A policy iteration algorithm for computing fixed points in static analysis of programs

A. Costan[†], S. Gaubert[*], E. Goubault[+], M. Martel[+], S. Putot[+]

[1] † Polytehnica Bucarest
[2] * INRIA Rocquencourt
[3] + CEA Saclay

Abstract. We present a new method for solving the fixed point equations that appear in the static analysis of programs by abstract interpretation. We introduce and analyze a policy iteration algorithm for monotone self-maps of complete lattices. We apply this algorithm to the particular case of lattices arising in the interval abstraction of values of variables. We demonstrate the improvements in terms of speed and precision over existing techniques based on Kleene iteration, including traditional widening/narrowing acceleration mecanisms.

1 Introduction and related work

One of the important goals of static analysis by abstract interpretation (see Cousot & Cousot [11]) is the determination of invariants of programs. They are generally described by over approximation (abstraction) of the sets of values that program variables can take, at each control point of the program. And they are obtained by solving a system of (abstract) semantic equations, derived from the program to analyze and from the domain of interpretation, or abstraction, i.e. by solving a given fixed point equation in an order-theoretic structure.

Among the classical abstractions, there are the *non-relational* ones, such as the domain of *intervals* [11] (invariants are of the form $v_i \in [c1, c2]$), of *constant propagation* ($v_i = c$), of *congruences* [18] ($v_i \in a\mathbb{Z} + b$). Among the *relational* ones we can mention *polyedra* [31] ($\alpha_1 v_1 + \cdots + \alpha_n v_n \leq c$), *linear equalities* [26] ($\alpha_1 v_1 + \cdots + \alpha_n v_n = c$), *linear equalities modulo* [19] ($\alpha_1 v_1 + \cdots + \alpha_n v_n \equiv a$) or more recently the *octagon* domain [29] ($v_i - v_j \leq c$).

All these domains are (order-theoretic) lattices, for which we could think of designing specific fixed point equation solvers instead of using the classical, and yet not very efficient value iteration algorithms, known as Kleene's fixed point iteration. A classical way to improve these computations is to use widening/narrowing operators [12]. There exists concrete widening/narrowing operators for all classical domains of interpretation such as the one we mentioned above. They improve the rapidity of finding an over-approximated invariant at the expense of accuracy sometimes; i.e. they reach a fixed point, but not always the least fixed point of the semantic equations (we review some elements of this method in Section 2, and give examples in the case of the interval lattice).

In this paper, we introduce a new algorithm, based on policy iteration and not value iteration, that correctly and efficiently solves this problem (Section 3). It shows good performances in general with respect to various typical programs, see Section 4.4. We should add that this work started from the difficulty to find good widening and narrowing operators for domains used for characterizing the precision of floating-point computations, used by some of the authors in [17].

Policy iteration was introduced by Howard [23] to solve stochastic control problems with finite state and action space. In this context, a *policy* is a feedback strategy (which assigns to every state an action). The classical policy iteration generalizes Newton's algorithm to the equation $x = f(x)$, where f is monotone, non-differentiable, and convex. The convergence proof is based on the discrete version of the maximum principle for harmonic functions. This method is experimentally efficient, although its complexity is still not well understood theoretically. We refer the reader to the book of Puterman [32] for background.

It is natural to ask whether policy iteration can be extended to the case of zero-sum games: at each iteration, one fixes the strategy of one player, and solves a non-linear (optimal control problem) instead of a linear problem. This idea goes back to Hoffman and Karp [22]. The central difficulty in the case of games is to obtain the convergence, because the classical (linear) maximum principle cannot be applied any more. For this reason, the algorithm of [22] requires positivity conditions on transition probabilities, which do not allow to handle the case of deterministic games. In applications to static analysis, however, even the simplest fixed point problems lead to deterministic game problems. A policy iteration algorithm for deterministic games with ergodic reward has been given by Cochet-Terrasson, Gaubert, and Gunawardena [7,15]: the convergence proof relies on max-plus spectral theory, which provides nonlinear analogues of results of potential theory.

In the present paper, we present a new policy iteration algorithm, which applies to monotone self-maps of a complete lattice, defined by the infimum of a certain family satisfying a selection principle. Thus, policy iteration is not limited to finding fixed point that are numerical vectors or functions, fixed points can be elements of an abstract lattice. This new generality allows us to handle lattices which are useful in static analysis. For the fixed point problem, the convergence analysis is somehow simpler than in the ergodic case of [7,15]: we show that the convergence is guaranteed if we compute at each step the least fixed point corresponding to the current policy. The main idea of the proof is that the map which assigns to a monotone map its least fixed point is in some weak sense a morphism with respect to the inf-law, see Theorem 1. This shows that policy iteration can be used to compute the minimal fixed points, at least for a subclass of maps (Theorem 3 and Remark 5).

Other fixed point acceleration techniques have been proposed in the literature. There are mainly three types of fixed point acceleration techniques, as used in static analysis. The first one relies on specific information about the structure of the program under analysis. For instance, one can define refined iteration strategies for loop nests [2], or for interprocedural analysis [1]. These methods are completely orthogonal to the method we are introducing here, which does not use such structural properties. However, they might be combined with policy iteration, for efficient interprocedural analysis for instance. This is beyond the scope of this paper.

Another type of algorithm is based on the particular structure of the abstract domain. For instance, in model-checking, for reachability analysis, particular iteration strategies have been designed, so that to keep the size of the state space representation small (using BDDs, or in static analyzers by abstract interpretation, using binary decision graphs, see [28]), by a combination of breadth-first and depth-first strategies, as in [34]. For boolean equations, some authors have designed specific representations which allow for relatively fast least fixed point algorithms. For instance, [27] uses Bekic-Leszczyloiwski theorem. In strictness analysis, representation of boolean functions by "frontiers" has been widely used, see for instance [24] and [5]. Our method here is general, as hinted in Section 3. It can be applied to a variety of abstract domains, provided that we can find a "selection principle". This is exemplified here on the domain of intervals, but we are confident this can be equally applied to octagons and polyedra.

