4.3 Test

We want to find a constructive characterization of the best correct approximation $\mathsf{test}^A(\boldsymbol{a}, b, \leq)$ defined as:

$$\mathsf{test}^A(\boldsymbol{a}, b, \leq) \stackrel{\text{def}}{=} \alpha^A \, \mathsf{test}(\boldsymbol{a}, b, \leq) \, \gamma^A \ .$$

Given $\langle \boldsymbol{m}, \boldsymbol{M} \rangle \in \mathbb{B}\mathsf{ox}$, we have that

$$\begin{aligned} & \alpha^A(\mathsf{test}(\boldsymbol{a}, b, \leq)(\gamma^A(\langle \boldsymbol{m}, \boldsymbol{M} \rangle))) \\ = \ & \alpha^{\mathbb{B}}(A^{-1} \mathsf{test}(\boldsymbol{a}, b, \leq)(A \ \gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle))) \\ = \ & \alpha^{\mathbb{B}}(A^{-1}((A \ \gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle)) \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \leq b\})) \\ = \ & \alpha^{\mathbb{B}}(\gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) \cap \{A^{-1}\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \leq b\}) \\ = \ & \alpha^{\mathbb{B}}(\gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle) \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid (A^T \boldsymbol{a}) \cdot \boldsymbol{x} \leq b\}) \\ = \ & \alpha^{\mathbb{B}}(\mathsf{test}(\boldsymbol{a}, b, \leq)(\gamma^{\mathbb{B}}(\langle \boldsymbol{m}, \boldsymbol{M} \rangle))) \ . \end{aligned}$$

Hence, the abstract operator $\mathsf{test}^A(\boldsymbol{a}, b, \leq)$ can be easily computed by using the standard abstract operator on boxes, as $\mathsf{test}^A(\boldsymbol{a}, b, \leq) = \mathsf{test}^{\mathbb{B}}(A^T\boldsymbol{a}, b, \leq)$.

Proposition 2 (Test). *We have that*

$$\mathsf{test}^A(\boldsymbol{a}, b, \leq) = \mathsf{test}^{\mathbb{B}}(A^T\boldsymbol{a}, b, \leq) \ .$$

The computational complexity is $O(n^2)$.

The complexity of the algorithm to compute $\mathsf{test}^{\mathbb{B}}(\boldsymbol{a}, b, \leq)$ is $O(n)$. Thus, the complexity of computing $\mathsf{test}^A(\boldsymbol{a}, b, \leq)$ is $O(n^2)$, since we need to add the complexity for computing $A^T\boldsymbol{a}$. However, the latter might be computed only once in the analysis, and then memorized, in order to be reused every time we find such a conditional.

It is easy to see that, in the general case, the abstract counterpart of the operation $\mathsf{test}(\boldsymbol{a}, b, \bowtie)$ corresponding to the *then*-branch of the if-statement "`if` $(\boldsymbol{a} \cdot \boldsymbol{x} \bowtie b)$", where $\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$ and $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$, can be easily recovered by the corresponding operator on boxes. The same holds for the *else*-branch of the if-statement. For instance, the *else*-branch of "`if` $(\boldsymbol{a} \cdot \boldsymbol{x} \leq b)$" is exactly the *then*-branch of "`if` $(\boldsymbol{a} \cdot \boldsymbol{x} > b)$".

4.4 On the Implementation of Abstract Operators

Correctness of the abstract operators in actual implementations strictly depends on the exactness of matrix operations. The easiest way to ensure correctness is to use rational arithmetic. Alternatively, we could estimate the error of the floating point implementations of these operations, and use rounding to correctly approximate the abstract operators on real numbers, following the approach in [20].

Note that, although we have presented our domain as an abstraction of $\wp(\mathbb{R}^n)$, we may apply the same construction to build an abstraction of $\wp(\mathbb{Z}^n)$, in order to analyze programs with integer variables. In this case, whenever A is an integer matrix, we may perform almost all the computations on integers. In fact, observe that A^{-1} is an integer matrix divided by an integer number $d \in \mathbb{Z}$. Since all the operations involved in assign^A are linear, d may be factored out and only applied at the end, before rounding the intervals to integer bounds.

