Static Analysis by Policy Iteration on Relational Domains

Stephane Gaubert[1], Eric Goubault[2], Ankur Taly[3], Sarah Zennou[2]

[1] INRIA Rocquencourt, stephane.gaubert@inria.fr
[2] CEA-LIST, MeASI, {eric.goubault, sarah.zennou}@cea.fr
[3] IIT Bombay, ankurtaly@iitb.ac.in

Abstract. We give a new practical algorithm to compute, in finite time, a fixpoint (and often the least fixpoint) of a system of equations in the abstract numerical domains of zones and templates used for static analysis of programs by abstract interpretation. This paper extends previous work on the non-relational domain of interval to relational domains. The algorithm is based on policy iteration techniques– rather than Kleene iterations as used classically in static analysis– and generates from the system of equations a finite set of simpler systems that we call *policies*. This set of policies satisfies a *selection property* which ensures that the minimal fixpoint of the original system of equations is the minimum of the fixpoints of the policies. Computing a fix point of a policy is done by linear programming. It is shown, through experiments made on a prototype analyzer, compared in particular to analyzers such as LPInv or the Octagon Analyzer, to be in general more precise and faster than the usual Kleene iteration combined with widening and narrowing techniques.

1 Introduction

One of the crucial steps of static analysis by abstract interpretation [CC76] is the precise and efficient solving of the system of equations representing the abstraction of the program properties we want to find out. This is generally done by iteration solvers, based on Kleene's theorem, improved using extrapolation methods such as widening and narrowing operators [CC91]. These methods are quite efficient in practice, but are not always very precise and are difficult to tune as the quality and efficiency might depend a lot on the code under analysis.

In [CGG$^+$05], some of the authors proposed a new method for solving these abstract semantic equations, which is based on policy iteration. The idea of policy iteration was introduced by Howard in the setting of Markov decision processes (one player stochastic games), see [How60]. It reduces a fixed point problem to a sequence of simpler fixed points problems, which are obtained by fixing policies (strategies of one player). This method was extended to a subclass of (zero-sum) two player stochastic games by Hoffman and Karp [HK66]. However, static analysis problems lead to more general fixed point equations, which may be degenerate, as in the case of deterministic games [GG98]. The algorithm introduced in [CGG$^+$05] works in a general setting, it always terminates with a fixed point, the minimality of which can can be guaranteed for the important subclass of sup-norm nonexpansive maps, see Theorem 3 and Remark 5 of [CGG$^+$05]. The experiments showed that in general, policy iteration on intervals was faster and more precise than Kleene iteration plus (standard) widenings and narrowings. In this paper, we extend the framework of [CGG$^+$05] to deal with policy iteration in relational domains, such as zones [Min01a], octagons [Min01b] and TCMs (Template Constraint Matrices [SSM05b]). We describe a general finite time algorithm that computes a fixpoint of functionals in such domains, and often a least fixpoint. We did not treat polyhedral analyses [CH78], as general polyhedra are in general not scalable.

There are two key novelties by comparison with [CGG$^+$05]. The first one is that the computation of closures (canonical representatives of a set of constraints), which was trivial in the case of intervals, must now be expressed in terms of policies, in such a way that the selection property on which policy iteration relies is satisfied. We solve this problem by means of linear programming duality: we show in particular that every policy arising in a closure operation can be identified to an extreme point of a polyhedron. Secondly, for each policy, we have to solve a (simpler) set of equations, for which we use linear programming [Chv83], and not Kleene iteration as in [CGG$^+$05]. We have developped a prototype; first benchmarks show that we gain in efficiency and in general in precision with respect to Kleene iteration solvers, in zones (comparison was made possible thanks to a prototype on the more refined domain of octagons of A. Miné [Min05]), and in simple TCMs (using LPInv [SSM05a]). This is conjecturally true for general TCMs too, a claim which is not yet substantiated by experimental results as we have not yet implemented general TCMs in our analyzer.

The paper is organized as follows: In section 2, we recap some of the basics of abstract interpretation and recall the main operations on the zone and TCMs abstract domains. We then introduce our policy iteration technique for both domains in section 3. Algorithmic and implementation issues are treated in section 4. We end up in section 5 by showing that policy iteration exhibits very good results in practice.

2 Basics

2.1 Abstract Interpretation by Static Analysis

Invariants, that can be obtained by abstract interpretation based static analysis, provide sound overapproximations of the set of values that program variables can take at each control point of the program. They are obtained by computing the *least fixpoint* of a system of abstract equations derived from the program to analyze. The correctness of the approximation is in general[1] guaranteed by the theory of *Galois Connections* between the *concrete domain* (the set of variable

[1] See [CC92] for more general frameworks.

values of the program) and the *abstract domain* (more easily tractable representatives of possible sets of values that the program can take).

For a complete lattice $(\mathcal{L}_e, \sqsubseteq_e)$, we write $\bot_e$ for its lowest element, $\top_e$ for its greatest element, $\sqcup_e$ and $\sqcap_e$ for the meet and join operations, respectively. We say that a self-map f of $(\mathcal{L}_e, \sqsubseteq_e)$ is *monotone* if $x \sqsubseteq_e y \Rightarrow f(x) \sqsubseteq_e f(y)$. Existence of fixpoints is ensured by the *Knaster-Tarski theorem* which states that every monotone self-map on a complete lattice has a fixpoint and indeed a least fixpoint. The least fixpoint of a monotone self-map f on a complete lattice will be denoted f^-.

Let $(\mathcal{L}_c, \sqsubseteq_c)$ be the complete lattice representing the concrete domain and $(\mathcal{L}_a, \sqsubseteq_a)$ the one representing the abstract domain. In most cases, the link between the two domains is expressed by a *Galois connection* [CC77], that is a pair (α, γ) of maps with the following properties : $\alpha : \mathcal{L}_c \to \mathcal{L}_a$ and $\gamma : \mathcal{L}_a \to \mathcal{L}_c$ are both monotone, and $\alpha(v_c) \sqsubseteq_a v_a$ iff $v_c \sqsubseteq_c \gamma(v_a)$. The map α is the *abstraction function*, γ, the *concretization function*. These properties guarantee that α gives the best upper approximation of a concrete property, in the abstract domain.

Figure 1 gives a C program (`test2`, left part) together with the semantic equations (right part) for both zones and TCMs domain where M_i is the abstract local invariant to be found at program line i.

The function *context_initialization* creates an initial local invariant: typically by initializing the known variables to the top element of the abstract lattice of properties, or to some known value. The function $\mathit{Assignment}(\mathit{var} \leftarrow \mathit{val})(M_j)$ is the (forward) abstract transformer which computes the new local invariant after assignment of value *val* to variable *var* from local invariant M_i. Finally, $(\cdot)^*$ is the normalization, or closure, of an abstract value, see section 2.2.

