Widening with Thresholds for Programs with Complex Control Graphs

Lies Lakhdar-Chaouch, Bertrand Jeannet, Alain Girault

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Widening with Thresholds for Programs with Complex Control Graphs

Lies Lakhdar-Chaouch — Bertrand Jeannet — Alain Girault

N° 7673

Juillet 2011

Domaine Algorithmique, programmation, logiciels et architectures

ISSN 0249-6399 ISRN INRIA/RR--7673--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Widening with Thresholds for Programs with Complex Control Graphs

Lies Lakhdar-Chaouch , Bertrand Jeannet , Alain Girault

Domaine :
Équipe-Projet POP-ART

Rapport de recherche n° 7673 — Juillet 2011 — 17 pages

Abstract: The precision of an analysis based on abstract interpretation does not only depend on the expressiveness of the abstract domain, but also on the way fixpoint equations are solved: exact solving is often not possible. The traditional solution is to solve iteratively abstract fixpoint equations, using extrapolation with a widening operator to make the iterations converge. Unfortunately, the extrapolation too often loses crucial information for the analysis goal.

A classical technique for improving the precision is "widening with thresholds", which bounds the extrapolation. Its benefit strongly depends on the choice of relevant thresholds. In this paper we propose a semantic-based technique for automatically inferring such thresholds, which applies to any control graph, be it intraprocedural, interprocedural or concurrent, without specific assumptions on the abstract domain. Despite its technical simplicity, our technique is able to infer the relevant thresholds in many practical cases.

Key-words: Abstract Interpretation, Numerical and Symbolic Abstract Domains, Convex Polyhedra, Semantic Equation Solving, Widening

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Elargissement avec seuils pour les programmes à structure de contrôle complexe

Résumé : La précision d'une analyse fondée sur l'interprétation abstraite dépend no seulement de l'expressivité du domaine abstrait, mais aussi de la façon dont les équations abstraites sont résolues: la solution optimale n'est en effet pas toujors calculable. La technique traditionnelle est de résoudre itérativement les équations de point-fixe abstraites, en effectuant des extrapolations à l'aide d'un opérateur d'élargissement pour faire converger les itérations. Malheureusement, ces extrapolations induisent fréquemment la perte d'informations cruciales pour l'objectif de l'analyse.

Une technique classique pour améliorer la précision est « l'élargissement avec seuil », qui borne l'extrapolation. Son efficacité dépend fortement du choix de seuils pertinents. Nous proposons ici une technique de nature sémantique pour inférer automatiquement des seuils pertinents, qui s'applique à n"importe quel graphe de contrôle, qu'il soit intraprocédural, interprocédural ou concurrent, sans hypothèse spécifique sur le domaine abstrait. malgré sa simplicité technique, cette technique infère les seuils pertinents dans beaucoup de cas pratiques.

Mots-clés : Interprétation abstraite, Domaine abstraits numériques et symboliques, Polyèdres convexes, Résolution d'équations sémantiques, Élargissement

1 Introduction and Related Work

Many static analysis problems boil down to the computation of the least solution of a fixpoint equation $X = F(X), X \in C$ where C is a domain of concrete properties, and F a function derived from the semantics of the analyzed program. Abstract Interpretation [1] provides a theoretical framework for reducing this problem to the solving of a simpler equation in a domain A of *abstract properties*:

$$Y = G(Y), Y \in A \tag{1}$$

Having performed this *static approximation*, one is left with the problem of solving Eqn. (1). The paper focuses on this problem. It considers the traditional iterative solving technique with widening and narrowing, and focuses more specifically on the widening with thresholds technique. We first review existing techniques before presenting our approach.

Exact equation solving. Some techniques have been recently proposed for solving directly Eqn. (1) in the case where concrete properties are invariants on numerical variables. In [2, 3] classes of equations on intervals are identified, for which the least solution can be computed in polynomial time. *Policy iteration* methods, inspired by game theory, solve Eqn. (1) by solving a succession of simpler equations $Y = G_\pi(Y)$ indexed by a *policy* π. They have been applied for instance to intervals [4, 5] and template polyhedra [6, 7]. However, such approaches are currently restricted to domains that infer lower/upper bounds on a fixed set of numerical expressions. This excludes numerical abstract domains like convex polyhedra [8] or symbolic abstract domains for sets of words [9] or terms [10]. Hence, they do not make obsolete the classical iterative method described next.

Approximate equation solving by widening/narrowing. Under the classical hypothesis the sequence $Y_0 = \bot, Y_{n+1} = G(Y_n)$ converges to $lfp(G)$. However, this method is effective only if A does not contain infinite ascending sequences, which is not the case in all the abstract lattices mentioned above. Abstract Interpretation proposes to extrapolate the limit by using a *widening* operator $\nabla : A \times A \to A$. One computes the ascending sequence

$$Y_0 = \bot,\ Y_{n+1} = Y_n \nabla G(Y_n) \tag{2}$$

which converges within a bounded number of iterations to a post-fixpoint $Y_\infty \sqsupseteq lfp(G)$, see Fig. 1. The approximations induced by widening can be partially recovered by performing a few descending iterations defined by the sequence