5 Principal Component Analysis

Principal component analysis (PCA) is a standard technique in statistical analysis which transforms a number of possibly correlated variables into uncorrelated variables called *principal components*, ordered by the most to the least important. Consider an $n \times m$ data matrix D on the field of real numbers. Each row may be though of as a different instance of a statistical population, while columns are attributes (see, for instance, the 11×2 matrix in Figure 2). Consider a vector $\boldsymbol{v} \in \mathbb{R}^m$, which expresses a linear combination of the attributes of

the population. The projection of the n rows of the matrix D onto the vector $\boldsymbol{v}$ is given by $D\boldsymbol{v}$. Among the different choices of $\boldsymbol{v}$, we are interested in the ones which maximize the (sample) *variance* of $D\boldsymbol{v}$. We recall that the variance of a vector $\boldsymbol{u} \in \mathbb{R}^m$ is $\sigma^2_{\boldsymbol{u}} = \frac{1}{m}\sum_{i=1}^{m}(u_i - \bar{u})^2$ where $\bar{\boldsymbol{u}}$ is the (empirical) *mean* of $\boldsymbol{u}$, i.e. $\bar{\boldsymbol{u}} = \frac{1}{m}\sum_{i=1}^{m} u_i$. Any unit vector which maximizes the variance may be chosen as the first principal component. This represents the axis which best explains the variability of data. The search for the second principal component is similar, looking for vectors $\boldsymbol{v'}$ which are orthonormal to the first principal component and maximize the variance of $D\boldsymbol{v'}$. In turn, the third principal component should be orthonormal to the first twos, with maximal variance, and so on.

From a mathematical point of view, principal component analysis finds an orthogonal matrix that transforms the data to a new coordinate system. The columns (called principal components) are ordered according to the variability of the data that they are able to express. It turns out that the columns are the eigenvectors of the *covariance matrix* of D, i.e. an $n \times n$ symmetric matrix Q such that q_{ij} is the (sample) *covariance* of d_{i*} and d_{j*}. We recall that the covariance of two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^m$ is $\sigma_{\boldsymbol{vw}} = \frac{1}{m}\sum_{i=1}^{m}(v_i - \bar{\boldsymbol{v}})(w_i - \bar{\boldsymbol{w}})$. The columns are ordered according to the corresponding eigenvalues. However, principal components are generally computed using singular value decomposition for a greater accuracy.

Example 2. Consider the partial execution trace in Figure 2 as data matrix D. If we perform the PCA on D, we get the principal components $(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ and $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$, corresponding to the change of basis matrix

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

given in Example 1.

5.1 Orthogonal Simple Component Analysis

It is worth noting that small changes in the data cause small changes in the principal components which, however, may cause a big loss in precision. This depends on the interactions between the PCA and the parallelotope abstraction function: If $\mathcal{D}$ is an unbounded set of points (in $\mathbb{R}^n$), the bounds (in $\bar{\mathbb{R}}$) of the minimum enclosing box of $\mathcal{D}$ are not continuous w.r.t. the change of basis matrix.

Example 3. We consider the 10×2 matrix obtained removing the last line from the table in Figure 2. If we perform the PCA, we get the change of basis matrix

$$S = \begin{bmatrix} s_1 & s_2 \\ -s_2 & s_1 \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{1}{2} + \frac{1}{2\sqrt{257}}} & \sqrt{\frac{1}{2} - \frac{1}{2\sqrt{257}}} \\ -\sqrt{\frac{1}{2} - \frac{1}{2\sqrt{257}}} & \sqrt{\frac{1}{2} + \frac{1}{2\sqrt{257}}} \end{bmatrix}$$

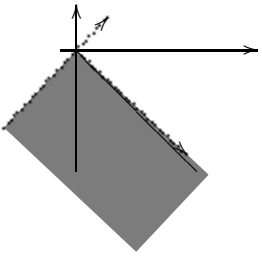

Fig. 6. Bad precision with PCA

corresponding to a clockwise rotation of about 43 degrees. The principal components are not very different from the previous ones, but now we are not able to represent parallelotopes bounded by constraints on $x + y$ and $x - y$. Therefore, the invariant $x + y = 0, x - y \geq 0$ cannot be represented directly. Since the difference between the first principal component and the axis $x + y = 0$ is unbounded, it is abstracted into $-\infty \leq s_2x + s_1y \leq 0$ and $0 \leq s_1x - s_2y \leq +\infty$, which is the shaded area in Figure 6. This cause a serious loss of accuracy.