Last but not least, there are some general purpose algorithms, such as general widening/narrowing techniques, [12], with which we compare our policy iteration technique. There are also incremental or "differential" computations (in order not to compute again the functional on each partial computations) [25], [14]. In fact, this is much like the static partitioning technique some of the authors use in [33]. Related algorithms can be found in [13], [30] and [4]. We have not been able to compare these techniques with our algorithm yet.

The results of the present paper were announced in [10].

2 Kleene's iteration sequence, widenings and narrowings

In order to compare the policy iteration algorithm with existing methods, we briefly recall in this section the classical method based on Kleene's fixed point iteration, with widening and narrowing refinements (see [12]).

Let $(\mathcal{L}, \leq)$ be a complete lattice. We write $\perp$ for its lowest element, $\top$ for its greatest element, $\cup$ and $\cap$ for the meet and join operations, respectively. We say that a self-map f of a complete lattice $(\mathcal{L}, \leq)$ is *monotone* if $x \leq y \Rightarrow f(x) \leq f(y)$.

The least fixed point of a monotone f can be obtained by computing the sequence: $x^0 = \perp$, $x^{n+1} = f(x^n)$ $(n \geq 0)$, which is such that $x^0 \leq x^1 \leq \ldots$ If the sequence becomes stationary, i.e., if $x^m = x^{m+1}$ for some m, the limit x^m is the least fixed point of f. Of course, this procedure may be inefficient, and it need not even terminate in the case of lattices of infinite height, such as the simple interval lattice (that we use for abstractions in Section 4). For this computation to become tractable, *widening* and *narrowing* operators have been introduced, we refer the reader to [12] for a good survey. As we will only show examples on the interval lattice, we will not recall the general theory. Widening operators are binary operators ∇ on $\mathcal{L}$ which ensure that any finite Kleene iteration $x^0 = \bot$, $x^1 = f(x^0)$, ..., $x^{k+1} = f(x^k)$, followed by an iteration of the form $x^{n+1} = x^n \nabla f(x^n)$, for $n > k$, yields an ultimately stationary sequence, whose limit x^m is a *post fixed point* of f, i.e. a point x such that $x \geq f(x)$. The index k is a parameter of the least fixed point solver. Increasing k increases the precision of the solver, at the expense of time. In the sequel, we choose $k = 10$. Narrowing operators are binary operators Δ on $\mathcal{L}$ which ensure that any sequence $x^{n+1} = x^n \Delta f(x^n)$, for $n > m$, initialized with the above post fixed point x^m, is eventually stationary. Its limit is required to be a fixed point of f but not necessarily the least one.

```
void main() {
  int x=0;            // 1
  while (x<100) {     // 2
    x=x+1;            // 3
  }                   // 4
}
```

$$\begin{aligned} x_1 &= [0,0] \\ x_2 &=]-\infty, 99] \cap (x_1 \cup x_3) \\ x_3 &= x_2 + [1,1] \\ x_4 &= [100, +\infty[\cap (x_1 \cup x_3) \end{aligned}$$

Fig. 1. A simple integer loop and its semantic equations

Consider first the program at the left of Figure 1. The corresponding semantic equations in the lattice of intervals are given at the right of the figure. The intervals $x_1, \ldots, x_4$ correspond to the control points $1, \ldots, 4$ indicated as comments in the C code. We look for a fixed point of the function f given by the right hand side of these semantic equations. The standard Kleene iteration sequence is eventually constant after 100 iterations, reaching the least fixed point. This fixed point can be obtained in a faster way by using the classical (see [12] again) widening and narrowing operators:

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

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

The iteration sequence using widenings and narrowings takes 12 iterations because we chose $k = 10$, and it reaches the least fixed point of f :

$$\begin{array}{l} x_2^1 = [0,0] \\ x_3^1 = [1,1] \\ x_4^1 = \bot \\ \hline x_2^2 = [0,1] \\ x_3^2 = [1,2] \\ x_4^2 = \bot \\ \hline \vdots \\ \hline \end{array} \qquad \begin{array}{l} x_2^9 = [0,8] \\ x_3^9 = [1,9] \\ x_4^9 = \bot \\ \hline \text{(widening)} \\ x_2^{10} = [0,\infty[\\ x_3^{10} = [1,\infty[\\ x_4^{10} = [100,\infty[\end{array} \qquad \begin{array}{l} \text{(narrowing)} \\ x_2^{11} = [0,99[\\ x_3^{11} = [1,100] \\ x_4^{11} = [100,100] \end{array}$$

3 Policy iteration algorithm in complete lattices

3.1 Lower selection

To compute a fixed point of a self-map f of a lattice $\mathcal{L}$, we shall assume that f is effectively given as an infimum of a finite set $\mathcal{G}$ of "simpler" maps. Here, and in the sequel, the infimum refers to the pointwise ordering of maps. We wish to obtain a fixed point of f from the fixed points of the maps of $\mathcal{G}$. To this end, the following general notion will be useful.

Definition 1 (Lower selection). *We say that a set $\mathcal{G}$ of maps from a set X to a lattice $\mathcal{L}$* admits a lower selection *if for all $x \in X$, there exists a map $g \in \mathcal{G}$ such that $g(x) \leq h(x)$, for all $h \in \mathcal{G}$.*

Setting $f = \inf \mathcal{G}$, we see that $\mathcal{G}$ has a lower selection if and only if for all $x \in X$, we have $f(x) = g(x)$ for some $g \in \mathcal{G}$. We next illustrate this definition.