$$Z_0 = Y_\infty,\ Z_{n+1} = G(Z_n) \tag{3}$$

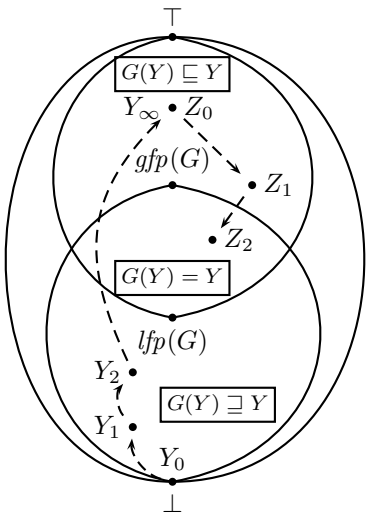


Figure 1: Kleene iteration with widening and narrowing

This is the most common instance of the concept of *narrowing* (see [11]). In practice, descending sequences propagates guards in the Control Flow Graph (CFG) of the program.

The use of widening adds *dynamic approximations* to the *static approximations* induced by the choice of the abstract domain. Although it is shown in [11] that abstract domains with infinitely ascending sequences can discover properties that simpler abstract domains cannot infer, these dynamic approximations often raise accuracy issues. For instance, for many numerical abstract domains (like intervals [12], octagons [13] and convex polyhedra [8]) the "standard" widening consists in keeping in the result $R = P \nabla Q$ the numerical constraints of P that are still satisfied by Q. Despite this clear geometrical intuition, in the context of program analysis it is

difficult to predict and to keep under control the loss of information it induces. More generally, to the best of our knowledge, no widening operator is monotonic and widening is ultimately a heuristic method. Moreover, as we will show in Section 3, narrowing often fails to recover important information lost by widening, even on simple examples. In the extreme case where the function G is *extensive* (*i.e.,* $\forall Y \in A, Y \sqsubseteq G(Y)$), narrowing has no effect at all.

Techniques for controlling dynamic approximations. Several techniques have been designed to address the widening problem. One approach is to propose improved widening operators, like [14, 15, 16]. Other approaches are more global. For instance, *abstract acceleration* aims at computing precisely with a single formula the effect of "accelerable" cycles in the CFG [17], and relies on widening for more complex cycles. The *guided static analysis* technique of [18] alternates ascending and descending sequences on an increasingly larger part of the system of equations. This method applies to any abstract domain and it improves the accuracy of the analysis in many cases, but it relies ultimately on the effectiveness of narrowing, which is far from being guaranteed (see Section 3).

Widening with thresholds. Among local techniques, *widening up-to* or *widening with thresholds* attempts to bound the extrapolation performed by the standard widening ∇ operator [8, 19]. The idea is to parameterize ∇ with a finite set $\mathcal{C}$ of *threshold constraints*, and to keep in the result $R = P \nabla_{\mathcal{C}} Q$ those constraints $c \in \mathcal{C}$ that are still satisfied by Q: $P \nabla_{\mathcal{C}} Q = (P \nabla Q) \sqcap \{c \in \mathcal{C} \mid Q \models c\}$. In practice, one extrapolates up to some threshold; in the next iteration, either the threshold is still satisfied and the result is better than with the standard widening, or it is violated and one extrapolates up to the remaining thresholds.

On simple examples with a relevant choice of thresholds, widening with thresholds does not improve upon standard widening and narrowing. However, in the cases where narrowing is not effective, widening with thresholds may behave much better: similarly to abstract acceleration techniques, widening with thresholds prevents from going too high in the lattice of properties (see Fig. 1) and from propagating inaccurate invariants in the CFG of the program, which cannot be strengthened later by narrowing. However, the benefit provided by widening with thresholds fully depends on the choice of the thresholds.

Our contribution: thresholds inference. This paper develops a semantic-based technique to infer automatically relevant thresholds, by propagating constraints in the CFG of the program in an adequate way.

After giving in Section 2 some preliminaries about iterative equation solving with widening and narrowing, Section 3 applies it to several small examples to illustrate the strengths and weaknesses of narrowing. We also analyze carefully the problem of inferring relevant thresholds on these examples and we show that, although widening with thresholds is defined as a local improvement of the standard widening, the relevant thresholds depend on more global properties of the program. After having progressively given the hints behind our inference technique, Section 4 formalizes it in a generic way, illustrates it on the running examples, and discusses its application to more complex abstract domains.

Section 5 evaluates it on a number of example programs and compares it w.r.t. both efficiency and precision to guided static analysis [18] and policy iteration [4]. This comparison shows that our techniques is almost always as or more precise than the mentioned techniques for all the examples we tested. w.r.t. efficiency, our technique is slightly slower for sequential, intraprocedural programs. We also discuss the less favorable cases of interprocedural and/or concurrent programs.

A strength of our approach is that it can easily be combined with other approaches aiming at improving dynamic approximations, in particular abstract acceleration [17], which may be more precise than our inference technique *when it is applicable*.