Fig. 1. A program (left part) and its representation by equations

```
0    i = 150;
1    j = 175;
2    while (j >= 100){
3      i++;
4      if (j<= i){
5          i = i - 1;
6          j = j - 2;
7      }
8    }
9
```

$$
\begin{aligned}
M_0 &= \mathit{context\_initialization}\\
M_2 &= (\mathit{Assignment}\ (i \leftarrow 150,\ \ j \leftarrow 175)(M_0))^*\\
M_3 &= ((M_2 \sqcup M_8) \sqcap (j \geq 100))^*\\
M_4 &= (\mathit{Assignment}\ (i \leftarrow i+1)(M_3))^*\\
M_5 &= (M_4 \sqcap (j \leq i))^*\\
\\
M_7 &= (\mathit{Assignment}\ (i \leftarrow i-1,\ \ j \leftarrow j-2)(M_5))^*\\
M_8 &= ((M_4 \sqcap (j > i))^* \sqcup M_7\\
M_9 &= ((M_2 \sqcup M_8) \sqcap (j < 100))^*
\end{aligned}
$$

Abstract versions f_a of the concrete primitives f_c such as *assignment*, *context_initialization* etc. are defined as $f_a(v) = \alpha(f_c(\gamma(v)))$, but in general we use a computable approximation $f_a(v)$ such that $\alpha(f_c(\gamma(v))) \sqsubseteq_a f_a(v)$. In particular invariants are preserved: if x is a (resp. the least) fixpoint of f_a then $\gamma(x)$ is a (resp. the least) (post) fixpoint of f_c.

Kleene iteration It is well-known since Kleene that the least fixpoint of a continuous function on a complete lattice is $\bigsqcup_{n \in \mathbb{N}} f^n(\bot)$. This result gives an immediate algorithm for computing the fixpoint : starting from the value $x_0 = \bot$, the k-th iteration computes $x_k = x_{k-1} \sqcup f(x_{k-1})$. The algorithm finishes when $x_k = x_{k-1}$. For (only) monotonic functions, one may need more general ordinal iterations. In practice though, as these iterations might not stabilize in finite time, it is customary to use acceleration techniques, such as widening and narrowing operators [CC91]: for instance, on intervals, we can use the following widening ∇ operator:

$$[a,b]\nabla[c,d] = [e,f] \text{ with } e = \begin{cases} a & \text{if } a \leq c \\ -\infty & \text{otherwise} \end{cases} \quad \text{and } f = \begin{cases} b & \text{if } d \leq b \\ \infty & \text{otherwise,} \end{cases}$$

These ensure finite time convergence to a fixpoint, which is not necessarily the least fixpoint: widening returns a post fixpoint while narrowing computes a fixpoint from a post fixpoint. On intervals, and for our running example 1, if these widening and narrowing operators are applied after 10 iterations (as was done in [CGG$^+$05] for matter of comparisons), we get the following iteration sequence, where we only indicate what happens at control points 3, 7, 8 and 9. We write (i_l^k, j_l^k) for the abstract values at line l and iteration k (describing the concrete values of variables i and j). Widening takes place between iteration 9 and 10 and narrowing between 11 and 12.

$$\begin{array}{ll}
(i_3^1, j_3^1) & = ([150,150],[175,175]) \\
(i_7^1, j_7^1) & = \bot \\
(i_8^1, j_8^1) & = ([151,151],[175,175]) \\
(i_9^1, j_9^1) & = \bot \\
 & \ldots \\
(i_3^9, j_3^9) & = ([150,158],[175,175]) \\
(i_7^9, j_7^9) & = \bot \\
(i_8^9, j_8^9) & = ([151,159],[175,175]) \\
(i_9^9, j_9^9) & = \bot \\
\text{(widening)} & \\
(i_3^{10}, j_3^{10}) & = ([150,+\infty[,[175,175])
\end{array}$$

$$\begin{array}{ll}
(i_7^{10}, j_7^{10}) & = ([149,+\infty[,[173,173]) \\
(i_8^{10}, j_8^{10}) & = ([151,+\infty[,[173,175]) \\
(i_9^{10}, j_9^{10}) & = \bot \\
(i_3^{11}, j_3^{11}) & = ([150,+\infty[,]-\infty,175]) \\
(i_7^{11}, j_7^{11}) & = ([150,+\infty[,]-\infty,149]) \\
(i_8^{11}, j_8^{11}) & = ([150,+\infty[,]-\infty,175]) \\
(i_9^{11}, j_9^{11}) & = ([150,+\infty[,]-\infty,99]) \\
\text{(narrowing)} & \\
(i_3^{11}, j_3^{11}) & = ([150,+\infty[,[100,175]) \\
(i_7^{11}, j_7^{11}) & = ([150,+\infty[,[98,149]) \\
(i_8^{11}, j_8^{11}) & = ([150,+\infty[,[98,175]) \\
(i_9^{11}, j_9^{11}) & = ([150,+\infty[,[98,99])
\end{array}$$

2.2 Two Existing Relational Abstract Domains

In this section we present some basics on the *zone* and *TCM* domains. In particular the loss of precision due to widening use is discussed. For an exhaustive treatment see respectively the references [Min01a] and [SSM05b,SCSM06]. These domains enable one to express linear relations between variables, all subpolyhedral ([CH78]): in zones, linear relations involve only differences between variables, whereas in TCM, they involve finitely many linear combinations of the variables. Unlike in the case of polyhedral domains, these linear combinations are given *a priori*.

In the sequel we consider a finite set $V = \{v_1, \ldots, v_n\}$ of real valued variables. Let $\mathbb{I} = \mathbb{R} \cup \{-\infty, \infty\}$ be the extension of the set $\mathbb{R}$ of real values with two special values $-\infty$ (will be used to model a linear relation without solution in $\mathbb{R}$) and ∞ (linear relation will be satified by any value). The operators $\leq, \geq, min, max$ are extended as usual to deal with these values.

Zone Abstract Domain To represent constraints like $v \leq c$ we extend V by a virtual fresh variable v_0 whose value is always zero so that $v \leq c$ becomes equivalent to $v_i - v_0 \leq c$. Let us denote $V_0 = V \cup \{v_0\}$. A *zone* is then a vector $c = (c_{0,0}, c_{0,1}, c_{0,2}, \ldots c_{n,n})$ where $c_{i,j} \in \mathbb{I}$ stands for the constraint $v_i - v_j \leq c_{i,j}$ for $v_i, v_j \in V_0$. The *concretization* of c is the set of real values of variables in V whose pairwise differences $v_i - v_j$ are bounded by the coordinates $c_{i,j}$ of c. Formally, $\gamma(c) = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i - x_j \leq c_{i,j},\ -c_{0,i} \leq x_i \leq c_{i,0}\}$.