Example 1. Take $\mathcal{L} = \overline{\mathbb{R}}$, and consider the self-map of $\mathcal{L}$, $f(x) = \bigwedge_{1\leq i\leq m}(a_i + x) \vee b_i$, where $a_i, b_i \in \mathbb{R}$. Up to a trivial modification, this is a special case of *min-max function* [20,7,21]. The set $\mathcal{G}$ consisting of the m maps $x \mapsto (a_i + x) \vee b_i$ admits a lower selection. We represent on Figure 2 the case where $m = 5$, $b_1 = -5, a_1 = 2.5, b_2 = -3, a_2 = 0.5, b_3 = 1, a_3 = -3, b_4 = 1.5, a_4 = -4, b_5 = 2.5, a_5 = -4.5$. The graph of the map f is represented in bold.

3.2 Universal policy iteration algorithm

In many applications, and specially in static analysis of programs, the smallest fixed point is of interest. We shall denote by f^- the smallest fixed point of a monotone self-map f of a complete lattice $\mathcal{L}$, whose existence is guaranteed by Tarski's fixed point theorem. We first state a simple theoretical result which brings to light one of the ingredients of policy iteration.

Theorem 1. *Let $\mathcal{G}$ denote a family of monotone self-maps of a complete lattice $\mathcal{L}$ with a lower selection, and let $f = \inf \mathcal{G}$. Then $f^- = \inf_{g\in\mathcal{G}} g^-$.*

The proof of Theorem 1 and of the theorems that follow is in the appendix. Theorem 1 is related to a result of [9] concerning monotone self-maps of $\mathbb{R}^n$ that are nonexpansive in the sup-norm (see also the last chapter of [6]).

We now state a very general policy iteration algorithm. The input of the algorithm consists of a finite set $\mathcal{G}$ of monotone self-maps of a lattice $\mathcal{L}$ with a lower selection. When the algorithm terminates, its output is a fixed point of $f = \inf \mathcal{G}$.

Algorithm (PI: Policy iteration in lattices).

1. Initialization. *Set $k = 1$ and select any map $g_1 \in \mathcal{G}$.*
2. Value determination. *Compute a fixed point x^k of g_k.*
3. *Compute $f(x^k)$.*
4. *If $f(x^k) = x^k$, return x^k.*

5. Policy improvement. *Take g_{k+1} such that $f(x^k) = g_{k+1}(x^k)$. Increment k and goto Step 2.*

We next show that the algorithm does terminate when at each step, the smallest fixed-point of g_k, $x^k = g_k^-$ is selected. We call *height* of a subset $\mathcal{X} \subset \mathcal{L}$ the maximal cardinality of a chain of elements of $\mathcal{X}$.

Theorem 2. *Assume that $\mathcal{L}$ is a complete lattice and that all the maps of $\mathcal{G}$ are monotone. If at each step k, the smallest fixed-point $x^k = g_k^-$ of g_k is selected, then the number of iterations of Algorithm PI is bounded by the height of $\{g^- \mid g \in \mathcal{G}\}$, and a fortiori, by the cardinality of $\mathcal{G}$.*

Remark 1. Any $x^k \in \mathcal{L}$ computed by Algorithm PI is a post fixed point: $f(x^k) \leq x^k$. In static analysis of programs, such a x^k yields a valid, although suboptimal, information.

Example 2. We first give a simple illustration of the algorithm, by computing the smallest fixed point of the map f of Example 1. Let us take the first policy $g_5(x) = b_5 \vee (a_5 + x) = 2.5 \vee (-4.5 + x)$, which has two fixed points, $+\infty$ and 2.5. We choose the smallest one, $x^1 = 2.5$. We have $f(x^1) = g_2(x^1)$ where $g_2(x) = b_3 \vee (a_3 + x) = 1 \vee (-3 + x)$. We take for x^2 the smallest fixed point of g_2, $x^2 = 1$. Then, the algorithm stops since $f(x^2) = x^2$. This execution is illustrated in Figure 2. By comparison, the Kleene iteration (right) initialized at the point $-\infty$ takes 11 iterations to reach the fixed point. Observe that by modifying the values of a_2 and b_2, one may reduce arbitrarily the distance between the graph of the map $b_2 \vee (a_2 + x)$ and the diagonal, so that the number of iterations of the Kleene iteration algorithm may be arbitrarily large, whereas the number of iterations of Algorithm PI always remains bounded by m.

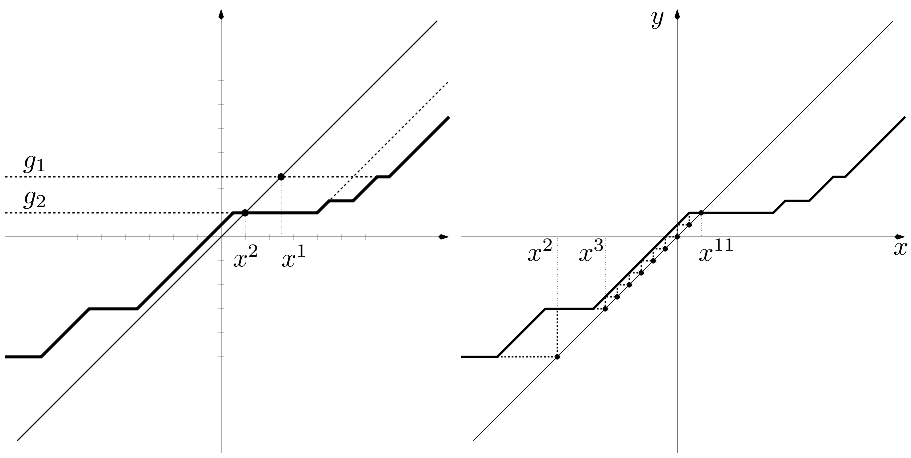

Fig. 2. Policy iteration (left) versus Kleene iteration (right)

Remark 2. A crucial difficulty in the application of the algorithm to static analysis is that even when the smallest fixed points $x^k = g_k^-$ are always chosen, the policy iteration algorithm need *not* return the smallest fixed point of f. For instance, in Example 2, if one takes the initial policy $x \mapsto a_1 \vee (b_1 + x)$ or $x \mapsto a_2 \vee (b_2 + x)$, we get $x^1 = \infty$, and the algorithm stops with a fixed point of f, ∞, which is non minimal. This shows the importance of the initial policy and of the update rule for policies.