2 Preliminaries and Notations

The equations to be solved. We assume a static analysis problem formalized as an equation system

$$X^{(k)} = F^{(k)}(X) \qquad X = (X^{(1)}, \ldots, X^{(K)}) \in C^K \tag{4}$$

where $X^{(k)} \in C$ is typically the concrete property associated with a node of the CFG of the program. The technical assumptions are that $(C, \subseteq)$ is a complete lattice ordered by logical implication, and the functions $F^{(k)}$ are continuous. In the case of an imperative program without procedure call, we have an intraprocedural CFG and Eqn. (4) can be rewritten into a system $X^{(k)} = \bigcup_{k' \rightsquigarrow k} F^{k' \rightsquigarrow k}(X^{(k')})$ where $F^{k' \rightsquigarrow k}$ is the semantic function associated to the CFG edge $k' \rightsquigarrow k$. In relational interprocedural analysis however, the procedure return operation may combine both the information at the call-site in the caller and at the exit-site of the callee, see for instance [20, 21].

We assume an abstract domain $(A, \sqsubseteq)$ connected to C with a monotone concretization function $\gamma : A \to C$, yielding the system of equations

$$Y^{(k)} = G^{(k)}(Y) \qquad Y = (Y^{(1)}, \ldots, Y^{(K)}) \in A^K \tag{5}$$

derived from Eqn. (4) in the sense that $\forall k : \gamma \circ G^{(k)}(Y) \supseteq F(\gamma(Y^{(1)}) \ldots \gamma(Y^{(K)}))$. The functions $G^{(k)}$ are assumed to be monotonic and A is assumed to be a lattice equipped with a *widening operator* $\nabla : A \times A \to A$ satisfying the technical assumptions described in [11].

In all the examples of this paper, the static analysis problem is the computation of reachable values of the numerical variables of a program. A will be the convex polyhedra domain, equipped with its standard widening operator, see [8] for the full definition.

Solving by Kleene chaotic iteration. In all the examples, in order to solve Eqn. (5), we apply the technique of Bourdoncle [22], *i.e.,* chaotic iterations with widening. That is, we follow the iteration order $1 \ldots K$ and we apply widening to the subset W of widening nodes as follows:

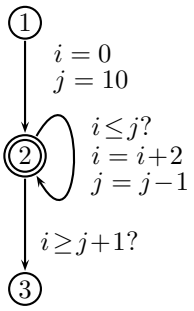
$$Y_0^{(k)} = \bot \qquad Y_{n+1}^{(k)} = \begin{cases} Y_n^{(k)} \nabla Y' & \text{if } k \in W \\ Y' & \text{otherwise} \end{cases} \quad \text{where } Y' = G^{(k)}(Y_{n+1}^{(0)} \ldots Y_{n+1}^{(k-1)}, Y_n^{(k)} \ldots Y_n^{(K)}) \tag{6}$$

The subset W is such that any dependency cycle in Eqn. (5) contains a node in W. Narrowing by descending iteration (Eqn. (3)) is performed similarly on a partitioned system. In practice, one first decomposes the dependency graph induced by the system of equations (5) into strongly connected components, and then one solves each component using widening and narrowing following a linearized order compatible with the topological order between components.

3 The widening/narrowing approach in practice

This section illustrates the widening/narrowing approach for the analysis of numerical variables of small examples. It points out the limitations of narrowing for recovering the information lost by widening, and gives the intuition about how to infer relevant threshold constraints. We end the section with the rationale for the inference method we propose in Section 4.

```
var i,j:int;
begin
  i=0; j=10;
  while i<=j do
    i = i+2;
    j = j-1;
  done;
end
```



Ascending sequence		
n	$Y_n^{(2)}$	$Y_n^{(3)}$
1	$i{=}0 \wedge j{=}10$	
2	$i{+}2j{=}20 \wedge 0{\leq}i{\leq}2$	
3′	$i{+}2j{=}20 \wedge 0{\leq}i{\leq}4$	
3	$i{+}2j{=}20 \wedge 0{\leq}i$	$i{+}2j{=}20 \wedge 22{\leq}3i$

Descending sequence		
n	$Z_n^{(2)}$	$Z_n^{(3)}$
1	$i{+}2j{=}20 \wedge 0{\leq}3i{\leq}26$	$i{+}2j{=}20 \wedge 22{\leq}3i{\leq}26$

Figure 2: Example: single loop

3.1 Analysis of a simple loop program

Our introductory example is the program depicted on Fig. 2. The double-line around a CFG node indicates a widening node in W. The table on the right details the Kleene iteration with widening and narrowing (descending sequence), starting from $\bot$ at the two node ② and ③. In the steps 1 and 2, the widening operator has no effect. The row indexed by 3' corresponds to the computation of Y' in Eqn. (6). In step 3, we have $Y_3^{(2)} = Y_2^{(2)} \nabla Y_{3'}^{(2)}$ and the effect of widening is the loss of the upper bound on i. One descending step discovers the constraint $i \leq 26/3$, which comes from the postcondition of $Y_3^{(2)}$ by the loop:

$$\exists i,j : \Big(\overbrace{i{+}2j{=}20}^{\text{implied by } Y_3^{(2)}} \wedge \overbrace{i{\leq}j \wedge i'{=}i{+}2 \wedge j'{=}j{-}1}^{\text{loop transition}}\Big) \quad = \quad (i'{=}20{-}2j' \wedge \boxed{i'{\leq}j'{+}3}) \quad \Rightarrow \quad \underbrace{i'{\leq}20{-}2(i'{-}3)}_{=\,3i'\,\leq\,26} \tag{7}$$

This example calls for two important observations:

(1) The invariant $Z^{(3)}$ at point ③ can be rewritten into $i{+}2j = 20 \wedge 8{-}\frac{2}{3} \leq i \leq 8{+}\frac{2}{3}$. This means that $i \leq 26/3$ is the right bound for i at node ② if one abstracts away the arithmetic properties.

