Acceleration of fixpoint computations in static analysis of programs by policy iteration algorithms

A. Costan†, S. Gaubert∗, E. Goubault+, M. Martel+, S. Putot+

† Polytehnica Bucarest
∗ INRIA Rocquencourt
+ CEA Saclay

Abstract

We present a new method for solving the fixpoint equations that appear in the static analysis of programs by abstract interpretation. This method is based on policy iteration algorithms and not Kleene like value iteration algorithms. These algorithms have been introduced in the sixties for solving optimal control problems, and extended more recently to the case of games with two players and zero sum. We apply this technique to the particular case of the interval abstraction of values of variables, and demonstrate the improvements over more classical techniques, including traditional widening/narrowing acceleration mecanisms.

1 Introduction and related work

One of the important goals of static analysis by abstract interpretation 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. These 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 fixpoint equation in an order-theoretic structure.

Among the classical abstractions, there are the *non-relational* ones, such as the domain of *intervals* [CC77] (invariants are of the form $v_i \in [c1, c2]$), of *constant propagation* ($v_i = c$), of *congruences* [Gra90] ($v_i \in a\mathbb{Z} + b$). Among the *relational* ones we can mention *polyedra* [PH] ($\alpha_1 v_1 + \cdots + \alpha_n v_n \leq c$), *linear equalities* [Kar76] ($\alpha_1 v_1 + \cdots + \alpha_n v_n = c$), *linear equalities modulo* [Gra91] ($\alpha_1 v_1 + \cdots + \alpha_n v_n \equiv a$) or more recently the *octagon* domain [Min01] ($v_i - v_j \leq c$).

All these domains are (order-theoretic) lattices, for which we could think of designing specific fixpoint equation solvers instead of using the classical, and yet not very efficient value iteration algorithms, known as Kleene's fixpoint

iteration. A classical way to do this is to use widening/narrowing operators [CC91]. 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 fixpoint, but not always the least fixpoint 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 [GMP02].

Policy iteration algorithms were introduced by Howard [How60] 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). Classical policy iteration may be thought of as a generalization to monotone non-differentiable convex functions of Newton's algorithm to compute the fixpoint of a function. We refer the reader to [Put94] for a detailed presentation of policy iteration algorithms for stochastic control. Policy iteration is known to be experimentally efficient, although its complexity is still not well understood theoretically. 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 at least to [HK66], but restrictive assumptions were made on transition probabilities to guarantee the convergence, so that deterministic games could not be solved along these lines. General versions of the policy iteration algorithm for games have been designed recently [GG98, CTGG99, CT01], exploiting precise results on the structure of the fixpoint set of dynamic programming operators associated to optimal control problems. In Section 2, we present a new version of the 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 fixpoint that are numerical vectors or functions, fixpoints can be elements of an abstract lattice. This new generality allows us to handle lattices which are useful in static analysis. In our context, we avoid cycling by computing at each step the least fixpoint corresponding to the current policy. The main idea of the proof is that the map which assigns to a monotone map its least fixpoint is in some weak sense a morphism with respect to the inf-law, see Theorem 1.

Other fixpoint acceleration techniques have been proposed in the litterature. There are mainly three types of fixpoint 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 [Bou93], or for interprocedural analysis [Bou92]. 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 analyses 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 (using BDDs, or in static analyzers by abstract interpretation, using binary decision graphs, see [Mau99]) small, by a combination of breadth-first and depth-first stategies, as in [RS99]. For boolean equations, some authors have designed specific representations which allow for relatively fast least fixpoint algorithms. For instance, [KL03] uses Bekić-Leszczyloiwski theorem. In strictness analysis, representation of boolean functions by "frontiers" has been widely used, see for instance [Hun91] and [CP85]. 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, [CC91], 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) [HE02], [FS98]. In fact, this is much like the static partitioning technique some of the authors use in [PGM03]. Related algorithms can be found in [Dam01], [O'K87] and [CH92]. We have not been able to compare these techniques with our algorithm yet.