Although Algorithm PI may terminate with a nonminimal fixed point, it is often possible to check that the output of the algorithm is actually the smallest fixed point thanks to the following kind of results. We consider the special situation where f is a monotone self-map of $\overline{\mathbb{R}}^n$, with a restriction $\mathbb{R}^n \to \mathbb{R}^n$ that is nonexpansive for the sup-norm, meaning that $\|f(x) - f(y)\|_\infty \leq \|x - y\|_\infty$, for all $x, y \in \mathbb{R}^n$. We shall say that such maps f have *Property N.* (For instance, the maps in Example 1 all have Property N.) The following theorem identifies situations where the *uniqueness* of the terminal policy guarantees that the fixed point returned by Algorithm PI is the smallest one.

Theorem 3. *Assume that $\mathcal{G}$ is a finite set of monotone self-maps of $\overline{\mathbb{R}}^n$ that all have Property N, that $\mathcal{G}$ has a lower selection, and let $f = \inf \mathcal{G}$. If Algorithm PI terminates with a finite fixed point $x^k = g_k^-$ such that there is only one $g \in \mathcal{G}$ such that $f(x^k) = g(x^k)$, then, x^k is the smallest finite fixed point of f.*

Remark 3. The nonexpansiveness assumption cannot be dispensed with in Theorem 3. Consider the self-map of $\overline{\mathbb{R}}$, $f(x) = 0 \wedge (1 + 2x)$, and take the set $\mathcal{G}$ consisting of the maps $x \mapsto 0$ and $x \mapsto 1 + 2x$. Algorithm PI initialized with the map $g_1 = 0$ stops immediately with the fixed point $x^1 = 0$, and g_1 is the only map g in $\mathcal{G}$ such that $g(0) = f(0)$, but x^1 is a nonminimal finite fixed point of f, since $f(-1) = -1$.

Remark 4. Of course, when x^k is the smallest finite fixed point of f, there may exist a smaller fixed point, with some coordinates equal to $-\infty$, so Theorem 3 reduces to some extent the search of the minimal fixed point to questions concerning the pattern (set of coordinates distinct from $-\infty$), which are sometimes easier (the case where $x_i = -\infty$ represents situations in static analysis where a fragment of the program is never executed).

Remark 5. When the policy g such that $f(x^k) = g(x^k)$ is not unique, we can check whether x^k is the smallest finite fixed point of f in the following way. We scan the set of maps $g \in \mathcal{G}$ such that $g(x^k) = f(x^k)$, until $g^- < x$ or all these maps g have been scanned. In the former case, an improved post fixed point of f has been found, and by nonexpansiveness of f, it can be shown that there is a smaller finite fixed point of f (which can be computed by restarting the policy iteration algorithm with the initial map $g_1 := g$). In the latter case, a small variation of the proof of Theorem 3 shows that x is the smallest finite fixed point of f (if all the maps in $\mathcal{G}$ have Property N).

Algorithm PI requires to compute at every step a fixed point x^k of the map g_k, and if possible, the minimal one, g_k^-. An obvious way to do so is to apply Kleene iteration to the map g_k. Although this may seem surprising at the first sight, this implementation may preserve the performance of the algorithm. In fact, it is optimal in Example 2, since Kleene iteration converges in only one step for every map g_k. In many cases, however, some precise information on the map g_k is available, and policy iteration will benefit from fast algorithms to compute x^k. For instance, the classical policy iteration algorithm of Howard, concerns the special case where the maps g_k are affine. In that case, the fixed point x^k is obtained by solving a linear system. A non classical situation, where the maps g_k are dynamic programming operators of deterministic optimal control problems, i.e., max-plus linear maps, is solved in [8,15].

4 Application to the lattice of intervals in static analysis

In the sequel, we shall consider the set $\mathcal{I}(\mathbb{R})$ of closed intervals of $\mathbb{R}$. This set, ordered by inclusion, is a complete lattice. It will be convenient to represent an interval $I \in \mathcal{I}(\mathbb{R})$ as $I = [-a, b] := \{x \in \mathbb{R} \mid -a \leq x \leq b\}$ with $a, b \in \mathbb{R} \cup \{\pm\infty\}$. We changed the sign in order to get a monotone map $\psi : I \mapsto (a = -\inf I, b = \sup I)$, from $\mathcal{I}(\mathbb{R}) \to \overline{\mathbb{R}}^2$. Observe that ψ is a right inverse of $\imath : (a, b) \mapsto [-a, b]$. By extending ψ and $\imath$ to products of spaces, entrywise, we see that any monotone self-map f of $\mathcal{I}(\mathbb{R}^n)$ induces a monotone self-map of $(\overline{\mathbb{R}}^2)^n$, $\psi \circ f \circ \imath$, that we call the *lift* of f. It is useful to keep in mind that the minimal fixed point of f is the image by $\imath$ of the minimal fixed point of its lift, although our algorithms apply preferably to the map f rather than to its lift.

4.1 The interval abstraction

We consider a toy imperative language with the following instructions:

1. loops: `while (condition) instruction;`
2. conditionals: `if (condition) instruction [else instruction];`
3. assignment: `operand = expression;` We assume here that the arithmetic expressions are built on a given set of variables (belonging to the set Var), and use operators +, -, * and /, together with numerical constants (only integers here for more simplicity).

There is a classical [12] Galois connection relating the powerset of values of variables to the product of intervals (one for each variable). This is what gives the correction of the classical [12] abstract semantics $[\![.]\!]$, with respect to the standard collecting semantics of this language. $[\![.]\!]$ is given by a set of equations over the variables $x_1, \ldots, x_n$ of the program that we will show on some examples. Each variable x_i is interpreted as an interval $[-x_i^-; x_i^+]$.