(2) If one wants to use widening with thresholds, the guard of the loop $i \leq j$ is not a useful threshold constraint. Starting from step 3, we would obtain

$$\begin{array}{lcll} Y_3^{(2)} & = & Y_2^{(2)} \nabla_{\{i\leq j\}} Y_{3'}^{(2)} & = i{+}2j{=}20 \wedge 0{\leq}i{\leq}j \\ Y_{4'}^{(2)} & = & G(Y_3^{(2)}) & = i{+}2j{=}20 \wedge 0{\leq}i{\leq}j{+}3 \\ Y_4^{(2)} & = & Y_3^{(2)} \nabla_{\{i\leq j\}} Y_{4'}^{(2)} & = i{+}2j{=}20 \wedge 0{\leq}i \end{array}$$

The effect of using this threshold constraint allows us to keep the constraint $i \leq j$ at step 3, but this bound is violated at step 4′ by the postcondition of the loop transition, hence this does not change the final result.

The important point here is that *the important constraint in a simple while loop is the postcondition of the guard of the loop by the loop body*, here $i \leq j{+}3$, see Eqn. (7).

3.2 Four problematic examples

Two non-deterministic loops. The CFG of Fig. 3 is typically the result of the asynchronous parallel product of two threads with a simple loop. It shows the limitation of descending sequences. The ascending sequence converges to $Y^{(2)} = 0 \leq i \wedge 0 \leq j$. The descending sequence

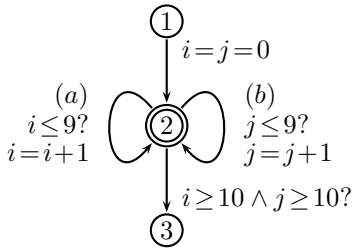


Figure 3: Example: two non-deterministic loops

```
var i:int;
begin
  i=0;
  while true do
    if ? then
      i=i+1;
      if i>=100 then
        i=0;
  done;
end
```

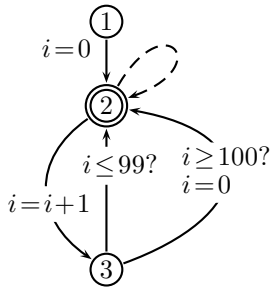
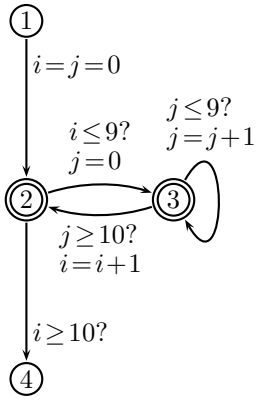


Figure 4: Example: a single loop with break

```
var i,j:int;
begin
 i=0; j=0;
 while i<=9 do
   j=0;
   while j<=9 do
     j=j+1;
   done;
   i=i+1;
 done;
end
```



Ascending sequence		
n	$Y_n^{(2)}$	$Y_n^{(3)}$
1	$i=j=0$	$i=j=0$
$2'$	$i=j=0$	$i=0 \wedge 0\leq j\leq 1$
2	$i=j=0$	$i=0 \wedge 0\leq j$
$3'$	$0\leq i\leq 1 \wedge 10i\leq j$	$0\leq i\leq 9 \wedge 0\leq j$
3	$0\leq i \quad \wedge 10i\leq j$	$0\leq i \quad \wedge 0\leq j$
$4'$	$0\leq i \quad \wedge 0\leq j$	$0\leq i \quad \wedge 0\leq j$
4	$0\leq i$	$0\leq i \quad \wedge 0\leq j$

Descending sequence		
n	$Z_n^{(2)}$	$Z_n^{(3)}$
1	$0\leq i \quad \wedge 0\leq j$	$0\leq i \quad \wedge 0\leq j\leq 10$
2	$j\leq 10i \wedge 0\leq j\leq 10$	$0\leq i \quad \wedge 0\leq j\leq 10$

Figure 5: Example: nested loop

fails to improve it:

$$\begin{array}{rcccccc} Z_1^{(2)} & = & G^{1\rightsquigarrow 2}(Y^{(1)}) & \sqcup & G^{2\rightsquigarrow 2(a)}(Y^{(2)}) & \sqcup & G^{2\rightsquigarrow 2(b)}(Y^{(2)}) \\ & = & \{i=j=0\} & \sqcup & \{1\leq i\leq 10 \wedge 0\leq j\} & \sqcup & \{0\leq i \wedge 0\leq j\leq 10\} \\ & = & & & \{0\leq i \wedge 0\leq j\} & & \end{array}$$

The problem is that, for both variables i and j, there is always one incoming edge in node ② that propagates an invariant without an upper bound on it. As a result, no variable gets an upper bound in the result.