The results of the present paper were announced in [Cos03].

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 fixpoint iteration, with widening and narrowing refinements (see [CC91]).

Let $(\mathcal{L}, \leq)$ be a complete lattice. We write $\perp$ for its lowest element, $\top$ for its greatest, $\cup$ 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 computation of the least fixpoint of f can be done using the following (maybe countable) iteration sequence:

$$\begin{array}{rcl} x_0 & = & \perp \\ x_1 & = & x_0 \cup f(x_0) \\ \dots & & \\ x_{n+1} & = & x_n \cup f(x_n) \\ \dots & & \end{array}$$

Then the least fixpoint (lfp) may be reached as one of these x_n or as the supremum of all the x_n (we say that the iteration sequence converges to the lfp of f). Of course, this is unefficient, and may even be uncomputable, 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 [CC91] for a good survey. As we will only show examples on the interval lattice, we will not recall the general elements of the theory. Widening operators are binary operators ∇ on $\mathcal{L}$ that ensure at least that any iteration sequence of the form:

$$\begin{array}{rcl} x_0 & = & \perp \\ x_1 & = & x_0 \cup f(x_0) \\ \dots & & \\ x_{k+1} & = & x_k \cup f(x_k) \\ x_{k+2} & = & x_{k+1} \nabla f(x_{k+1}) \\ \dots & & \\ x_{n+1} & = & x_n \nabla f(x_n) \\ \dots & & \end{array}$$

converges to a post-fixpoint of f (i.e. a point x such that $x \geq f(x)$), in a finite time, i.e. the iteration sequence above is eventually constant, say at iteration m on. Index k is in general a parameter of the least fixpoint solver. The bigger k is the more precise it can be, but at the expense of time. In the sequel, we choose $k = 10$.

Narrowing operators are binary operators Δ on $\mathcal{L}$ that ensure at least that any iteration sequence starting from iteration m above:

$$\begin{array}{rcl} x_{m+1} & = & x_m \Delta f(x_m) \\ \dots & & \\ x_{l+1} & = & x_l \Delta f(x_l) \\ \dots & & \end{array}$$

is eventually constant, equal to a fixpoint of f, but not necessarily the least fixpoint.

Consider first the example of Figure 1. The corresponding semantic equations in the lattice of intervals are given in Figure 2. The functional f for which we want a fixpoint of, is the right term of this set of equations. The standard Kleene iteration sequence is eventually constant after 100 iterations, reaching the least fixpoint described in Figure 3. Now, using the classical (see [CC91] again) widening and narrowing operators that we are using here, as a reference for comparison for our policy iteration method, are:

$$[a, b] \nabla [c, d] = [e, f]$$

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

$$[a, b] \Delta [c, d] = [e, f]$$

with $e = \begin{cases} c & \text{if } a = -\infty \\ a & \text{otherwise} \end{cases}$ and $f = \begin{cases} d & \text{if } b = \infty \\ b & \text{otherwise} \end{cases}$

The iteration sequence using widenings and narrowings is given in Figure 4. It takes 12 iterations because we chose $k = 10$, and it reaches the least fixpoint of f.

3 Policy iteration algorithm for self-maps of complete lattices

We equip the set of self-maps of a complete lattice $\mathcal{L}$ with the product ordering: thus, $f \leq g$ if $f(x) \leq f(y)$ holds for all $x \in \mathcal{L}$. In order to compute a fixpoint of f, it will be convenient to assume that f is effectively given as an infimum of a finite set of "simpler" maps. We wish to obtain a fixpoint of f from the fixpoints of these maps. To this end, the following notion will be useful.