4.2 Selection property for a family of finitely generated functions on intervals

We now define a class of monotone self-maps of $\mathcal{I}(\mathbb{R})$, which is precisely the class of functions arising from the semantic equations of the previous section. This class may be thought of as an extension of the min-max functions introduced in [20]. For an interval $I = [-a, b]$, we set $\uparrow I \stackrel{\text{def}}{=} [-a, \infty[$ and $\downarrow I \stackrel{\text{def}}{=}]-\infty, b]$.

Definition 2. *A finitely generated function of intervals, $(\mathcal{I}(\mathbb{R}))^n \to (\mathcal{I}(\mathbb{R}))^p$, is a map f whose coordinates $f_j : x = (x_1, \dots, x_n) \mapsto f_j(x)$ are terms of the following grammar G:*

$$CSTE ::= \ [-a, b] \quad VAR ::= \ x_i$$

$$\begin{array}{rllll}
EXPR ::= & CSTE & \mid VAR & \mid EXPR + EXPR & \mid \\
& EXPR * EXPR & \mid EXPR/EXPR & \mid EXPR - EXPR & \\
TEST ::= & \uparrow EXPR \cap EXPR & \mid \downarrow EXPR \cap EXPR & \mid CSTE \cap EXPR & \\
G ::= & EXPR & \mid TEST & \mid G \cup G &
\end{array}$$

where i can take arbitrary values in $\{1, \dots, n\}$, and a, b can take arbitrary values in $\overline{\mathbb{R}}$.

We write $\mathcal{F}$ for the set of such functions. The variables $x_1, \dots, x_n$ correspond to the different variables in Var. We set $x_i^- = -\inf x_i$, $x_i^+ = \sup x_i$, so that $x_i = [-x_i^-, x_i^+]$. Non-terminals $CSTE$, VAR, $EXPR$ and $TEST$ do correspond to the semantics of constants, variables, arithmetic expressions, and (simple) tests. For instance, the fixed point equation at the right of Figure 1 is of the form $x = f(x)$ where f is a finitely generated function of intervals.

In order to write maps of $\mathcal{F}$ as infima of simpler maps, when $I = [-a, b]$ and $J = [-c, d]$, we also define $l(I, J) = I$ (l is for "left"), $r(I, J) = J$ (r for "right"), $m(I, J) = [-a, d]$ and $m^{\text{op}}(I, J) = [-c, b]$ (m is for "merge"). These four operators will define the four possible policies on intervals, as shown in Proposition 1 below.

Let $G_\cup$ be the grammar, similar to G except that we cannot produce terms with $\cap$.

$$\begin{array}{rlllll}
G_\cup ::= & EXPR & \mid \uparrow EXPR & \mid \downarrow EXPR & \mid G_\cup \cup G_\cup & \mid \\
& l(G_\cup, G_\cup) & \mid r(G_\cup, G_\cup) & \mid m(G_\cup, G_\cup) & \mid m^{\text{op}}(G_\cup, G_\cup) &
\end{array}$$

We write $\mathcal{F}_\cup$ for the set of functions defined by this grammar. Terms $l(G, G)$, $r(G, G)$, $m(G, G)$ and $m^{\text{op}}(G, G)$ represent respectively the left, right, m and m^{op} policies.

The intersection of two intervals, and hence, of two terms of the grammar, interpreted in the obvious manner as intervals, is given by the following formula:

$$G_1 \cap G_2 = l(G_1, G_2) \cap r(G_1, G_2) \cap m(G_1, G_2) \cap m^{\text{op}}(G_1, G_2) \qquad (1)$$

To a finitely generated function of intervals $f \in \mathcal{F}$, we associate a family $\Pi(f)$ of functions of $\mathcal{F}_\cup$ obtained in the following manner: we replace each occurrence of a term $G_1 \cap G_2$ by $l(G_1, G_2)$, $r(G_1, G_2)$, $m(G_1, G_2)$ or $m^{\text{op}}(G_1, G_2)$. We call such a choice a *policy.* Using Equation (1), we get:

Proposition 1. *If f is a finitely generated function of intervals, the set of policies $\Pi(f)$ admits a lower selection. In particular, $f = \inf \Pi(f)$.*

4.3 Implementation principles of the policy iteration algorithm

A simple static analyzer has been implemented in C++. It consists of a parser for a simple imperative language (a very simplified C), a generator of abstract semantic equations using the interval abstraction, and the corresponding solver, using the policy iteration algorithm described in Section 3.

A policy is a table that associates to each intersection node in the semantic abstraction, a value modeling which policy is chosen among l, r, m or m^{op}, in Equation (1). There is a number of heuristics that one might choose concerning the initial policy, which should be a guess of the value of $G_1 \cap G_2$ in Equation (1). The choice of the initial policy may be crucial, since some choices of the initial policy may lead eventually to a fixed point which is not minimal. (In such cases, Remark 5 should be used: it yields an heuristics to improve the fixed point, which can be justified rigorously by Theorem 3, when the lift of f has Property N.) The current prototype makes a sensible choice: when a term $G_1 \cap G_2$ is encountered, if a finite constant bound appears in G_1 or G_2, this bound is selected. Moreover, if a $+\infty$ upper bound or $-\infty$ lower bound appears in G_1 or G_2, then, this bound is not selected, unless no other choice is available (in other words, choices that give no information are avoided). When the applications of these rules is not enough to determine the initial policy, we choose the bound arising from the left hand side term. Thus, when $G_1 = [-a, \infty[$, the initial policy for $G_1 \cap G_2$ is $m(G_1, G_2)$, which keeps the lower bound of G_1 and the upper bound of G_2.

The way the equations are constructed, when the terms $G_1 \cap G_2$ correspond to a test on a variable (and thus no constant choice is available for at least one bound), this initial choice means choosing the constraint on the current variable brought on by this test, rather than the equation expressing the dependence of current state of the variable to the other states. These choices often favor as first guess an easily computable system of equations.

We then compute the fixed point of the reduced equations, using if possible specific algorithms. In particular, when the lift of f is a min-max function, shortest path type algorithms may be used, in the spirit of [7,15]. Linear or convex programming might also be used in some cases. For the time being, we only use a classical Kleene like value iteration algorithm, discussed in Section 4.4.