A single loop with break. Another example, inspired by a real controller, is depicted on Fig. 4. The dashed self-loop comes from the non-deterministic test "`?`" modeling an input from the environment. When the "then" branch is not taken, nothing happens in the loop body. It makes the transfer function on node ② extensive: $G^{(2)}(Y) \sqsupseteq G^{2\rightsquigarrow 2}(Y^{(2)}) = Y^{(2)}$. Hence, the descending sequence will never improve the invariant $Y^{(2)} = i\geq 0$ found by the ascending sequence.

Nested loop. The nested loop program of Fig. 5 contains two widening nodes ② and ③ and raises some additional issues. The ascending sequence loses the two constraints $j\leq 10$ (step 2) and $i\leq 10$ (step 3) as expected (it even loses $0\leq j$ at step 4). The descending sequence first recovers $j\leq 10$ at point ③, but then fails to recover $i\leq 10$ at point ②. The problem is similar to the problem with the non-deterministic loops of Fig. 3:

- at point ②, the incoming edge ③ $\rightsquigarrow$ ② is not guarded by $i\leq 9$, and
- at point ③ the self-loop ③ $\rightsquigarrow$ ③ is also not guarded by $i\leq 9$.

Hence, $i\leq 10$ is neither recovered at node ② nor ③. On this example, the guided static analysis of [18] also fails to discover this bound.

This example calls for additional remarks:

(3) Applying the heuristics sketched at the end of Section 3.1 for generating the threshold constraint, considering the postcondition of the guard $i \leq 9$ by the body of the outer loop on i, becomes more difficult: this already implies a fixpoint computation because of the inner loop on j. Even if this inner loop does not modify i, the incrementation of i still depends on its termination.

(4) It shows that, once an important fact is lost and the induced approximation is propagated, it is not always possible to recover it with narrowing.
Hence, if one is able to infer $i \leq 10$ as a threshold constraint at the loop head ②, we need to propagate it also to the inner loop head ③, otherwise widening at node ③ can lose this property preserved at node ②, and propagate this loss back to node ②.

A loop with conditional and guided analysis. Our last example, depicted on Fig. 6, is taken from the "guided static analysis" paper [18]. The loop proceeds in two phases: in the first one, i and j are incremented together until $i = 51$; in the second one, i is incremented and j is decremented, and the loop exits with $i = 102$ and $j = -1$. The standard approach finds, at node ④, $Y^{(4)} = j \leq -1 \wedge j \leq i+1$ and $Z_1^{(4)} = 51 \leq i \wedge j = -1$ and does not discover $i \leq 102$.

```
var i,j:int;
begin ①
  i=0; j=0; ②
  while true do
    if i<=50 then j=j+1;
             else j=j-1;
    if j<0 then goto ④
    i=i+1;
  done; ④
end
```

Figure 6: Example: loop with conditional

The idea of guided static analysis is to perform a sequence of ascending/descending iterations on a increasingly larger part of the CFG. The intuition is that widening makes the implicit assumption that the behavior of the program is "regular". This assumption is obviously violated when a new behavior is activated in the program (in Fig. 6, such a new behavior is the activation of the "else" branch in the loop body). Hence the principle of [18] is (i) to discover the currently active part of the CFG (by a simple propagation); (ii) to perform a complete analysis with widening and narrowing on this part, starting from the invariants discovered so far; (iii) and only at this point to go back to step (i) to check whether new parts of the CFG may be activated. The process is iterated up to convergence, which is guaranteed because the CFG is finite.

In this example, guided static analysis detects that only the "then" branch is initially activated. The ascending sequence on the active part of the CFG discovers $0 \leq i = j$ at node ② *followed by a descending sequence* that adds the bound $i \leq 51$. Only at this point does it take into account the activation of the "else" branch. The technique restarts a new analysis from the invariants inferred so far, and eventually obtains $Z_1^{(4)} = 51 \leq i \leq 102 \wedge j = -1$.

Our last observation is the following:

(5) Thresholds are useful not only to bound $lfp(G)$, but also to *temporarily* bound the ascending iteration up to the activation of a new behavior.

In this example, widening with thresholds behaves like guided static analysis, provided that the threshold constraint $i \leq 51$ is inferred.

3.3 Rationale for inferring thresholds

We made the following observations in the previous sections:

(1)(2) For a while loop, the relevant threshold constraints are found in the postcondition of the guard of the loop by its body.

(3) Computing this postcondition may imply a fixpoint computation when the loop body itself contains loops; but then it implies widening.

(4) Threshold constraints inferred at a widening node should be propagated to the other widening nodes of the CFG.
(5) Thresholds are useful not only to bound the extrapolation, but also to detect the activation of new behaviors and to emulate guided analysis.