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

Setting $f \stackrel{\text{def}}{=} \inf \mathcal{G}$, we see that the set $\mathcal{G}$ has a lower selection if and only if for all $x \in \mathcal{L}$, we have $f(x) = g(x)$ for some $g \in \mathcal{G}$. This selection property originates from optimal control: the dynamic programming operator corresponding to an optimal control problem with state space $\{1, \ldots, n\}$ can be naturally written as an infimum of a set of affine maps, every affine maps corresponding to a feedback strategy, and in this context, the existence of a selection is guaranteed by standard assumptions.

Since $\mathcal{L}$ is a complete lattice, Tarski's fixpoint theorem shows that every monotone self-map f of $\mathcal{L}$ has a least fixpoint. We denote it by f^-.

The essence of the policy iteration algorithm in complete lattices is contained in the following abstract result, which shows that the least fixpoint of a monotone map written as an infimum of a set having a lower selection can be determined from the least fixpoints of the maps in this set. This result is inspired by a related result, proved in [CTGG01] for monotone self-maps of $\mathbb{R}^n$ that are nonexpansive in the sup-norm (see also the last chapter of [CT01]).

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,*

$$f^- = \inf_{g \in \mathcal{G}} g^- \quad .$$

Proof. By Tarski's theorem, the least fixpoint 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 $\inf_{g \in \mathcal{G}} g^- \leq f^-$. $\square$

This result motivates the following policy iteration algorithm. The input of the algorithm consists of a finite set $\mathcal{G}$ of self-maps of $\mathcal{L}$. We assume that we have an oracle returning g^-, for every $g \in \mathcal{G}$. The output is a fixpoint of $f \stackrel{\text{def}}{=} \inf \mathcal{G}$.

Algorithm 1 (Policy iteration in complete lattices).

1. Initialisation. *Select any map $g_1 \in \mathcal{G}$. Set $k = 1$.*
2. Value determination. *Compute g_k^-.*
3. *Compute $f(g_k^-)$.*
4. *If $f(g_k^-) = g_k^-$ return g_k^-.*
5. Policy improvement. *Take g_{k+1} such that $f(g_k^-) = g_{k+1}(g_k^-)$. Increment k and goto Step 2.*

In the applications that we shall consider, the cardinality of the whole set $\mathcal{G}$ will be huge, but $\mathcal{L}$ will be a cartesian product of relatively simple lattices, like the lattice of intervals, and every coordinate of f will be represented efficiently by a certain term in a grammar. The collection of these terms will be the actual input of the algorithm. Additionnally, an efficient oracle taking $x \in \mathcal{L}$ and returning a map $h \in \mathcal{G}$ such that $f(x) = h(x)$ will be available.

We call *height* of a subset $\mathcal{X} \subset \mathcal{L}$ the maximal cardinality of a chain of elements of $\mathcal{X}$.

Theorem 2 (Convergence of Policy Iteration in complete lattices). *The number of iterations of Algorithm 1 is bounded by the height of $\{g^- \mid g \in \mathcal{G}\}$, and a fortiori, by the cardinality of $\mathcal{G}$.*

Proof. When the policy is improved at step k, we have $f(g_k^-) < g_k^-$, and we choose g_{k+1} such that $g_{k+1}(g_k^-) = f(g_k^-)$, so that $g_{k+1}(g_k^-) \leq g_k^-$. By Tarski's fixpoint 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 fixpoint of g_{k+1}. Thus, the sequence $g_1^-, g_2^-, \ldots$ produced by the algorithm is strictly decreasing, which implies that the number of iterations is bounded by the height of $\{g^- \mid g \in \mathcal{G}\}$. $\square$

Remark 1. Although the minimal fixpoints of a map $g \in \mathcal{G}$ is computed at every intermediate step, the policy iteration algorithm need not return the minimal fixpoint of f.