TCM Abstract Domain A *Template Constraint Matrix* T (TCM) is an ordered set $T = \{e_1, \ldots, e_m\}$ of linear relations $e_i(x) = a_{i,1}x_1 + \ldots + a_{i,n}x_n$ where $(a_{i,1}, \ldots, a_{i,n})$ and $x = (x_1, \ldots, x_n)$ are real valued vectors of length $n = |V|$. In practice this TCM T can represented by a matrix M of dimension $m \times n$ and such that its entry (i, j) is $a_{i,j}$. Hence the ith line of this matrix is the vector $(a_{i,1}, \ldots, a_{i,n})$. For the sake of simplicy, we sometimes identify a TCM T with its representation by matrix in the sequel where T_i will denote its ith row.

An element of a TCM $T = \{e_1, \ldots, e_m\}$ is a vector $c = (c_1, \ldots, c_m)$ with $c_i \in \mathbb{I}$ and its concretization $\gamma(c)$ is the set of real values $x = (x_1, \ldots, x_n)$ that satisfy every $e_i(x) + c_i \geq 0$ with $c_i \neq +\infty$. Thus, $\gamma(c) = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid e_i(x) + c_i \geq 0 \wedge c_i \neq +\infty\}$. In particular, $\gamma(c) = \emptyset$ if $c_i = -\infty$ for some i. A linear assertion (a conjunction of linear relations) of the form $e_i(x) + c_i \geq 0$ will be denoted $e(x) + c \geq 0$ with $e = (e_1, \ldots, e_m)$ and $c = (c_1, \ldots, e_m)$.

A precise fixpoint detection in static analysis can be made by use of one TCM per control point in the program to analyse. As it complicates the presentation but does not change our theoretical results, we present operations in the case of one TCM. In the case of several TCMs, operation results or operands have to be expressed in the same TCM. This operation is called *projection.*

Linear Programming The emptyness of the concretization can be checked using *Linear Programming* (see [Chv83] for an systematic treatment).

Let $e(x) + c \geq 0$ be a linear assertion. A linear programming (LP) problem consists in minimizing a linear relation $f(x)$, called the *objective function*, subject to the constraint of $e(x) + c \geq 0$. The concretization emptiness problem corresponds to the case where $f(x)$ is the constant map 0. A LP problem may have three answers: the problem is *infeasible*, or there is *one optimal solution*, or the problem is *unbounded* ($f(x)$ can be decreased down to $-\infty$). A linear programming problem can be solved either by the simplex algorithm (whose theoretical complexity is exponential, but which is efficient in practice) or by modern interior point methods, which are polynomial time and practically efficient.

Order and extrema To get a lattice structure, the zone and TCM domains are extended with a supremum $\top = (\top, \ldots, \top)$ (whose concretization is $\mathbb{I}^n$ itself) and an infimum $\perp$ which is any vector with at least one coordinate whose value is $-\infty$ (its concretization is empty). If $\gamma(c)$ of a zone or TCM c is not empty, c is said to be *consistent* otherwise it is *inconsistent.* The *order* $\sqsubseteq$ is the vector order: $c_1 \sqsubseteq c_2$ iff $c_1(i) \leq c_2(i)$ for every $i = 1, .., |c_1| = |c_2|$. We have property that $c_1 \sqsubseteq c_2 \implies \gamma_1(c_1) \subseteq \gamma_2(c_2)$ but the converse is not true. This problem is addressed by the *closure* operation.

Closure Several zones or TCMs vectors may have the same concrete domain. As a canonical representative, the *closed* one is chosen. The *closure* c^* of a *consistent* zone or TCM vector c is the $\sqsubseteq$-minimal zone or TCM vector such that $\gamma(c^*) = \gamma(c)$.

Closure on zones If $c = (c_{0,1}, \ldots, c_{n,n})$ is a consistent zone then $c^* = (c^*_{0,0}, \ldots c^*_{n,n})$ is such that $c^*_{i,j} = min\{c_{ii_1} + \ldots + c_{i_{k-1}j}\}$ for some k. A zone c is consistent iff every diagonal coordinate of c^* is non-negative. It follows that the consistency and closure problems reduce to an all pairs shortest path problem.

Closure on TCMs Let $c = (c_1, \ldots, c_m)$ be a consistent vector on the TCM T seen as a matrix of dimension $m \times n$. Let us denote $c_{|\mathbb{R}}$ the subvector of c in which ∞ coordinates are deleted. Let $T_{|\mathbb{R}}$ be the corresponding submatrix of lines T_i of T such that $c_i \neq \infty$. Closure c^* of c is the vector $(c^*_1, \ldots, c^*_m)$ such that c^*_i is the solution of the LP problem "*minimize* $c_{|\mathbb{R}}\lambda$ *subject to* $T_{|\mathbb{R}}\lambda = T_i$, $\lambda \geq 0$". It has been shown in [SSM05b] that as c is consistent no LP problem may be unbounded[2]. Hence as $\gamma(c^*) = \gamma(c) \neq \emptyset$ we conclude all these m LP problems do have an optimal solution.

Meet and Join The $\sqcup$ operation is a pointwise maximum between the vector coordinates: $c_1 \sqcup c_2 = (max\{c_1(1), c_2(1)\}, \ldots, max\{c_1(k), c_2(k)\})$. This operation is an overapproximation, but preserves closure. In the context of TCM this definition corresponds to the so called *weak join* [SCSM06]. Our work extends to *strong* and *restricted* joins presented also in this article as they are combination of the closure operation with weak join, for which we show that policy iteration is compatible. The $\sqcap$ operation is a pointwise mimimum operation between the vector coordinates: $c_1 \sqcap c_2 = (min\{c_1(1), c_2(1)\}, \ldots, min\{c_1(k)), c_2(k)\})$. This operation is exact but *does not* preserve closure.

Widening on zones $c_1 \nabla c_2 = c$ with $c_i = c_1(i)$ if $c_2(i) \leq c_1(i)$ otherwise $c_i = \infty$. An important remark about the widening is that its use forbids to close the left operand otherwise termination is not guaranteed. The consequence when computing a fixpoint with a Kleene iteration is that after a widening, closure is forbidden so that for a pair $v_i - v_j$ whose bound becomes $+\infty$, this difference will

[2] if it were not the case, this would contradict $\gamma(c) = \gamma(c^*)$ as we would have $\gamma(c^*) = \emptyset$ and $\gamma(c) \neq \emptyset$ by hypothesis.

remain unbounded until the end of the Kleene iteration. This situation occurs on the left part of the computation table below, where the triple $(i_9^3, j_9^3, i_9^3 - j_9^3)$ stands for the zone $\{150 \leq x_1 - x_0 \leq 158 \wedge 175 \leq x_2 - x_0 \leq 175 \wedge -25 \leq x_1 - x_2 \leq -17\} \cup \{x - x \leq 0\}$. At iteration [10] a widening iteration is computed. It can be seen that the result on every constraint involving the upper bound of i from control point [3] remains unbounded (this is a special case where zones computing a widening gives a closed zone but it is not true in general as shown in Mine's thesis [Min04]). In the worst case, every pair is concerned by the widening so that the constraint set becomes a set of intervals. This drawback does not exist with the policy iteration as we do not use the widening operator.