Because of observations (1)(2)(4), we tried solutions consisting in propagating constraints, without trying to converge to a fixpoint because of observation (3). We list below several unsuccessful attempts we made to infer relevant thresholds in our running examples, in order to motivate the solution proposed in this paper.

a) The first idea to implement observations (1)(2) was to compute the first steps of a greatest fixpoint computation by computing the first term of the sequence $Y_0 = \top, Y_{n+1} = G(Y_n)$. On the simple loop example of Fig. 2, we obtain

$$Y_1^{(2)} = \{i{=}0 \wedge j{=}10\} \sqcup \{i{\leq}j{+}3\} = \{i{\leq}j{+}3\}$$

which is the needed threshold constraint, see Eqn. (7). This technique actually provides an upper bound for $lfp(G)$ according to Tarski's theorem. However, for the example of Fig. 3 this technique fails to infer any information, because of the approximation induced by the least upper bound:

$$Z_1^{(2)} = \{i{\leq}10\} \sqcup \{j{\leq}10\} = \top$$

b) We then considered the disjunctive completion $\wp(A)$ instead of A for propagating thresholds. In practice we replace the least upper bound in A by set union. This works for the example of Fig. 3 above, but not for the example of Fig. 4. Indeed, because of the self-loop labeled by identity, we have

$$Z_1^{(2)} = \overbrace{\top}^{G^{2\rightsquigarrow 2}(\top)} \cup\{i{\leq}99\} \cup \{i = 0\} = \top$$

c) The remedy is *not to simplify disjuncts* of abstract values in $\wp(A)$, *i.e.,* we do not simplify $a_1 \cup a_2$ with a_2 when $a_1 \sqsubseteq a_2$, as we did above with $a_2 = \top$. However we fail to infer an important threshold constraint for the example of Fig. 5:

n	$Z_n^{(2)}$	$Z_n^{(3)}$
	$G^{1\rightsquigarrow 2} \cup G^{3\rightsquigarrow 2}$	$G^{2\rightsquigarrow 3} \cup G^{3\rightsquigarrow 3}$
1	$\{i{=}j{=}0\}\cup\{j{\geq}10\}$	$\{i{=}0\wedge j{=}0\}\cup\{i{\leq}9\wedge j{=}0\}\cup \{j{\leq}10\}$
2	$\{i{=}j{=}0\}\cup\{j{=}10\}$	$\{i{=}0\wedge j{=}0\}\cup\{i{\leq}9\wedge j{=}0\}\cup\{i{=}0\wedge j{=}1\} \cup \{i{\leq}9\wedge j{=}1\} \cup \{j{\leq}10\}$

The constraint $i \leq 10$ does not appear in $Z^{(2)}$ as we would like to. This is because in $Z^{(3)}$, the constraint $i{\leq}9$ appears only in conjunction with $j{=}0$ or $j{=}1$, hence it is not propagated by the edge ③ $\rightsquigarrow$ ② which is guarded by the exit condition of the inner loop $j{\geq}10$.

d) The solution described in Section 4 is thus a variant of the previous attempt in which after computing a transfer function $G^{(k)}$, we decompose the result of the form $\bigvee_i P_i = \bigvee_i(\wedge_j c_{i,j})$ in a pure disjunction of atomic constraints $\bigvee_{i,j} c_{i,j}$ (seen as a disjunction of convex polyhedra). This way, we also take into account observation (5).

Another intuition we did not explicitly refer to is the idea of propagating (backward to the loop head) the negation of the tests attached to transitions exiting a loop [23]. Our technique will actually propagate forward the conditions for staying in the loop body, which has a similar effect. In addition, it also emulates guided analysis by propagating tests attached to conditionals inside the loop.

4 Inferring thresholds by propagating disjunctions

We assume the hypothesis of Section 2: we have to solve Eqn. (4), which is abstracted in the abstract domain A into Eqn. (5) (*i.e.*, $Y^{(k)} = G^{(k)}(Y)$ with $Y = (Y^{(1)}, \ldots, Y^{(K)}) \in A^K$). A is equipped with a widening operator ∇.

Definition 1 (Widening with thresholds) *Given two abstract values $a_1, a_2 \in A$, and a finite set $\mathcal{T} \subseteq A$ of* threshold values*, we define*

$$a_1 \nabla_{\mathcal{T}} a_2 = (a_1 \nabla a_2) \sqcap \bigsqcap \{a \in \mathcal{T} \mid a_1 \sqsubseteq a \wedge a_2 \sqsubseteq a\}$$

Such a widening meets the technical assumptions of a widening, because the set of thresholds is finite and each threshold either is satisfied by the computed post-fixpoint, or has been violated at some point and did not play any role later.

4.1 The method

Extracting thresholds from an abstract property. We need to extract threshold values from an abstract value, as sketched in attempt (d) of Section 3.3.

We thus assume that we have an *extraction function* $\pi : A \to \wp(Elt(A))$ that extracts, from any value $a \in A$, a set of "elementary" abstract values $\{a_1, \ldots, a_t\}$ that satisfies $\forall i : a \sqsubseteq a_i$. The definition of $Elt(A) \subseteq A$ depends on the domain A and possibly on the widening operator ∇. For *numerical domains*, $Elt(A)$ is typically the set of numerical constraints on which abstract values are built by conjunction:

- For instance, for boxes, octagons or convex polyhedra, it is the set of constraints respectively of the form $a_i x_i \geq b$ with $a_i \in \{-1, 1\}$, $a_i x_i + a_j x_j \geq b$ with $a_i, a_j \in \{-1, 1\}$, or $\sum_i a_i x_i \geq b$ with $a_i \in \mathbb{Q}$. $\pi(a)$ extracts from a the minimal set of such constraints defining it.