Remark 2. In some circumstances, the least fixed point g_k^- may be difficult to obtain, but a fixpoint u_k of g_k such that $u_k \leq g_k(u_{k-1})$ may be available. Let us call Algorithm 1' the generalization of Algorithm 1 in which g_k^- is replaced by such a fixpoint u_k. One readily checks that if f commutes with the infimum of a denumerable set, then, the infimum v of the sequence u_k produced by Algorithm 1' is a fixpoint of f.

4 Application to the lattice of intervals in static analysis

In the sequel, we shall consider intervals of $\mathbb{R}$. The set of intervals, $\mathcal{I}(\mathbb{R})$, ordered by inclusion, is a complete lattice.

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).

We do not describe the interprocedural fragment, since this is a completely orthogonal issue to our problem here.

There is a classical [CC91] 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 following 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. Each variable x_i is interpreted as an interval $[-x_i^-; x_i^+]$. $x^{\complement}$ denotes the least interval including the complement in $]-\infty; +\infty[$ of the set x.

$$[\![\texttt{(0) while } c \texttt{ (1) } p \texttt{ (2) ; (3)}]\!] =$$
$$x_i^1 = [\![c]\!] \cap (x_i^0 \cup x_i^2) \ \wedge \ x_i^3 = [\![c]\!]^{\complement} \cap (x_i^0 \cup x_i^2) \ \wedge \ [\![p]\!]$$

$$[\![\texttt{(0) if } c \texttt{ (1) } p_1 \texttt{ (2) else (3) } p_2 \texttt{ (4); (5)}]\!] =$$
$$x_i^1 = x_i^0 \cap [\![c]\!] \ \wedge \ x_i^3 = x_i^0 \cap [\![c]\!]^{\complement} \ \wedge \ x_i^5 = x_i^2 \cup x_i^4 \ \wedge \ [\![p_1]\!] \ \wedge \ [\![p_2]\!]$$

$$[\![\texttt{(0) } x_i \texttt{ = } e \texttt{ (1)}]\!] \ = \ x_i^1 = [\![e]\!]$$

For conditions, we have:

$$[\![\texttt{(0) } x_i \texttt{ = } a \texttt{ (1)}]\!] \ = \ x_i^1 = [a, a]$$

$$[\![\texttt{(0) } x_i \texttt{ = } x_j \texttt{ (1)}]\!] \ = \ x_i^1 = x_i^0 \ \cap \ x_j^0 \ \wedge \ x_j^1 = x_i^0 \ \cap \ x_j^0$$

$$[\![\texttt{(0) } x_i \texttt{ < } a \texttt{ (1)}]\!] \ = \ x_i^1 = x_i^0 \cap \,]-\infty; a[$$

$$[\![\texttt{(0) } x_i \texttt{ < } x_j \texttt{ (1)}]\!] \ = \ x_i^1 = x_i^0 \cap \,]-\infty; x_j^{0+} - 1] \ \wedge \ x_j^1 = x_j^0 \ \cap \ [x_i^{0-} + 1; +\infty[$$

4.2 Min-max functions and the selection principle

For an interval $I = [-a, b]$, we set $\uparrow I \stackrel{\text{def}}{=} [-a, \infty[$ and $\downarrow I \stackrel{\text{def}}{=}]-\infty, b]$. And if $J = [-c, d]$, we also define $l(I, J) = I$, $r(I, J) = J$, $m(I, J) = [-a, d]$ and $m^{\text{op}}(I, J) = [-c, b]$. These four operators will define the four possible policies on intervals, as shown by Proposition 2.

J. Gunawardena [Gun94] introduced the class of min-max functions from $\mathbb{R}^n$ to $\mathbb{R}^n$, after a work of Olsder [Ols91] concerning a subclass of min-max functions. Min-max functions are precisely dynamic programming operators of deterministic zero-sum repeated games with perfect information, state space $\{1, \ldots, n\}$, and finite action space. We define here a similar class, for maps defined on $(\mathcal{I}(\mathbb{R}))^n$, which could be extended, but corresponds here precisely to what the abstract semantic equations of the previous section give us.