$$\begin{array}{rcl}
(i_3^1, j_3^1, i_3^1 - j_3^1) & = & ([150,150], [175,175], \\
 & & [-25,25]) \\
(i_7^1, j_7^1, i_7^1 - j_7^1) & = & \perp \\
(i_8^1, j_8^1, i_8^1 - j_8^1) & = & ([151,151], [175,175], \\
 & & [-24,24]) \\
(i_9^1, j_9^1, i_9^1 - j_9^1) & = & \perp \\
 & \ldots & \\
(i_3^9, j_3^9, i_3^9 - j_3^9) & = & ([150,158], [175,175], \\
 & & [-25,-17]) \\
(i_7^9, j_7^9, i_7^9 - j_7^9) & = & \perp \\
(i_8^9, j_8^9, i_8^9 - j_8^9) & = & ([151,159], [175,175], \\
 & & [-24,-16]) \\
(i_9^9, j_9^9, i_9^9 - j_9^9) & = & \perp \\
\text{(widening)} & & \\
(i_3^{10}, j_3^{10}, i_3^{10} - j_3^{10}) & = & ([150,\infty], [175,175], \\
 & & [-\infty,-25] \\
(i_4^{10}, j_4^{10}, i_4^{10} - j_4^{10}) & = & ([149,\infty], [173,173], \\
 & & [-\infty,-24]
\end{array}
\qquad
\begin{array}{rcl}
 & \ldots & \\
c_3^1 & = & (-150, 150, -175, 175, \\
 & & -25, 25) \\
c_7^1 & = & \perp \\
c_8^1 & = & (-151, 151, -175, 175, \\
 & & -24, 24) \\
c_9^1 & = & \perp \\
 & \ldots & \\
c_3^9 & = & (-150, 158, -175, 175, \\
 & & -17, 25) \\
c_7^9 & = & \perp \\
c_8^9 & = & (-151, 158, -175, 175, \\
 & & -16, 24) \\
c_9^9 & = & \perp \\
\text{(widening)} & & \\
c_3^{10} & = & (-150, -175, 175, 25) \\
c_4^{10} & = & (-149, -173, 175, 24) \\
 & \ldots &
\end{array}$$

Widening on TCMs It corresponds to the computation of a vector c', $|c'| \leq |c_1|$ from c_1's coordinates such that $\gamma(c') \subseteq \gamma(c_2)$. There are two cases to consider: either c_1 or c_2 is inconsistent and $c_1 \nabla c_2$ is simply $c_1 \sqcup c_2$. Otherwise let us denote b_i to be the solution of the LP problem "*minimize* $e_i(x) + c_1(i)$ *subject to* $e(x) + c_2 \geq 0$". If b_i is positive then $c(i) = b_i$ otherwise the linear expression e_i *is deleted from* T. Deleting a linear expression avoids the problem described on zones which is that after a widening, closures are no more allowed in a Kleene iteration. Nevertheless deleting a linear expression in a TCM impoverishes the expressiness of the abstract domain. The major drawback of the widening operator when used with Kleene iteration is not solved. For instance on the program of Figure 1, Kleene iterations with TCM $T = \{x_1, -x_1, x_2, -x_2, x_1 - x_2, x_2 - x_1\}$ (which models zones) are shown on the right part of the computation table above. Results are identical to those in the case of zones, when widening occurs: from iteration [10] on, the TCM reduces to $T' = T \backslash \{-x_1, x_2 - x_1\}$ so that further vectors have only four coordinates.

3 Policy Iteration for Relational Abstract Domains

The aim of policy iteration is to compute a fixpoint of some monotonic function F which is a combination of "simpler" monotonic maps g, for which we can hope for fast algorithms to compute their least fixpoints. For complete lattices such as the interval domain [CGG$^+$05], the maps g do not contain the intersection operator. F is the intersection of a certain number of such g maps, and the goal of policy iteration techniques is to ensure (and find) the simpler g which has as least fixpoint, a fixpoint of F (not the least one in general). We will prove that if we have a "selection property", Definition 1, then we can compute the least fixpoint of F from the least fixpoints of the maps g, theorem 2. The policy iteration algorithm will traverse in a clever manner the space of these g maps to find efficiently a fixpoint of F.

To present policy iteration in a uniform manner for zone and TCM we use notion of *closed domains*. A closed domain $(\overline{\mathcal{L}}, \overline{\bot}, \overline{\top}, \overline{\sqsubseteq}, \overline{\sqcup}, \overline{\sqcap})$ is such that $\mathcal{L}$ is an abstract domain and $\overline{\mathcal{L}}$ contains only closed elements of $\mathcal{L}$. As closure is only defined for consistent elements, we introduce the bottom element $\overline{\bot}$ representing all inconsistent elements to equip $\overline{\mathcal{L}}$ with a lattice structure. Top element is $\overline{\top}$. The order $\overline{\sqsubseteq}$ is $\overline{\bot}\overline{\sqsubseteq}\overline{c}\overline{\sqsubseteq}\overline{\top}$ for every c and for $c_1, c_2 \neq \overline{\bot}$ $c_1 \overline{\sqsubseteq} c_2$ iff $c_1 \sqsubseteq c_2$. Operators are as follows:

$x \overline{\sqcap} y \widehat{=} z$ with $z = \overline{\bot}$ if $x = \overline{\bot}$ or $y = \overline{\bot}$; $z = (x \sqcap y)^*$ otherwise.

$x \overline{\sqcup} y \widehat{=} z$ with $z = \overline{\bot}$ if $y = \overline{\bot}$; $z = y$ if $x = \overline{\bot}$; $z = x \sqcup y$ otherwise.

Note that both zone and TCM closure closure satisfy that $x^* = x^{**} \sqsubseteq x$ and they are monotonic.

3.1 Selection Property

Remember (see [CGG$^+$05]), that in intervals, policies are of four types l, r, m and m^{op} defined below. When $I = [-a, b]$ and $J = [-c, d]$, $l(I, J) = I$ (l is for "left"), $r(I, J) = J$ (r for "right"), $m(I, J) = [-a, d]$ and $m^{\mathrm{op}}(I, J) = [-c, b]$ (m is for "merge"). The maps g are derived from F by replacing the operator $\cap$ (intersection) by any of these four operators.

For instance, $F([a, b]) = ([1, 2] \cap [a, b]) \cup ([3, 4] \cap [a, b])$, where $x = [a, b]$ is an interval of real values, has eight policies :

$llll(x) = [1, b] \cup [3, 4]$, $lrll(x) = [1, b] \cup [3, 4]$, $rlll(x) = [2, b] \cup [3, 4]$, $rrll(x) = [a, b] \cup [3, 4]$, $lllr(x) = [1, 2] \cup [3, b]$, $llrl(x) = [1, 2] \cup [a, 4]$, $llrr(x) = [1, 2] \cup [a, b]$, $rrrr(x) = [a, b]$.