We then proceed to the improvement of the policy, as explained in Section 3. In short, we change the policy at each node for which a fixed point of the complete system of equations is not reached, and compute the fixed point of the new equations, until we find a fixed point of the complete system of equations. Even when this fixpoint is reached, using Remark 5 can allow to get a smaller fixpoint in some cases, when the current fixpoint is obtained for several policies.

In the following examples, we count the number of policies the algorithm had to consider in order to reach the least fixed point, and the cost of solving the remaining equations, without intersection. But the overall speedup of policy iteration algorithms could be improved by using specific solvers.

4.4 Examples and comparison with Kleene's algorithm

In this section, we discuss a few typical examples, that are experimented using our prototype implementation. We compare the policy iteration algorithm with Kleene's iteration sequence with widenings and narrowings (the very classical one of [12]), called Algorithm K here. For both algorithms, we compare the accuracy of the results, and the number of fixed point iterations.

We did not experiment specific algorithms for solving equations in $G_{\cup}$ (meaning, without intersections), as we consider this to be outside the scope of this paper, so we chose to use an iterative solver (algorithm K') for each policy. Algorithm K' is exactly the same solver as algorithm K, but used on a smaller class of functions, for one policy. So Algorithm PI will run Algorithm K' at every value determination step (Step 2). Of course, using Algorithm K' instead of a specific solver is an heuristics, since the convergence result, Theorem 2, requires that at every value determination step, the smallest fixed point is computed. We decided to widen intervals, both in Algorithms K and K', only after ten standard Kleene iterations. This choice is conventional, and in most examples below, one could argue that an analyzer would have found the right result with only two Kleene iterations. In this case, the speedup obtained by the policy iteration algorithm would be far less; but it should be argued that in most static analyzers, there would be a certain unrolling before trying to widen the result. In the sequel we will count also the number of "elementary operations" performed by algorithms K and PI. We count one per elementary operations (+, - etc.), min and max (used when taking unions and intersections), tests ($\leq$, $\geq$, $=$) used for checking whether we reach local fixed points during iterations of algorithms K and K'.

A simple typical (integer) loop is shown on Figure 1, together with the equations generated by the analyzer. The original policy is m^{op} in equation 2 in Figure 1 (by equation i, we mean the equation which determines the state of variables at control point i, here x_2), and m in the equation determining x_4, This is actually the right policy on the spot, and we find in one iteration (and 34 elementary operations), the correct result (the least fixed point). This is to be compared with the 12 iterations of Algorithm K (and 200 elementary operations), in Section 2. In the sequel, we put upper indices to indicate at which iteration the abstract value of a variable is shown. Lower indices are reserved as before to the control point number.

The analysis of the program below leads to an actual policy improvement:

```
void main(){            while (j >= i) { // 3
int i,j;                  i = i+2; // 4
i=1; // 1                 j = -1+j; // 5
j=10; // 2              } // 6 }
```

Semantic equations at control points 3 and 6 are

$$(i_3, j_3) = (]-\infty, \max(j_2, j_5)] \cap (i_2 \cup i_5), \ \ [\min(i_2, i_5), +\infty[\cap (j_2 \cup j_5))$$
$$(i_6, j_6) = ([\min(j_2, j_5) + 1, +\infty[\cap (i_2 \cup i_5), \ \ ]-\infty, \max(i_2, i_5) - 1] \cap (j_2 \cup j_5))$$

The algorithm starts with policy m^{op} for variable i in equation 3, m for variable j in equation 3, m for variable i equation 6 and m^{op} in equation 6, variable j. The first iteration using Algorithm K' with this policy, finds the value $(i_6^1, j_6^1) = ([1, 12], [0, 11])$. But the value for variable j given by equation 6, given using the previous result, is $[0, 10]$ instead of $[0, 11]$, meaning that the policy on equation 6 for j should be improved. The minimum (0) for j at equation 6 is reached as the minimum of the right argument of $\cap$. The maximum (10) for j at equation 6 is reached as the maximum of the right argument of $\cap$. Hence the new policy one has to choose for variable j in equation 6 is r. In one iteration of Algorithm K' for this policy, one finds the least fixed point of the system of semantic equations, which is at line 6, $(i_6^2, j_6^2) = ([1, 12], [0, 10])$. Unfortunately, this fixed point is reached by several policies, and Remark 5 is used, leading to another policy iteration. This in fact does not improve the result since the current fixed point is the smallest one. Algorithm PI uses 2 policy iterations, 5 values iterations and 272 elementary operations. Algorithm K takes ten iterations (and 476 elementary operations) to reach the same result.

Some benchmarks In the following table, we describe preliminary results that we obtained using Algorithms K and PI on simple C programs, consisting essentially of loop nests. We indicate for each program (available on http://www.di.-ens.fr/˜goubault/Politiques) the number of variables, the number of loops, the maximal depth of loop nests, the number of policy iterations slash the total number of potential policies, value iterations and elementary operations for each algorithm (when this applies). The last column indicates the speedup ratio of Algorithm PI with respect to K, measured as the number of elementary operations K needs over the number that PI needs.