Definition 2. *A min-max function of intervals, $(\mathcal{I}(\mathbb{R}))^n \to (\mathcal{I}(\mathbb{R}))^p$, is a map whose coordinates are terms of grammar G:*

$$\begin{array}{rcllll}
CSTE & ::= & [-a, b] & & & \\
VAR & ::= & x_i & & & \\
EXPR & ::= & CSTE & | & VAR & | \\
 & & EXPR + EXPR & | & EXPR * EXPR & | \\
 & & EXPR / EXPR & | & EXPR - EXPR & \\
TEST & ::= & \uparrow EXPR \cap EXPR & | & \downarrow EXPR \cap EXPR & | \\
 & & CSTE \cap EXPR & & & \\
G & ::= & EXPR & | & TEST & | \\
 & & G \cup G & & &
\end{array}$$

for all $1 \leq i, j \leq n$, and where $a, b \in \mathbb{Z}$, and $x_i \in Var$ takes its value in the interval $[-x_i^-, x_i^+] \in \mathcal{I}(\mathbb{R})$.

We write $\mathcal{M}$ for the set of such functions. Non-terminals $CSTE$, VAR, $EXPR$ and $TEST$ do correspond to the semantics of constants, variables, arithmetic expressions, and (simple) tests.

Let $G_\cup$ be the grammar given by the same production rules as G except that we cannot produce terms with $\cap$.

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

We write $\mathcal{M}_\cup$ 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:

$$x_i \cap x_j = l(x_i, x_j) \cap r(x_i, x_j) \cap m(x_i, x_j) \cap m^{\text{op}}(x_i, x_j) \tag{1}$$

To a min-max function of intervals $f \in \mathcal{M}$, we associate a family $\Pi(f)$ of functions of $\mathcal{M}_\cup$ obtained in the following manner: we replace each occurrence of a term $x_i \cap x_j$ by $l(x_i, x_j)$, $r(x_i, x_j)$, $m(x_i, x_j)$ or $m^{\mathrm{op}}(x_i, x_j)$. Such a choice is a *policy*.

Proposition 1. *We have*

$$f = \bigcap_{h \in \Pi(f)} h \tag{2}$$

and the selection principle is satisfied.

In Equation (2), we denote by intersection the inf law with respect of the product ordering of the maps in $\mathcal{M}$.
Proof. This follows readily from the fact that the intersection is attained by one of the four terms in Equation (1): we have either $x_i \cap x_j = l(x_i, x_j) = x_1$ or $x_i \cap x_j = r(x_i, x_j) = x_j$ or $x_i \cap x_j = m(x_i, x_j)$ or $x_i \cap x_j = m^{\mathrm{op}}(x_i, x_j)$. This is crucial to apply the selection principle mentioned earlier.

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 modelling 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 $x_1 \cap x_2$ in Equation (1), when the fixpoint is reached for x_1 and x_2. The choice of the initial policy may be crucial, since some choices of the initial policy may lead eventually to a fixpoint which is not minimal. We will study such problems in detail elsewhere.

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 favour as first guess an easily computable system of equations.

This choice is static, and does not use the values of the variables known when getting to an intersection. Another choice that looks attractive, consists, when we have to choose between two variable quantities, in evaluating these quantities when encountering the intersection node, and in taking the quantity which value is the best, depending whether we want the lower or upper bound. However, the policy that looks the best at a given fixpoint iteration may not be the policy that leads to the least fixpoint. It looks sensible to make a few iterations on the system with no policy before choosing the initial policy, using the evaluation of the terms. We will briefly show an example of this dynamic policy change in Section 4.4, but most of the discussion here is left for future work.

We then solve the fixpoint equation of the reduced equations, using possibly specific algorithms (Ford-Bellman if the reduced equations are very simple, another policy iteration algorithm for the $\cup$ part, specific numerical solvers etc.). 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 fixpoint of the complete system of equations is not reached, and compute the fixpoint of the new equations, until we find a fixpoint of the complete system of equations.