In order to overcome the difficulties outlined above, we need a way of "stabilizing" the result of the PCA, so that it is less sensible to small changes in the data. There are two possible approaches to this problem: we may remove outliers (i.e., points which are very "different" from the others) by the execution trace before computing the PCA, or refine the result of the PCA. In this paper, we follow the second approach, since we prefer to maintain the whole set of original data. Our idea is that, in many cases, we expect that optimal parallelotopes which abstracts program states should contain only linear constraints with integer coefficients. This is obvious for programs with integer variables only (such as Example 1), or when we are interested in properties described by integer values (such as bounds of arrays, division by 0, etc...). Therefore, we would like to minimally change the result of the PCA (the matrix A) in such a way that A^{-1} is an integer matrix. Of course, the new matrix is not going to be the matrix with principal components anymore, but in this way we compensate for possible deviations of the principal components with respect to the "optimal" vectors.

There are several procedures in the statistic literature for simplifying the result of the PCA, in order to obtain integer matrices. In this paper we follow the approach of the *orthogonal simple component analysis*, introduced in [1]. The authors define a simplification procedure which transforms the result A of the PCA in an integer matrix B such that the columns of B are orthogonal and the angle between any column of A and the corresponding column of B is less then a specified threshold θ. Note that, in the general case, matrix B is not orthogonal because the columns of B are orthogonal but not orthonormal (i.e., their length is not one). This is not a problem since our domains of parallelotopes do not require matrices to be orthogonal. Note that, although B^{-1} may contain non-integer elements, each row is exactly an integer vector multiplied by a rational. Hence, it expresses integer constraints.

6 Implementation

In order to investigate on the feasibility of the ideas introduced above, we have developed a prototypical implementation for the intra-procedural analyses of a simple imperative language. The analyses may be performed with either the standard domain of boxes, the domains of parallelotopes, or with their combination. In order to collect the partial execution traces, the program under analysis is automatically augmented for recording the values of the variables at every program point. The implementation automatically recovers partial execution traces starting from the input values (which may be randomly generated or provided by the user), computes the orthogonal simple components, and performs static analysis with the three domains. Program equations are solved with a *recursive chaotic iteration strategy* on the *weak topological ordering* induced by the program structure (see [4]). The analyzer uses the standard widening [9] which extrapolates unstable bounds to infinity and the standard narrowing which improves infinite bounds only.

The prototype has been written in R [23], a language and environment for statistical computing. R is a functional language with call-by-value semantics, powerful meta-programming features, vectors as primitive data types and a huge library of built-in statistical functions. The benefits we got using R for developing our application were many. For example, thanks to the powerful meta-programming features, it was easy to augment programs with instructions which record the partial execution traces, and there was no need to implement a parser for the static analyzer. Actually, code can be manipulated programmatically in R, as in Lisp. Moreover, the vast library of statistical functions allowed us to implement easily the PCA (just a function call was sufficient) and the simplification procedure for obtaining the orthogonal simple components. Correctness of abstract operators was ensured using rational arithmetic.

The main drawback of R, at least for our application, is speed. Since it only supports call-by-value semantics, manipulating complex data structures may require several internal copy operations. For a prototype, this was deemed less important then fast coding. However, this means that we cannot compare the effective speed of the Parallelotope domains with the speed of octagons or polyhedra, because all the standard implementations of the latter domains, in libraries such as `APRON` [18] or `PPL` [2], are in C or C++.

6.1 Optimizing the Parallelotope Domains

Using the Parallelotope domains, we have occasionally experimented some problems in the bootstrap phase of the analysis. Consider the sample program `start1` in Figure 7. If we perform the analyses with the standard box domain, we may easily infer that, at the end of the function, both the variables x and y assume the value 10. However, using the Parallelotope domain with the axes clockwise rotated by 45 degrees, the analysis starts with the abstract state which covers the entire $\mathbb{R}^2$. The assignment $x = 10$ has no effect: since there are no bounds on the possible values for y, then nothing may be said about $x + y$ and $x - y$, even if we

```
start1 = function()
{
  x=10
  y=x
}
```

```
start2 = function(x)
{
  y=10
  x=y
}
```

```
cousot78 = function()
{
  i=2
  j=0
  while (TRUE) {
    if (i*i==4)
      i=i+4
    else {
      j=j+1
      i=i+2
} } }
```

Fig. 7. Example programs

know the value of x. Therefore, after the second assignment, we only know that $x - y = 0$, loosing precision with respect to the standard box analysis, although $x = y = 10$ may be expressed in the rotated domain as $x + y = 20, x - y = 0$.