We then say that F satisfies the *selection property* since F is such that for all interval x $F(x) = min\{g(x) \mid g \in \mathcal{G}\}$. We extend this definition to deal with relational domains:

Definition 1. Let $\mathcal{G}$ denote a finite or infinite set of monotone self maps on the complete lattice $\overline{\mathcal{L}}$, we say that a monotone self map F satisfies the *selection property* if the two following properties are satisfied:

(1) $F = F^* = (inf\{g \mid g \in \mathcal{G}\})$

(2) for all $x \in \overline{\mathcal{L}}$, there exists $h \in \mathcal{G}$ (a policy) such that $F(x) = h(x)$.

In condition (1), $F = F^*$ holds as $g(x)$ are closed and we have property that $x^* = x^{**}$. Hence the least fixpoint of F is a least fixpoint of some policy:

Theorem 2. *Let F be a monotone self map on a complete lattice $\overline{\mathcal{L}}$, satisfying the selection property for a set of monotone self maps $\mathcal{G}$. Then the least fixpoint of F is reached by the least fixpoint of some policy:*

$$F^- = inf\{(g^-)^* \mid g \in \mathcal{G}\}$$

This theorem proves that algorithm 1 computes a fixpoint of an application F that satisfies the selection property. Starting from an initial policy provided by a function *initial_policy* this algorithm computes iteratively the least fixpoint x of some policy g_k (done at iteration k). If x is a fixpoint of F then algorithm terminates otherwise a new policy g_{k+1} is selected for iteration $k+1$ in such a way that $g^*_{k+1} = x$ (this is always possible as F has the selection property).

Algorithm 1 Policy iteration algorithm

```
$k \leftarrow 1$ ; $g_1 \leftarrow$ initial_policy($\mathcal{G}$)
while true do
  $x_k \leftarrow (g_k^-)^*$
  if $x_k = F(x_k)$ then
    return $x_k$
  else
    find $g$ such that $F(x_k) = g^*(x_k)$
    $k \leftarrow k+1$ ; $g_k \leftarrow g$
  end if
end while
```

Algorithm 1 may not return the least fixpoint. However, for some classes of monotone maps, including sup-norm non-expansive maps, an extension of Algorithm 1 does provide the least fixpoint, see Theorem 3 and Remark 5 in [CGG$^+$05]: when a fixpoint for F is detected at iteration n it is possible to scan all the remaining policies g that belong to $\mathcal{G}\backslash\{g_1, \ldots, g_n\}$ and to compute their least fixpoint and finally returning the least one between all of them.

The following theorem states that algorithm 1 is correct and computes a decreasing chain of post fixpoints of an application satisfying the selection porperty:

Theorem 3. *Let F be a monotone self map on the complete lattice $\overline{\mathcal{L}}$ satisfying the selection property for a set of maps $\mathcal{G}$. We have the two following properties:*

(i) If algorithm 1 finishes then the returned value is a fixpoint of F

(ii) The sequence of maps $g_k \in \mathcal{G}$ generated by the algorithm 1 is a strictly decreasing chain, that is

$$(g^-_{k+1})^* \sqsubseteq (g^-_k)^*$$

If $\mathcal{G}$ is finite and has n policies then algorithm 1 finishes within at most n iterations:

Corollary 4. *If the set $\mathcal{G}$ is finite then algorithm 1 returns a fixpoint of F and the number of iterations of algorithm 1 is bounded by the height of $\{g^-|g \in \mathcal{G}\}$ which in turn is bounded by the cardinality of $\mathcal{G}$.*

3.2 Operations with policy

We show that the meet and closure operations of any map can be expressed as an infimum of simpler maps.

Meet policies The meet $c = c_1 \sqcap c_2$ of two vectors (zones or TCM vectors) c_1 and c_2 of length k is obtained by taking the pointwise minimum between each pair of coordinates. That is the ith coordinate of the result comes either from the left or right operand coordinate.

We use this remark to build a family of meet policies in the following way: Let $L \sqsubseteq \{1, \ldots, k\}$ be a set of coordinates whose corresponding policy will be denoted $\sqcap_L$. The set L contains every index i for which we take the ith coordinate of the left operand: if $i \in L$ then $c(i) = c_1(i)$ otherwise $c(i) = c_2(i)$.

We have trivially $c_1 \sqcap c_2 = \bigcap_{L \subseteq \{1,..,k\}} c_1 \sqcap_L c_2$ that is $\sqcap$ satisfies the selection property for the set of $\sqcap_L$ policies.

Closure policies Remember that the closure of a consistent zone or TCM c is the minimal c^* such that $\gamma(c) \sqsubseteq \gamma(c^*)$.

Closure policies for zones For a consistent zone c let $c^p_{ij} = c_{i,i_1} + c_{i_2,i_3} + \ldots + c_{i_{k-1},i_k} + c_{i_k,j}$ with $c_{i_p i_{p+1}}$ a coordinate of c and $p = i, i_1, ..., i_k, j$ and a sequence of variable indices called a *path from i to j*.

By definition, $c^*_{i,j}$ is the minimal c^p_{ij} amongst all pathes p from i to j. As mentioned in Section 2.2 the minimal c^p_{ij} can be obtained for a path length $|p|$ less than the number of variables. Hence $c^*_{i,j} = \bigcap_{p,|p| \leq n} c^p_{i,j}$ which satisfies the selection property.

Example Take the example of Figure 1. Our policy analyzer finds (in one policy iteration) the loop invariant at control point [9] left below, wheras a a typical static analyzer using Kleene iteration finds a less precise invariant (right below, using A. Mine's octagon analyzer).

$$\left\{\begin{array}{c} 150 \leq i \leq 174 \\ 98 \leq j \leq 99 \\ -76 \leq j - i \leq -51 \end{array}\right. \qquad \left\{\begin{array}{c} 150 \leq i \\ 98 \leq j \leq 99 \\ j - i \leq -51 \\ 248 \leq j + i \end{array}\right.$$

Consider now the program shown left below. The fixpoint found by our method (after two unfoldings) is given right below. This is incomparable to the fixpoint found in octogons (below), but its concretisation is smaller in width. This example needs two policies to converge.

```
0   void main() {
1    i = 1; j = 10;
2    while (i <= j){
3     i = i + 2;
4     j = j - 1; }
5   }
```

Policy iteration:

$5 \leq i \leq 10,\ 4 \leq j \leq 8,\ -3 \leq j{-}i \leq -1$

Kleene on octagons:

$6 \leq i \leq 12,\ \dfrac{9}{2} \leq j \leq 10,\ -3 \leq j{-}i \leq -1$

At [5] the initial policy chosen (see Section 4.2) gives the invariant of the right part below. The value of the functional on the invariant found using this initial policy is (and this is the only control point at which we have not reached the least fixpoint) is on the left below:

$$\begin{cases} 5 \leq i \leq 11,\ 2 \leq j \leq 8 \\ \quad -3 \leq j - i \leq -1 \end{cases} \qquad\qquad \begin{cases} 5 \leq i \leq 10,\ 4 \leq j \leq 8 \\ \quad -3 \leq j - i \leq -1 \end{cases}$$

It is easy to see that the entry describing the maximum of i has to be changed to a length two closure, and the minimum of the entry describing the minimum of j has to be changed to a length two closure, the rest of the equations being unchanged.