In the following examples, we count the number of policies the algorithm had to consider in order to reach the least fixpoint, and the cost of solving the remaining equations, without intersection. But the overall speedup of policy iteration algorithms should be improved if we use one of the possibly very efficient algorithm for those more specific equations.

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, discussed in the previous section. We compare the policy iteration algorithm with Kleene's iteration sequence with widenings and narrowings (the very classical one of [CC91]), called algorithm A here. For these two algorithms, we compare the accuracy of the results, and the number of fixpoint 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 B) for each policy. Algorithm B is exactly the same solver as algorithm A, but used on a smaller class of functions, for one policy. We decided to widen intervals, in both cases, only after ten standard Kleene iterations. This choice is completely conventional, and in most examples below, one could argue that an analyzer with only two standard Kleene iterations would have found the right result. 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.

A simple typical (integer) loop would be the one of Figure 1. The equations generated by the analyzer are the ones of Figure 2. The control points in the examples to follow are indicated as comments in the C code.

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

Figure 1: A simple integer loop

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

Figure 2: Semantic equations

The original policy is m^{op} in equation 2 in Figure 2 (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, since we always choose a min or max that gives information (∞ as a max, or $-\infty$ as a min do not give any information). This is actually the right policy on the spot, and we find in one iteration, the correct result (the least fixpoint), see Figure 3. 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.

$$
\begin{array}{lcr}
x_1 & = & [0,0] \\
x_2 & = & [0,99] \\
x_3 & = & [1,100] \\
x_4 & = & [100,100]
\end{array}
\qquad
\begin{array}{lcr}
x_1^1 & = & [0,0] \\
x_2^1 & = & [0,0] \\
x_3^1 & = & [1,1] \\
x_4^1 & = & \bot \\
\hline
x_1^2 & = & [0,0] \\
x_2^2 & = & [0,1] \\
x_3^2 & = & [1,2] \\
x_4^2 & = & \bot \\
\hline
\ldots & &
\end{array}
$$

Figure 3: The first and only iteration by algorithm B leads to the least fixpoint (loop of Figure 1)

$$
\begin{array}{lcr}
x_1^9 & = & [0,0] \\
x_2^9 & = & [0,8] \\
x_3^9 & = & [1,9] \\
x_4^9 & = & \bot \\
\hline
(widening) & & \\
x_1^{10} & = & [0,0] \\
x_2^{10} & = & [0,\infty[\\
x_3^{10} & = & [1,\infty[\\
x_4^{10} & = & [100,\infty[\\
\hline
(narrowing) & & \\
x_1^{11} & = & [0,0] \\
x_2^{11} & = & [0,99[\\
x_3^{11} & = & [1,100] \\
x_4^{11} & = & [100,100]
\end{array}
$$

Figure 4: Iterations of algorithm A (loop of Figure 1)

This is to be compared with the 12 iterations of algorithm A, of Figure 4.

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

Figure 5: A more complex loop

$$
\begin{array}{lcr}
(i_1, j_1) & = & ([1,1], \top) \\
(i_2, j_2) & = & (i_1, [10,10]) \\
(i_3, j_3) & = & (]-oo, max(j_2, j_5)] \cap (i_2 \cup i_5), \\
 & & [min(i_2, i_5), +oo[\cap(j_2 \cup j_5)) \\
(i_4, j_4) & = & (i_3 + [2,2], j_3]) \\
(i_5, j_5) & = & (i_4, [-1,-1] + j_4) \\
(i_6, j_6) & = & ([min(j_2, j_5) + 1, +oo[\cap(i_2 \cup i_5), \\
 & &]-oo, max(i_2, i_5) - 1] \cap (j_2 \cup j_5))
\end{array}
$$