  A similar principle holds for Max-Plus polyhedra [24], although there is no unique constraint representation.
- Other domains are not defined directly as the conjunction of constraints. For instance, in the zonotope domain [25], an abstract value represents the set of vectors $\vec{x} \in \mathbb{R}^n$ satisfying equations of the form $\forall 0 \leq i < n : x_i = x_i^0 + \sum_{0 \leq j \leq m} a_i^j \epsilon_j$ in which $\vec{x^0}$ is the center of the zonotope, the a_i^j's are coefficients, and the ϵ^j's are existentially quantified noise symbols constrained in the interval $[-1, 1]$. The widening operator proposed in [26] proceeds in several step: (i) elimination of unstable noise symbols, (ii) then merging of unshared noise symbols, (iii) and last, interval widening on the interval part. One can thus consider interval thresholds, by defining $\pi(a)$ as the set of constraints defining the bounding box of the zonotope a.

π is extended to the disjunctive domain $\wp(A)$ with $\pi(X) = \bigcup_{a \in X} \pi(a)$.

Propagating thresholds in the system of equations. We now assume that Eqn. (4) is abstracted into $\wp(A)$ rather than A. This can be done by replacing $\sqcup$ by $\cup$ inside the functions $G^{(k)}$ in Eqn. (5). We thus have an equation system $T^{(k)} = G_d^{(k)}(T)$ with $T = (T^{(1)}, \ldots, T^{(K)}) \in (\wp(A))^K$. We also assume that, in the disjunctive domain $\wp(A)$, disjuncts are not simplified using the order $\sqsubseteq$ in A.[1]

[1] In our implementation, we just remove duplicated values.

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq j+3}, i=0, j=10\}$
2	$\{i \leq 12, i=2, j \geq -1, j=9\}$

k	2	3
$Y^{(k)}$	$i+2j=20 \wedge$ $3i \leq 26 \wedge i \geq 0$	$i+2j=20 \wedge$ $22 \leq 3i \leq 26$

(a) Single loop example of Fig. 2

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq 10}, \underline{j \leq 10}, i=0, j=10\}$
2	$\{i=1, j=1\}$

k	2	3
$Y^{(k)}$	$0 \leq i \leq 10 \wedge 0 \leq j \leq 10$	$i=10 \wedge j=10$

(b) Two non-deterministic loops example of Fig. 3

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq 99}, i=0, \top\}$
2	$\{i=1\}$

k	2	3
$Y^{(k)}$	$0 \leq i \leq 99$	$1 \leq i \leq 100$

(c) Single loop with break example of Fig. 4

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$	$T_n^{(3)} \setminus T_{n-1}^{(3)}$
1	$\{j \geq 10, i=0, j=0\}$	$\{\underline{i \leq 9}, \underline{j \leq 10}, i=0, j=0\}$
2	$\{\underline{i \leq 10}, \underline{j=10}, i=1\}$	$\{i=1, j=1\}$

k	2	3	4
$Y^{(k)}$	$j \leq 10i \wedge i \leq 10 \wedge$ $0 \leq j \leq 10$	$0 \leq i \leq 9 \wedge$ $0 \leq j \leq 10$	$i=10 \wedge j=10$

(d) Nested loops example of Fig. 5

Figure 7: Inferring thresholds and widening with thresholds on running examples

We infer thresholds by considering the first steps of the following sequence:

$$\begin{array}{rcl} T_0^{(k)} & = & \top_{\wp(A)} = \{\top_A\} \\ T_{n+1}^{(k)} & = & \pi \circ G_d^{(k)}(T_{n+1}^{(0)} \ldots T_{n+1}^{(k-1)}, T_n^{(k)} \ldots T_n^{(K)}) \end{array} \tag{8}$$

Given a number N of iterations, we define the set $\mathcal{T}^{(k)}$ of threshold values attached to the node $k \in T$ as $\mathcal{T}^{(k)} = T_N^{(k)}$. In practice, we take $N = 2$. This allows us to propagate conditions from loop heads to each node of their body (first iteration) but also to propagate conditions of possible inner loops back to the head of the outer loops (second iteration).

Applying widening with thresholds. Finally we solve Eqn. (5) by computing the sequence defined by Eqn. (6) in which $\nabla_{\mathcal{T}^{(k)}}$ replaces the standard widening operator ∇:

$$\begin{array}{rcl} Y_0^{(k)} & = & \bot \quad Y_{n+1}^{(k)} = \left\{ \begin{array}{ll} Y_n^{(k)} \nabla_{\mathcal{T}_k} Y' & \text{if } k \in W \\ Y' & \text{otherwise} \end{array} \right. \\ & & \text{where } Y' = G^{(k)}(Y_{n+1}^{(0)} \ldots Y_{n+1}^{(k-1)}, Y_n^{(k)} \ldots Y_n^{(K)}) \end{array} \tag{9}$$

4.2 Application to the running examples

Figs. 7 shows the application of our method to the examples described in Section 3. In each subfigure, the upper table shows the thresholds computed at each step while the lower table gives the result of the ascending sequence using thresholds. In all cases, the ascending sequence discovers the expected invariant.