The problem arises from the fact that assignments are naturally biased towards the standard axes, since the left hand side is always a variable. At the beginning of the analysis, when the abstract state does not contain any constraint, all constant assignments are lost, and this is generally unfavorable to the precision of the analysis. For this reason, our analyzer initializes all the local variables to zero, as done in many programming languages. Unfortunately, this does not always solve the problem, due to the presence of input parameters. Consider the program `start2` in Figure 7. In this case, we assume that $y = 0$ at the beginning of the function, but we cannot assume that $x = 0$, since this is a parameter. However, our parallelotope (the 45 degree clockwise rotated boxes) cannot express the fact that $y = 0$. Hence the abstract state at the beginning of the function is the full space $\mathbb{R}^2$, and the result at the end of the function is again $x - y = 0$. From the point of view of precision, an optimal solution to this kind of problems would be to use the reduced product of the box domain and Parallelotope domains. However, this may severely degrade performance. A good trade-off could be to perform both analysis in parallel: at the end of each abstract operations, we use the information which comes from one of the two domains to refine the other, and vice versa. Given a box and a parallelotope, a satisfactory and computationally affordable solution is to compute the smallest parallelotope which contains the box, and then the intersection between the two parallelotopes. The symmetric process can be used to refine the box. We have adopted this solution in our analyzer (see [11,12] for a similar approach).

6.2 Experimental Evaluation

We applied the analyzer to different toy programs we collected from the literature. Although an exhaustive comparison of the speed and precision of the domains of parallelotopes with other domains is outside the scope of this paper, we present here some preliminary results. We considered the following programs: `bsearch`: binary search over 100-element arrays, as appeared in [7]; `xyline`: the

program	Box	Parallelotope	combined	Octagon
`bsearch`	$1 \leq lwb \leq 100$ $1 \leq upb \leq 100$ $0 \leq m \leq 100$ $(-99 \leq upb - lwb)$ $(-100 \leq m - lwb)$	$0 \leq upb - lwb$	as box+ptope	as box+ptope+ $-99 \leq m - lwb$
`bsearch`*	as above	$0 \leq upb - lwb$ $-101 \leq -upb - lwb + 2m \leq 50.5$ $(-50.5 \leq m - lwb \leq 74.75)$	as box+ptope	as above
`xyline`		$-x + y \leq 0$ $x + y = 0$	as ptope	as ptope
`bsort`	$1 \leq b \leq +\infty$ $0 \leq j \leq +\infty$ $0 \leq t \leq +\infty$		as box	$1 \leq b \leq 100$ $0 \leq j \leq 100$ $0 \leq t \leq 99$ $b + j \leq 199$ $b + t \leq 198$ $0 \leq j - t$ $0 \leq b - t$
`bsort`*	as above	$1 \leq b$	$1 \leq b \leq 100$ $0 \leq j \leq 100$, $0 \leq t \leq 99$ $0 \leq j - t$	as above
`cousot78`	$2 \leq i$ $0 \leq j$	$2 \leq i + j$ $-i + j \leq -2$	as box+ptope	as box+ptope
`cousot78`†	as above	$-\infty \leq -i + 2j \leq -2$	as box+ptope	as box+ptope

Fig. 8. Results of the analyses for several programs and domains. Constraints in parentheses are not part of the result of the analyses, but may be inferred from them.

example program in Figure 1; `bsort`: bubblesort over 100-element arrays, which is the first example program in [9]; `cousot78`: the program in Figure 7, which is an instance of a skeletal program in [9].