Figure 6: Its semantic equations

The analysis of the program shown on Figure 5 leads to an actual policy improvement. 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 B with this policy, finds the values of Figure 7. 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 B for this policy, one finds the least fixpoint of the system of semantic equations, see Figure 8.

$$
\begin{array}{lcr}
(i_1^1, j_1^1) & = & ([1,1], \top) \\
(i_2^1, j_2^1) & = & ([1,1], [10,10]) \\
(i_3^1, j_3^1) & = & ([1,10], [1,10]) \\
(i_4^1, j_4^1) & = & ([3,12], [1,10]) \\
(i_5^1, j_5^1) & = & ([3,12], [0,9]) \\
(i_6^1, j_6^1) & = & ([1,12], [0,11])
\end{array}
$$

Figure 7: Result of the initial policy for the loop of Figure 5

$$
\begin{array}{lcr}
(i_1^2, j_1^2) & = & ([1,1], \top) \\
(i_2^2, j_2^2) & = & ([1,1], [10,10]) \\
(i_3^2, j_3^2) & = & ([1,10], [1,10]) \\
(i_4^2, j_4^2) & = & ([3,12], [1,10]) \\
(i_5^2, j_5^2) & = & ([3,12], [0,9]) \\
(i_6^2, j_6^2) & = & ([1,12], [0,10])
\end{array}
$$

Figure 8: Result with the second policy for the loop of Figure 5

Algorithm A takes ten iterations to reach the same result.

We now consider the program of Figure 9. It contains five loops, and their fixpoint are computed sequentially : the code after a loop is ignored as long as the fixpoint is not reached for the given loop.

The algorithm using policy iteration uses only one widening of algorithm B (i.e. only on one of the five loops). It actually converges to the least fixpoint,

```
int main(int n) {
  int i, j, k, l, m;
  i = 0;              // 1
  j = 100;            // 2
  k = 1000;           // 3
  l = 10000;          // 4
  m = 100000;         // 5
  while (i < 1000)    // 6
    i = i+1;          // 7
                      // 8
  while (j  < 1000)   // 9
    j = j+k;          // 10
                      // 11
  while (k > 100)     // 12
    k = k-j;          // 13
                      // 14
  while (l > 1000)    // 15
    l = l+k;          // 16
                      // 17
  while (m > 1)       // 18
    m = m-l;          // 19
                      // 20
}
```

Figure 9: Multiple loops

using 13 value iterations and 0 policy iteration. Algorithm A needs 43 iterations to converge to the same result. For lack of space, we do not reproduce the corresponding semantic equations nor the complete value of the lfp for this example, but only what is found for line 20:

$$(i_{20}, j_{20}, k_{20}, l_{20}, m_{20}) = (1000, [1000, 1999], [-1898, 100], [-897, 1000], [-998, 1])$$

Finally we look at a typical bad case example for our policy iteration method, where the number of iterations needed to reach the least fixpoint is at least equal when using the policy and the value iteration algorithm, and where the end result for the policy iteration method is a fixpoint, but not the least fixpoint of the system of semantic equations.

Consider the program of Figure 10, that we want to analyze. The original

```
int x;
x=0;              // 0
while (x<100) {   // 1
  x=-1-x;         // 2
}                 // 3
```

Figure 10: A bad case

$$\begin{array}{lcl} x_0 & = & [0,0] \\ x_1 & = &]-\infty, 99] \cap (x_0 \cup x_2) \\ x_2 & = & [-1,-1] - x_1 \\ x_3 & = & [100, \infty[\cap (x_0 \cup x_2) \end{array}$$

Figure 11: Its semantic equations

policy our algorithm chooses is m^{op} for equation 2, variable x and m for equation 4, variable x. It does so because of the $-\infty$ as the minimum of the left argument of $\cap$ in equation 2, and because of the ∞ as the maximum of the right argument of $\cap$ in equation 4. Then we find in two iterations of algorithm B for this policy the fixpoint of Figure 12, which is not the least fixpoint, and thus is rather imprecise.