Closure policies for TCM Let $c = (c_1, \ldots, c_m)$ be a consistent TCM vector on T seen as a matrix of dimension $m \times n$. Closure c^* of c is the vector $(c_1^*, \ldots, c_m^*)$ such that c_i^* is the solution of the LP problem "*minimize* $c_{|\mathbb{R}}\lambda$ *subject to* $T^*_{|\mathbb{R}} = T_i$, $\lambda \geq 0$". As already said every LP problem has an optimal solution. It has been shown that this optimal solution is reached for a vertex of the polyhedron P_i $T^*_{|\mathbb{R}} = T_i$, $\lambda \geq 0$ whose number is finite. Hence we have $c^*(i) = inf\{x \in \mathbb{R} \mid x \ is \ a \ vertex \ of \ P_i\}$. A policy map is then any map that returns any vector whose ith coordinate is a vertex of polyhedron P_i so that $c^* = inf\{(\lambda c \mid \lambda \ is \ a \ vertex \ of \ P_i\}$.

4 Algorithmic issues

Algorithm 1 gives a general method to compute a fixpoint of some map F that satifies the selection property (Definition 1). In this section we give a method (based on linear programming rather than Kleene iteration as in [CGG$^+$05]) to compute least fixpoints of the policies. We give also some heuristics for the choice of intial policies on zone and TCM domain.

4.1 Least Fixpoint Computation, for a given policy

Each iteration k of algorithm 1 needs to compute the least fixpoint of a policy g_k, where every entry of g_k is a finite supremum of affine maps. By Tarski's theorem, this least fixpoint is the minimal vector x such that $g_k(x) \leq x$. If this least fixed point is finite, it can be found by solving a linear program: we minimize the linear form $\sum_{1\leq i\leq p} x_i$ over the constraints $g_k(x) \leq x$, where $x_1, \ldots, x_p$ are the variables composing the vector x. If the value of the latter linear program is unbounded, some entries of the least fixpoint x of g_k must be equal to $-\infty$. Note that the simplex method provides at least one of these entries, because, when a linear program is unbounded, the simplex method returns a half-line included in the feasible set, on which the objective function is still unbounded. Hence, the least fixed point can be found by solving a sequence of linear programs. The method is described below. It takes into account the "block upper triangular form" of the system $g_k(x) \leq x$ to reduce the execution time. In fact, the size of "blocks" turns out to be small, in practice, so the linear programs that we call only involve small subsets of variables.

The algorithm is then as follows:

- Form a graph G with nodes $1, 2, \ldots, p$ corresponding to each of the linear constraints $C_1, C_2, \ldots C_p$ respectively, describing the constraint $g_k(x) \leq x$ (recall that we want to find g_k^-). C_l are thus of the form $r_l + M_l x \leq x_l$. We form an edge in G from any node i to a node j iff x_j appears on the left hand side of the constraint $C_i : r_i + M_i x \leq x_i$ (we then say that $j \in C_i$).
- Split this graph into strongly connected components (SCC) and then sort the set of SCCs topologically. Let $C_1, C_2 \ldots C_m$ be the set of SCCs in decreasing order of depth.
- Corresponding to any SCC C_i, we form the linear program $Minimize\ \Sigma_{j\in C_i} x_j$ subject to $r_i + M_i x \leq x_i\ \forall i \in C_i$.
- Starting from C_1 , we start solving the corresponding linear programs in order. The result of the linear program corresponding to any C_i would either be a *finite* solution, *infeasible* solution or an *unbounded* solution. These are handled as follows :
  - (i) *Finite* solution : In this case we set the values of x_j, for all $j \in C_i$ to those at the extreme point where the least solution was obtained. Next we propagate these values upwards in the SCC chain i.e. we substitute any occurrence of these variables in the SCCs C_{i+1}, C_{i+2}... by their corresponding values.
  - (ii) *Infeasible* solution : In this case we set each x_j to $+\infty$ for all $i \in C_i$. These values are then propagated up the SCC chain in the same way as in the above case.
  - (iii) *Unbounded* solution : This is a very rare case. Unboundedness means that one or more variables $x_j (j \in C_i)$ are not bounded from below i.e. their minimum value is -∞. In order to find a value for these variables, we solve the linear program again with the same constraints but with the objective function being just x_j (this is done for all $j \in C_i$). If the corresponding linear program returns *unbounded*, x_j is set to -∞. As in the above cases the value of each x_j is propagated up the SCC chain.
- Finally, after all linear programs are solved, we return the vector $(x_1, x_2, \ldots x_p)$ as the least fixpoint of g.

4.2 Initial policy for zones and TCM

For meet policy, we do as for intervals in [CGG$^+$05]: we choose the left coordinate (respectively the right constraint) if the right coordinate (respectively the left entry) does not bring any information on a constraint between variables, i.e. is $+\infty$. We also give priority to constant entries. In case of a tie, we choose first the left coordinate. In the case of zone closure, we begin by paths of length one that is the zone itself. Initial closure policy on TCM chooses any vertex. The choice may sometimes depend on the LP programming method. For instance with a simplex algorithm that enumerates vertices in an order that decreases the objective function the first considered vertex may be taken as initial policy.

5 Experiments

A prototype has been developped for experiments. It takes C programs, constructs abstract semantic equations on the zone domain, solves them by the policy iteration algorithm of this article, and outputs the local invariants in text format. The front end is based on CIL [CIL], the equations are solve using the GLPK library [GLP] through its OCAML binding [Mim].

In this section, we show some experiments on simple programs, which can be found at `http://www.di.ens.fr/~goubault/policy.html`. These programs are briefly described below. We write in the columns from left to right, the number of lines, of variables, of while loops, the maximum depth of nested loops. Then we give the number of “elementary operations”/policy iteration that our analyzer used, the number of elementary operations/Kleene iterations in the case of the octagon analyzer, and the number of elementary operations/Kleene iterations for LPInv. These elementary operations are estimated, as follows: we indicate below columns “compl./#pols” (resp. compl./iter.oct., compl./iter. LPInv) the number of calls to the simplex solver: s/the average dimension: d (number of variables involved)/the average simplex iteration number: k (resp. the number of closure operations: c/assignment operations: a, and the same format as for our analyzer for the LPInv analyzer). These operations account for the main complexity in the three analyzers: the number of operations is of the order sd^2k for our analyzer and LPInv, and $cn^3 + an$, where n is the number of variables, for the octagon analyzer. We can see that the complexity is far less for our analyzer. The octagon analyzer spends a lot of operations doing closure operations, that we do not have to do. LPInv needs to solve the same order or even more linear programming problems, but more complex (i.e. needing more iterations to converge) and with a much higher dimensionality. Our method needs very few policies to converge, hence has few linear programming problems, which are very simple (very low dimensionality in particular) because of the SCC algorithm of Section 4.1.