All programs have at least one loop. For each program, we show the abstract state inferred by the analyzer at the beginning of the loop. Since `bsort` has two nested loops, we only show the abstract state for the outer one. In order to compare our results to the Octagon domain [21], we have used the Interproc analyzer [17,18], enabling the option for guided analysis (see [13]). This has required converting the sample programs from the R syntax to the syntax supported by Interproc. For the parallelotope and combined domains, we have used a change of basis matrix determined by orthogonal simple component analysis with an accuracy threshold of $\cos\frac{\pi}{4}$. The only exception is `cousot78`†, where we have used an accuracy of 0.98. For `bsort` and `bsearch` we have shown two different results: the first one is for a standard analyses, while in the second one we have instructed the tracer and analyzer not to consider the variables `k` and `tmp` respectively. The variable `k` is the key for the binary search, while `tmp` is just a temporary variable used to swap two elements of an array. Both are either compared with array elements or assigned to/from array elements, but our analyzer does not deal with arrays at all, nor does Interproc. Removing these variables by the analysis helps the PCA procedure. This suggests a possible improvement, not implemented yet, which is to automatically remove from the partial execution traces those variables which are assigned to/from array elements, or compared with them.

The results show that, in most cases, the domains of parallelotopes gives interesting properties, which cannot be inferred by the corresponding results of the box domain. In the `bsort`* case, the domain of parallelotopes does not yield anything interesting, but its combination with standard boxes does: the combined domains is able to prove (like Octagon) that all accesses to arrays are correct. In most of the cases, Octagon was able to obtain more precise abstract states than ours, but the theoretical complexity of its operations is greater. However, in the `bsearch`* case we were able to obtain a property which cannot be represented in Octagon, and cannot be inferred by the corresponding results. A practical comparison of speed is not possible at the moment, since our implementation in R is definitively slower than the APRON [18] library used in Interproc.

7 Related Work

The idea of parametrizing analyses for a single program, or a class of programs, has been pursued in a few papers. The analysis for digital filters proposed in [11] is an example of domains developed for a specific class of applications. The same holds for the domain of arithmetic-geometric progressions [12], used to determine restrictions on the value of variables, as a function of the program execution time.

In our paper, we extend this idea and propose parametric domains which may be specialized for a single program. The same approach can be found in the domain of symbolic intervals [24], which depend on a total ordering of variables in the program, and most importantly, in the domain of template polyhedra [25], that is, domains of fixed form polyhedra. For each program, the authors fix a matrix A and consider all the polyhedra of type $A\boldsymbol{x} \leq \boldsymbol{b}$. The choice of the matrix is what differentiates template polyhedra from other domains, where the matrix is fixed for all programs (such as intervals or Octagon) or varies freely (such as polyhedra.)

However, in all these papers, the choice of the parameters is performed using a syntactic inspection of the program. To the best of our knowledge, the present work is the first attempt of inferring parameters on the base of partial execution traces. Moreover, we try to be as conservative as possible, and reuse the operators of the original abstract domains, instead of devising completely new operators.

There are also parametrization strategies applicable to almost all numeric domains. For example, the accuracy of widening operators can be enhanced through the adoption of intermediate thresholds [3], from a simple syntactic analysis of the program (e.g., maximum size of arrays, constants declared in the program). Moreover, the complexity of relational analyses can be reduced by using *packing*, which partitions the set of all program variables into groups, performs relational analyses within the partitions and non-relational analyses between the partitions [3]. These strategies are orthogonal to our approach, and can be applied to our domains as well.

A different approach which exploits execution traces can be found in [16]. The authors collect (probabilistic) execution traces, in order to directly derive linear relationships between program variables, which hold with a given probability. On the contrary, in our approach we use the information gathered from partial execution traces as an input for a subsequent static analysis.

8 Conclusions and Future Work

We have presented a new technique for shaping numerical abstract domains to single programs, by applying a "best" linear transformation to the space of variable values. One of the main advantages of this technique is the ability to transform non-relational analysis into relational ones, by choosing the abstract domain which best fits for a single program. Moreover, this idea may be immediately applied to any numerical abstract domain which is not closed by linear transformations, such as octagons [21], bounded differences [19], simple congruences [14]. It suffices to give specialized algorithms for the assignment operation.

We have realized a prototypical analyzer and, as an application, we have fully developed our technique for the interval domain. The experimental evaluation seems promising, but also shows that there is still space for many improvements. We may choose specific program points where values are collected, such as a loop entry point, in order to better focus the statistical analysis and we may use techniques of code coverage, as in software testing, to improve the quality of execution traces. Moreover, we may partition the set of values we apply PCA to. One idea could be to partition the set of program variables into groups (variables used for array indexes, variables for temporary storage, etc...) which are expected not to be correlated, and perform PCA separately on each group (an idea similar to packing [3]). In addition, we may partition the program code itself (for example around loops), perform a different PCA on each partition, and change the abstract domain appropriately when crossing partitions. In the extreme, we could choose different parameters for each program point, like Sankaranarayanan et al. [25] do for template polyhedra.