By algorithm A, we find the least fixpoint (see Figure 13) in only two iterations.

$$\begin{array}{lcl} x_0 & = & [0,0] \\ x_1 & = & [-100,99] \\ x_2 & = & [-100,99] \\ x_3 & = & \bot \end{array}$$

Figure 12: The fixpoint found by the policy iteration algorithm, for loop Figure 10

$$\begin{array}{lcl} x_0 & = & [0,0] \\ x_1 & = & [-1,0] \\ x_2 & = & [-1,0] \\ x_3 & = & \bot \end{array}$$

Figure 13: The least fixpoint, found by algorithm A

If we used the dynamic policy iteration mentioned in Section 4.3, then we would have found, after just one iteration of the functional, that the correct policy for variable x in equation 2 is r, and reached the least fixpoint. This will be developped elsewhere.

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 (such as the dynamic policy iteration mechanism, and other ideas).

One of our aims is to generalize the policy iteration algorithm to more complex lattices of properties, such as the one of octagons (see [Min01]). We would like also to apply this technique to symbolic lattices (using techniques to transfer numeric lattices, see for instance [Ven02]).

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 $G_\cup$, using a dual method, 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

[Bou92] F. Bourdoncle. Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4):407–435, 1992.

[Bou93] F. Bourdoncle. Efficient chaotic iteration strategies with widenings. Number 735, pages 128–141. Lecture Notes in Computer Science, Springer-Verlag, 1993.

[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*, pages 238–252, 1977.

[CC91] 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.

[CH92] 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.

[Cos03] 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.

[CP85] 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.

[CT01] J. Cochet-Terrasson. *Algorithmes d'itération sur les politiques pour les applications monotones contractantes.* Thèse, spécialité mathématiques et automatique, École des Mines, Dec. 2001.

[CTGG99] 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.

[CTGG01] J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. Policy iteration algorithms for monotone nonexpansive maps. Draft, 2001.

[Dam01] D. Damian. Time stamps for fixed-point approximation, 2001.

[FS98] C. Fecht and H. Seidl. Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems, 1998.

[GG98] S. Gaubert and J. Gunawardena. The duality theorem for min-max functions. *C. R. Acad. Sci. Paris.*, 326, Série I:43–48, 1998.

[GMP02] 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.

[Gra90] P. Granger. *Analyse de congruences.* PhD thesis, Ecole Polytechnique, 1990.

[Gra91] 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.

[Gun94] J. Gunawardena. Min-max functions. *Discrete Event Dynamic Systems*, 4:377–406, 1994.

[HE02] Kwangkeun Yi Hyunjun Eo. An improved differential fixpoint iteration method for program analysis. November 2002.

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Sci.*, 12:359–370, 1966.

[How60] R. Howard. *Dynamic Programming and Markov Processes.* Wiley, 1960.

[Hun91] L. S. Hunt. *Abstract Interpretation of Functional Languages: From Theory to Practice.* Ph.D. thesis, Department of Computing, Imperial College, London, UK, 1991.

[Kar76] M. Karr. Affine relationships between variables of a program. *Acta Informatica*, (6):133–151, 1976.

[KL03] 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.

[Mau99] 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.

[Min01] A. Miné. The octagon abstract domain in analysis, slicing and transformation. pages 310–319, October 2001.

[O'K87] 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.

[Ols91] G. J. Olsder. Eigenvalues of dynamic max-min systems. *Discrete Event Dyn. Syst.*, 1(2):177–207, 1991.

[PGM03] S. Putot, E. Goubault, and M. Martel. Static analysis-based validation of floating-point computations. Springer-Verlag, 2003.

[PH] P. and N. Halbwachs. Discovery of linear restraints among variables of a program.

[Put94] 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.

[RS99] 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.

[Ven02] 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.