Program	lines	vars	loops	depth	compl./#pols.	compl./iters.oct.	compl./iters.LPInv
test1	11	2	1	1	20/2 113/1.02/0.17	1132/7 138/14	14014/6 88/11.14/1.28
test1b	15	2	1	1	20/2 113/1.02/0.17	548/6 130/14	12952/6 78/11.6/1.23
test2	15	2	1	1	40/1 86/1.03/0.43	1268/12 309/16	31828/16 267/10.5/1.08
test3	14	2	1	1	34/1 96/1.03/0.33	1364/12 333/16	62348/16 282/14/1.12766
test4	13	2	2	2	68/3 124/1.27/0.34	906/4 220/13	50940/16 302/11.75/1.20
ex3	20	5	1	1	49/1 212/1.56/0.09	56250/8 225/13	22599614/16 1251/67.9/3.92
ex5	23	5	5	1	392/1 659/1.49/0.27	49370/23 394/24	33133177/20 3007/67.96/2.38

The results that our analyzer, A. Miné's octagon analyzer and LPInv (which uses octagons in our case) obtain are shown in Figure 2. We can see that although our analyzer is much faster, and computes in a less precise domain (zones) than octagons, it provides very similar invariants than both analyzers. It is even far more precise for `test2` and `test3` as already explained in Section 2.2. It provides in general better results than LPInv. The Octagon analyzer is better for programs of the style of `test1` since in that case, constraints on forms of the type $i + j$ (in zones, but not in octagons) are useful for getting invariants. Still, it suffices to unroll two times the main loop (`test1b`) to have comparable or even better results, with our analyzer.

6 Conclusion

We have described in this paper a new algorithm to compute efficiently and precisely, fixed points in relational abstract domains such as zones and TCMs, thus applicable to a large variety of situations.

There are two directions in which we would like to go from here. The first one is to extend this work to other domains, like the relational ones of [GP06], or domains dealing with pointers and general aliasing properties. The second direction of interest is the use of policy iteration algorithms to have better "incremental" analyzes [CDEN06]. As a matter of fact, one can hope that given a program P (identified with the abstract functionnal giving its semantics), a policy π giving the least fixpoint of P, light perturbations P' of P will only perturbate very little policy π. Hence π will be a very good initial policy guess for the policy iteration algorithm run on P'.

References

[CC76] P Cousot and R Cousot, *Static determination of dynamic properties of programs*, 2nd International Symposium on Programming, Paris, France, 1976.

[CC77] P. Cousot and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points*, Principles of Programming Languages 4 (1977), 238–252.

[CC91] P. Cousot and R. Cousot, *Comparison of the Galois connection and widening/narrowing approaches to abstract interpretation. JTASPEFL '91, Bordeaux*, BIGRE **74** (1991), 107–110.

[CC92] P. Cousot and R. Cousot, *Abstract interpretation frameworks*, Journal of Logic and Computation **2** (1992), no. 4, 511–547.

[CDEN06] C. Conway, D. Dams, S. A. Edwards, and K. Namjoshi, *Incremental algorithms for inter-procedural automaton-based program analysis*, Computer Aided Verification, Springer-Verlag, LNCS, 2006.

[CGG$^+$05] A. Costan, S. Gaubert, E. Goubault, M. Martel, and S. Putot, *A policy iteration algorithm for computing fixed points in static analysis of programs*, CAV, LNCS, vol. 3576, 2005, pp. 462–475.

[CH78] P. Cousot and N. Halbwachs, *Automatic discovery of linear restraints among variables of a program*, Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1978, pp. 84–97.

[Chv83] V. Chvátal, *Linear programming*, Freeman and Co., 1983.

[CIL] *CIL*, Tech. report, Berkeley University, http://manju.cs.berkeley.edu/cil/.

[GG98] S. Gaubert and J. Gunawardena, *The duality theorem for min-max functions*, C.R. Acad. Sci. **326** (1998), no. 1, 43–48.

[GLP] *GLPK*, Tech. report, Gnu, http://www.gnu.org/software/glpk/.

[GP06] E. Goubault and S. Putot, *Static analysis of numerical algorithms*, Static Analysis Symposium, Springer-Verlag, LNCS, 2006.

[HK66] A. J. Hoffman and R. M. Karp, *On nonterminating stochastic games*, Management sciences **12** (1966), no. 5, 359–370.

[How60] R. Howard, *Dynamic programming and markov processes*, Wiley, 1960.

[Mim] S. Mimram, *OcamlGLPK*, Tech. report, Gnu, http://ocaml-glpk.sourceforge.net/.

[Min01a] A. Miné, *A new numerical abstract domain based on difference-bound matrices*, PADO II, LNCS, vol. 2053, 2001, pp. 155–172.

[Min01b] A. Miné, *The octagon abstract domain*, AST 2001 in WCRE 2001, IEEE, 2001, pp. 310–319.

[Min04] A. Miné, *Weakly relational numerical abstract domains*, Ph.D. thesis, Ecole Nationale Supérieure, France, 2004.

[Min05] A. Miné, *The octagon domain library*, 2005.

[SCSM06] S. Sankaranarayanan, M. Colon, H. Sipma, and Z. Manna, *Efficient strongly relational polyhedral analysis*, VMCAI, LNCS, 2006, to appear.

[SSM05a] H. Sipma S. Sankaranarayanan and Z. Manna, *Lpinv: Linear programming invariant generator*, 2005.

[SSM05b] S. Sankaranarayanan, H. Sipma, and Z. Manna, *Scalable analysis of linear systems using mathematical programming*, VMCAI, LNCS, vol. 3385, 2005.