Program	vars	loops	depth	pols./tot.	iters.K/PI	ops.K/PI	speedup
`test1`	1	1	1	1/16	12/1	200/34	*5.88*
`test2`	2	1	1	2/256	10/5	476/272	*1.75*
`test3`	1	1	1	1/256	5/2	44/83	*0.51*
`test4`	5	5	1	0/1048576	43/29	2406/1190	*2.02*
`test5`	2	2	2	0/256	164/7	5740/198	*28.99*
`test6`	2	2	2	1/1048576	57/19	2784/918	*3.03*
`test7`	3	3	2	1/4096	62/13	3242/678	*4.78*
`test8`	3	3	3	0/4096	60/45	3916/2542	*1.54*
`test9`	3	3	3	2/4096	185/41	11348/1584	*7.16*
`test10`	4	4	3	3/65536	170/160	11274/10752	*1.05*

The relative performances can be quite difficult to predict (for instance, for `test3`, Algorithm K is about twice as fast as PI, which is the only case in the benchmark), but in general, in nested loops Algorithm PI can outperform Algorithm K by a huge factor. Furthermore, for nested loops, Algorithm PI can even be faster and more precise than Algorithm K as in the case of `test10`:

```
int main() {
  int i,j,k;
  i = 0; //1
  k = 9; //2
  j = -100; //3
  while (i <= 100) //4 {
    i = i + 1; //5
    while (j < 20) //6
      j = i+j; //7 //8
    k = 4; //9
    while (k <=3) //10
      k = k+1; //11 //12
  } //13 }
```

Algorithm PI reaches the following fixed point, at control point 13, in 13 value iterations, $i = [101, 101]$, $j = [-100, 120]$ and $k = [4, 9]$ whereas Algorithm K only finds $i = [101, 101]$, $j = [-100, +\infty]$ and $k = [4, 9]$ in 62 iterations. The fact that Algorithm K does not reach the least fixed point can be explained as follows. At control point 4, Algorithm K finds successively:

	then:	up to:	then widening:
$i_4 = [0, 0]$	$i_4 = [0, 1]$	$i_4 = [0, 9]$	$i_4 = [0, +\infty]$
$j_4 = [-100, -100]$	$j_4 = [-100, 20]$	$j_4 = [-100, 28]$	$j_4 = [-100, +\infty]$
$k_4 = [9, 9]$	$k_4 = [4, 9]$	$k_4 = [4, 9]$	$k_4 = [4, 9]$

From here on, there is no way, using further decreasing iterations, to find that j is finite (and less than 20) inside the outer while loop, since this depends on a relation between i and j that cannot be simulated using this iteration strategy.

5 Future work

We have shown in this paper that policy iteration algorithms can lead to fast and accurate solvers for abstract semantic equations, such as the ones coming from classical problems in static analysis. We still have some heuristics in the choice of initial policies we would like to test (using for example a dynamic initial choice, dependent on the values of variables after the first fixpoint iterations), and the algorithmic consequences of Theorem 3 should be investigated.

One of our aims is to generalize the policy iteration algorithm to more complex lattices of properties, such as the one of octagons (see [29]). We would like also to apply this technique to symbolic lattices (using techniques to transfer numeric lattices, see for instance [35]). Finally, we should insist on the fact that a policy iteration solver should ideally rely on better solvers than value iteration ones, for each of its iterations (i.e. for a choice of a policy). The idea is that, choosing a policy simplifies the set of equations to solve, and the class of such sets of equations can be solved by better specific solvers. In particular, we would like to experiment the policy iteration algorithms again on grammar $G_\cup$, so that we would be left with solving, at each step of the algorithm, purely numerical constraints, at least in the case of the interval abstraction. For numerical constraints, we could then use very fast numerical solvers, for large classes of functions (linear equations but not only).

References

1. F. Bourdoncle. Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4):407–435, 1992.
2. F. Bourdoncle. Efficient chaotic iteration strategies with widenings. Number 735, pages 128–141. Lecture Notes in Computer Science, Springer-Verlag, 1993.
3. Ronald E. Bruck, Jr. Properties of fixed-point sets of nonexpansive mappings in Banach spaces. *Trans. Amer. Math. Soc.*, 179:251–262, 1973.
4. Baudouin Le Charlier and Pascal Van Hentenryck. A universal top-down fixpoint algorithm. Technical Report CS-92-25, Department of Computer Science, Brown University, May 1992. Mon, 11 Sep 100 15:20:30 GMT.
5. Chris Clack and Simon L. Peyton Jones. Strictness Analysis — A Practical Approach. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 35–49, Nancy, France, September 16–19, 1985. Springer, Berlin.
6. J. Cochet-Terrasson. *Algorithmes d'itération sur les politiques pour les applications monotones contractantes.* Thèse, spécialité mathématiques et automatique, École des Mines, Dec. 2001.
7. J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. A constructive fixed point theorem for min-max functions. *Dynamics and Stability of Systems*, 14(4):407–433, 1999.
8. J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. A constructive fixed point theorem for min-max functions. *Dynamics and Stability of Systems*, 14(4):407–433, 1999.
9. J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. Policy iteration algorithms for monotone nonexpansive maps. Draft, 2001.
10. A. Costan. Analyse statique et itération sur les politiques. Technical report, CEA Saclay, report number DTSI/SLA/03-575/AC, and Ecole Polytechnique, August 2003.
11. 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*, pages 238–252, 1977.
12. P. Cousot and R. Cousot. Comparison of the Galois connection and widening/narrowing approaches to abstract interpretation. JTASPEFL '91, Bordeaux. *BIGRE*, 74:107–110, October 1991.
13. D. Damian. Time stamps for fixed-point approximation. *ENTCS*, 45, 2001.
14. C. Fecht and H. Seidl. Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems. pages 90–104, 1998.
15. S. Gaubert and J. Gunawardena. The duality theorem for min-max functions. *C. R. Acad. Sci. Paris.*, 326, Série I:43–48, 1998.
16. S. Gaubert and J. Gunawardena. The Perron-Frobenius theorem for homogeneous, monotone functions. *Trans. of AMS*, 356(12):4931–4950, 2004.
17. E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: A simple abstract interpreter. *Lecture Notes in Computer Science*, 2305, 2002.
18. P. Granger. *Analyse de congruences.* PhD thesis, Ecole Polytechnique, 1990.
19. P. Granger. Static analysis of linear congruence equalities among variables of a program. In Samson Abramsky and T. S. E. Maibaum, editors, *TAPSOFT '91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Colloquium on Trees in Algebra and Programming*