The use of linear transformations also suggests to combine PCA with different approaches. We may infer the axes in the new coordinate system from both the semantics and the syntax of the program. The analysis could vastly benefit from the ability to express constraints occurring in the linear expressions of the program, especially in loop guards and array accesses. However, the syntactic approach alone is not recommended, since not all the interesting invariants appear as expressions in the source code. For example, the `cousot78` program does not contain the expressions `i+j`, `j-i` or `2*j-i`: nonetheless, the analysis was able to prove invariants on these constraints (see Figure 8). To overcome this limitation, we may use the probabilistic invariants found by the analysis in [16] instead of using the syntax of the program.

Finally, writing the implementation in R has been useful for rapid prototyping, but porting the code to a faster programming language, possibly within the framework of well known libraries such as `APRON` [18] or `PPL` [2], would make it available to a wider community, while improving performance.

References

1. Anaya-Izquierdo, K., Critchley, F., Vines, K.: Orthogonal simple component analysis. In: Technical Report 08/11, The Open University (2008), http://statistics.open.ac.uk/TechnicalReports/spca_final.pdf (last accessed 2010/03/26)
2. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1-2), 3–21 (2008)
3. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI 2003), San Diego, California, USA, June 7-14, pp. 196–207. ACM Press, New York (2003)
4. Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: Pottosin, I.V., Bjørner, D., Broy, M. (eds.) FMP&TA 1993. LNCS, vol. 735, pp. 128–141. Springer, Heidelberg (1993)
5. Chang, B.-Y.E., Rival, X.: Relational inductive shape analysis. In: Principles Of Programming Languages, POPL 2008 SIGPLAN Not., vol. 43(1), pp. 247–260. ACM, New York (2008)
6. Colóon, M.A., Sipma, H.B.: Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 67–81. Springer, Heidelberg (2001)
7. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, Paris, France, pp. 106–130. Dunod (1976)
8. Cousot, P., Cousot, R.: Abstract interpretation and applications to logic programs. The Journal of Logic Programming 13(2-3), 103–179 (1992)
9. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL 1978: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 84–97. ACM Press, New York (January 1978)
10. Dor, N., Rodeh, M., Sagiv, M.: Cleanness checking of string manipulations in C programs via integer analysis. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 194–212. Springer, Heidelberg (2001)
11. Feret, J.: Static analysis of digital filters. In: Schmidt [26], pp. 33–48
12. Feret, J.: The arithmetic-geometric progression abstract domain. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 42–58. Springer, Heidelberg (2005)
13. Gopan, D., Reps, T.: Guided static analysis. In: Nielson and Filé [22], pp. 349–365
14. Granger, P.: Static analysis of arithmetical congruences. International Journal of Computer Mathematics 32 (1989)
15. Gulavani, B.S., Gulwani, S.: A numerical abstract domain based on *expression abstraction* and *max operator* with application in timing analysis. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 370–384. Springer, Heidelberg (2008)
16. Gulwani, S., Necula, G.C.: Precise interprocedural analysis using random interpretation. In: Principles Of Programming Languages, POPL 2005. SIGPLAN Not., vol. 40(1), pp. 324–337. ACM, New York (2005)

17. Jeannet, B.: Interproc Analyzer for Recursive Programs with Numerical Variables. INRIA. Software and documentation are available at the following, `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi` (last accessed: 2010-06-11)
18. 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)
19. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, pp. 155–172. Springer, Heidelberg (2001)
20. Minè, A.: Relational abstract domains for the detection of floating-point run-time errors. In: Schmidt [26], pp. 3–17
21. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation 19(1), 31–100 (2006)
22. Nielson, H.R., Filé, G. (eds.): SAS 2007. LNCS, vol. 4634, pp. 249–264. Springer, Heidelberg (2007)
23. R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2009)
24. Sankaranarayanan, S., Ivančić, F., Gupta, A.: Program analysis using symbolic ranges. In: Nielson and Filé [22], pp. 366–383
25. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 25–41. Springer, Heidelberg (2005)
26. Schmidt, D. (ed.): Programming Languages and Systems. ESOP 2004. LNCS, vol. 2986. Springer, Heidelberg (2004)