A Proofs of section 3

Proof of theorem 2 To prove this theorem we need to show that

(1) $F^- \sqsubseteq (g^-)^*$ for all $g \in \mathcal{G}$
(2) F^- is a fixed point of some policy

(1) Let g be a policy. By definition 1, we have $F(x) \sqsubseteq g^*(x)$ for all x. By Tarski's theorem, we know that the least fixed point of a monotone self map h on $\overline{\mathcal{L}}$ is given by $h^- = inf\{x \in \overline{\mathcal{L}} \mid h(x) \sqsubseteq x\}$. We then deduce that $F^- \sqsubseteq g^*(x)$ for all x hence

$$F^- \sqsubseteq (g^*)^-$$

Since the * and g are monotonic, we have $g^*((g^-)^*) \sqsubseteq g^*((g^-)) = (g(g^-))^* = (g^-)^*$. Therefore $(g^-)^*$ is a post fixed point of g^*. So by Tarskis's theorem we have

$$(g^*)^- \sqsubseteq (g^-)^*$$

From this relation and $F^- \sqsubseteq (g^*)^-$ we get $F^- \sqsubseteq (g^-)^*$ Since this relation is true for all g, we get the desired relation

$$F^- \sqsubseteq (inf\{g^- \mid g \in \mathcal{G}\})^*$$

(2) By the selection property there exists a policy h such that $F^- = F(F^-) = h(F^-)$. So F^- is a fixed point of h hence is greater than h^- which implies $h^- \sqsubseteq F^-$. By definition of $*$, we have $(h^-)^* \sqsubseteq h^- \sqsubseteq F^-$. Therefore,

$$(inf\{(g^-) \mid g \in \mathcal{G}\})^* \sqsubseteq F^-$$

. □

Proof of theorem 3 Correctness of the algorithm (property (i)) is trivial as it terminates only if the test $x_k = F(x_k)$ is satisfied.

We prove the property (ii) by induction on the number n of iterations of the algorithm that is the length of the sequence of successive g_k.

The basis case, $n = 0$, is trivial. For the induction case, we suppose that the algorithm has been iterated n times and that the sequence is such that $(g_{k+1}^-)^* \sqsubseteq (g_k^-)^*$ for $k < n$.

If $F((g_n^-)^*) = (g_n^-)^*$ then algorithm terminates and the property is true. Otherwise the map g_{n+1} is such that

$$F((g_n^-)^*) = g_{n+1}((g_n^-)^*) \qquad (1)$$

Moreover, by condition (1) of F we get $F((g_n^-)^*) \sqsubseteq g_n((g_n^-)^*)$ so that $g_{n+1}((g_n^-)^*) \sqsubseteq g_n((g_n^-)^*)$. Since $*$ is monotonic we have $g_{n+1}((g_n^-)^*) \sqsubseteq g_n((g_n^-)^*) \sqsubseteq g_n((g_n^-)) = g_n^-$. Therefore

$$g_{n+1}((g_n^-)^*) \sqsubseteq g_n^- \qquad (2)$$

Now as $F = (inf_{g \in \mathcal{G}} \ g)^*$ we have $F^* = (inf_{g \in \mathcal{G}} \ g)^{**}$ and since for all x $x^* = x^{**}$ we deduce $F^* = F$. So from (1) we get $g_{n+1}((g_n^-)^*) = (g_{n+1}((g_n^-)^*))^*$.

By monotonicity of $*$ and from (2) we have $g_{n+1}((g_n^-)^*) = (g_{n+1}((g_n^-)^*))^* \sqsubseteq (g_n^-)^*$ that is $(g_n^-)^*$ is a post fixed point of g_{n+1}. And as it not a fixed point of g_{n+1}, by Tarski's theorem we conclude that

$$(g_{n+1}^-)^* \sqsubseteq g_{n+1}^- \sqsubset (g_n^-)^*$$

□

Proof of corollary 4 The bound is an immediate consequence of theorem 3. □

B Comparison results

Program	*Invariant*	*InvariantOct.*	*InvariantLPInv*
`test1`	$\begin{cases} 1 \le i \le 12 \\ 0 \le j \le 10 \\ -3 \le j - i \le -1 \end{cases}$	$\begin{cases} 6 \le i \le 12 \\ \frac{9}{2} \le j \le 10 \\ -3 \le j - i \le -1 \end{cases}$	$\begin{cases} 6 \le i \le 13 \\ 4 \le j \le 10 \\ -3 \le j - i \le -1 \end{cases}$
`test1b`	$\begin{cases} 5 \le i \le 10 \\ 4 \le j \le 8 \\ -3 \le j - i \le -1 \end{cases}$	$\begin{cases} 7 \le i \le 10 \\ \frac{11}{2} \le j \le 8 \\ -3 \le j - i \le -1 \end{cases}$	$\begin{cases} \frac{15}{2} \le i \le 11 \\ \frac{11}{2} \le j \le 8 \\ -3 \le j - i \end{cases}$
`test2`	$\begin{cases} 150 \le i \le 174 \\ 98 \le j \le 99 \\ -76 \le j - i \le -51 \end{cases}$	$\begin{cases} 150 \le i \\ 98 \le j \le 99 \\ j - i \le -51 \end{cases}$	$\begin{cases} 150 \le i \\ j \le 99 \end{cases}$
`test3`	$\begin{cases} 150 \le i \le 174 \\ 98 \le j \le 99 \\ -76 \le j - i \le -51 \end{cases}$	$\begin{cases} 150 \le i \\ 98 \le j \le 99 \\ j - i \le -51 \end{cases}$	$\begin{cases} j \le 99 \\ j - i \le -51 \end{cases}$
`test4`	$\begin{cases} i = 101 \\ 1 \le j \\ -100 \le j - i \end{cases}$	$\begin{cases} 101 \le i \le 120 \\ 1 \le j \le 139 \\ -100 \le j - i \le 19 \end{cases}$	$\{ 101 \le i$
`ex3`	$\begin{cases} i = 1000 \\ k \le 1000 \\ m \le 100000 \\ m - i \le 99000 \\ m - k \le 99000 \end{cases}$	$\begin{cases} i = 1000 \\ j \le -900 \\ k \le 0 \\ j - i \le -1900 \\ k - i \le -1000 \end{cases}$	$\begin{cases} i = 1000 \\ j \le 899 \\ k \le 0 \\ -9000 \le l \\ -99000 \le m \\ 90000 \le l - m \\ 99000 \le k - m \end{cases}$
`ex5`	$\begin{cases} i = 1000 \\ 1000 \le j \\ k \le 100 \\ l \le 1000 \\ m \le 1 \\ k - i \le -900 \\ l - i \le 0 \\ m - i \le -999 \\ k - j \le -99 \\ l - j \le 0 \\ m - j \le -999 \end{cases}$	$\begin{cases} i = 1000 \\ 1000 \le j \le 1999 \\ -1898 \le k \le -1 \\ -897 \le l \le 1000 \\ m \le 1 \\ 0 \le j - i \le 999 \\ -2898 \le k - i \le -1001 \\ -3897 \le k - j \le -1001 \\ -1897 \le l - i \le 0 \\ -2896 \le l - j \le 0 \\ 1001 \le l - k \le 2898 \\ m - i \le -999 \\ m - j \le -999 \\ m - k \le 1899 \\ m - l \le 898 \end{cases}$	$\begin{cases} i = 1000 \\ 1000 \le j \le 1999 \\ k \le -1 \\ l \le 1000 \\ m \le 1 \\ 1001 \le l - k \end{cases}$

Fig. 2. Comparison between our analyzer, LPInv and Octagons