(CAAP'91), volume 493 of *Lecture Notes in Computer Science*, pages 169–192. Springer-Verlag, 1991.
20. J. Gunawardena. Min-max functions. *Discrete Event Dynamic Systems*, 4:377–406, 1994.
21. J. Gunawardena. From max-plus algebra to nonexpansive maps: a nonlinear theory for discrete event systems. *Theoretical Computer Science*, 293:141–167, 2003.
22. A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Sci.*, 12:359–370, 1966.
23. R. Howard. *Dynamic Programming and Markov Processes.* Wiley, 1960.
24. L. S. Hunt. *Abstract Interpretation of Functional Languages: From Theory to Practice*. Ph.D. thesis, Department of Computing, Imperial College, London, UK, 1991.
25. Kwangkeun Yi Hyunjun Eo. An improved differential fixpoint iteration method for program analysis. November 2002.
26. M. Karr. Affine relationships between variables of a program. *Acta Informatica*, (6):133–151, 1976.
27. Viktor Kuncak and K. Rustan M. Leino. On computing the fixpoint of a set of boolean equations. Technical Report MSR-TR-2003-08, Microsoft Research (MSR), December 2003.
28. Laurent Mauborgne. Binary decision graphs. In A. Cortesi and G. Filé, editors, *Static Analyis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 101–116. Springer-Verlag, 1999.
29. A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001. `http://www.di.ens.fr/~mine/publi/article-mine-ast01.pdf`.
30. R. A. O'Keefe. Finite fixed-point problems. In Jean-Louis Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming (ICLP '87)*, pages 729–743, Melbourne, Australia, May 1987. MIT Press.
31. P. and N. Halbwachs. Discovery of linear restraints among variables of a program. 1978.
32. Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons Inc., New York, 1994. A Wiley-Interscience Publication.
33. S. Putot, E. Goubault, and M. Martel. Static analysis-based validation of floating-point computations. Springer-Verlag, 2003.
34. K. Ravi and F. Somenzi. Efficient fixpoint computation for invariant checking. In *International Conference on Computer Design (ICCD '99)*, pages 467–475, Washington - Brussels - Tokyo, October 1999. IEEE.
35. A. Venet. Nonuniform alias analysis of recursive data structures and arrays. In *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002*, volume 2477 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2002.

A Proof of Theorems 1,2 and 3

This appendix will be omitted in the final version of the present manuscript: we include it for the convenience of the referees.

Proof of Theorem 1. By Tarski's theorem, the least fixed point of a monotone self-map h of $\mathcal{L}$ is $h^- = \inf\{x \in \mathcal{L} \mid h(x) \leq x\}$. Therefore, the map $h \mapsto h^-$ is monotone. It follows that $f^- \leq \inf_{g\in\mathcal{G}} g^-$. Since $\mathcal{G}$ has a lower selection, we have $f^- = f(f^-) = h(f^-)$ for some $h \in \mathcal{G}$. Therefore, $h^- \leq f^-$, which shows that $\inf_{g\in\mathcal{G}} g^- \leq f^-$. □

Proof of Theorem 2. When the policy is improved at step k, we have $f(x^k) < x^k$, and we choose g_{k+1} such that $g_{k+1}(x^k) = f(x^k)$, so that $g_{k+1}(x^k) \leq x^k$. By Tarski's fixed point theorem, $g_{k+1}^- = \inf\{x \in \mathcal{L} \mid g_{k+1}(x) \leq x\}$. It follows that $g_{k+1}^- \leq g_k^-$. Moreover, $g_{k+1}^- \neq g_k^-$, because g_k^- is not a fixed point of g_{k+1}. Thus, the sequence $g_1^-, g_2^-, \dots$ produced by the algorithm is strictly decreasing, which implies that the number of iterations is bounded by the height of $\{g^- \mid g \in \mathcal{G}\}$. □

Proof of Theorem 3. Let $u := x^k = g_k^-$, and assume by contradiction that there is a finite fixed point v of f such that $u \leq v$ does not hold. Then, $z =: u \wedge v < x$. Since f is monotone, $f(z) \leq f(u) \wedge f(v) = z$. We deduce that the sequence $(f^k(z))_{k\geq 1}$ is nonincreasing. This sequence is bounded because f has a finite fixed point and f is nonexpansive. Thus, $f^k(z)$ converges to a finite fixed point, $w < u$. Define the map $\phi : \lambda \mapsto (\lambda + u) \wedge w$. Observe that ϕ is continuous, that $\phi(\lambda) = w$ for $\lambda \leq \lambda_1 := -\|u - w\|_\infty$, and that $\phi(0) = u$. The set of fixed points of a nonexpansive self-map of $\mathbb{R}^n$ is the image of $\mathbb{R}^n$ by a nonexpansive retraction P (this holds more generally for nonexpansive maps having certain compactness properties, see [3]). Moreover, the retraction P may be chosen to be monotone when f is monotone. (To see this, take for instance $P = Q \circ R$, where $R(x) = \limsup_k f^k(x)$ and $Q(x) = \lim_k f^k(x)$, and use the proof of Lemma 3 of [16].) Let $y_\lambda := P(\phi(\lambda))$. Since P is monotone, $\phi(\lambda) \leq u$ implies that $y_\lambda \leq u$. Moreover, $y_{\lambda_1} = P(w) = w < u$ and $y_0 = P(u) = u$. Consider now any map g distinct from g_k. Since g_k is the only policy that can be selected at point u, $g(u) \neq u$, and since g is continuous, we can find a neighborhood V_g of u such that $g(y) \neq y$ for all $y \in V_g$. Let V denotes the intersection of the neighborhoods V_g, for $g \in \mathcal{G}$ and $g \neq g_k$. By continuity of $\lambda \mapsto y_\lambda$, we can find $\lambda \in [\lambda_1, 0[$ such that $y_\lambda \in V$ and $y_\lambda < u$. By construction of V, we must have $y_\lambda = f(y_\lambda) = g_k(y_\lambda)$. Since $y_\lambda < u$, we contradict the fact that u is the minimal fixed point of g_k. □