- We do not break equality constraints $e=0$ in $e \geq 0 \wedge e \leq$ during the inference of thresholds, but we do it at the end of the inference. For instance, in Fig. 7(d), the threshold $j \leq 10$ at node ② is extracted from the value $j=10$.
- Although our method infers many useless threshold constraints, it does infer all the required ones (which are underlined). It can be noticed that the second iteration step adds useful threshold constraints only in the nested loop example: this confirms observation (4) of Section 3.3.

- The number of useless threshold constraints might be a concern: it slows down the inference itself, and it increases the computation cost of widening with thresholds. For instance, a sequence of n conditionals might generate 2^n threshold constraints at the end of the sequence. We did not yet study pruning heuristics, which are discussed in Section 5 Yet in this case, convex polyhedra defined by one constraint have a cheap complexity.

4.3 Generalizing the method to structured abstract domains.

Some abstract domains are built as functors $\mathcal{F}[A]$ on top of a basic abstract domain A. Widening on such domains is a combination of widening on the structure $\mathcal{F}$ and on the basic domain A. The method of Section 4.1 would infer thresholds in the domain $\wp(\mathcal{F}[A])$. However, it might be more relevant to consider instead the domain $\mathcal{F}[\wp(A)]$, or even $\wp(A)$ if the structure $\mathcal{F}$ does not require thresholds. We illustrate this issue on two examples.

The BddApron library [27] implements the domain $A^{\mathcal{BN}} = \mathbb{B}^m \to A^{\mathcal{N}}$ for abstracting concrete logico-numerical properties belonging to $\mathbb{B}^m \to \wp(\mathbb{R}^n) \simeq \wp(\mathbb{B}^m \times \mathbb{R}^n)$, where $A^{\mathcal{N}}$ is a numerical abstract domain for $\wp(\mathbb{R}^n)$ equipped with a widening $\nabla^{\mathcal{N}}$. Abstract elements are efficiently represented with MTBDDs. Widening on $A^{\mathcal{BN}}$ is defined as $f \nabla^{\mathcal{BN}} g = \lambda \vec{b} \in \mathbb{B}^m \,.\, f(\vec{b}) \nabla^{\mathcal{N}} g(\vec{b})$ and acts only on $A^{\mathcal{N}}$.

We could define $\nabla^{\mathcal{BN}}_{\mathcal{T}}$ with Def. 1 and define $\pi^{\mathcal{BN}}(f) = \{\lambda \vec{b}.Y \mid Y \in \pi^{\mathcal{N}} \circ f(\vec{b})\} \subseteq A^{\mathcal{BN}}$. But as no widening is performed on the Boolean part, we can define the operator $\nabla^{\mathcal{BN}}_{\mathcal{T}}$ and the function $\pi^{\mathcal{BN}}$ in a more specific and efficient way:

- $\pi^{\mathcal{BN}}(f) = \bigcup_{\vec{b}} \pi^{\mathcal{N}} \circ f(\vec{b}) \subseteq A^{\mathcal{N}}$ extracts the set of *numerical* thresholds in f;
- $f \nabla^{\mathcal{BN}}_{\mathcal{T}} = \lambda \vec{b} \,.\, f(\vec{b}) \nabla^{\mathcal{N}}_{\mathcal{T}} g(\vec{b})$ where $\mathcal{T} \subseteq A^{\mathcal{N}}$.

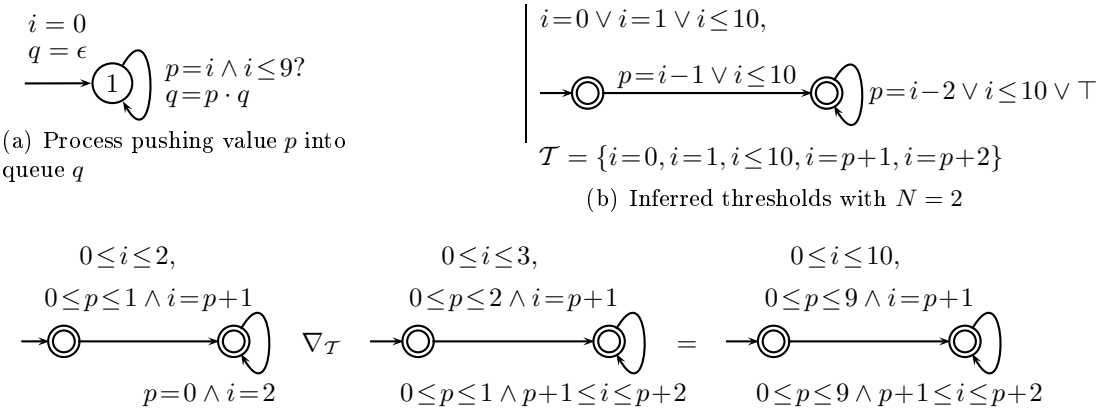
We implemented this later solution in our ConcurInterproc tool [28, 29] which exploits the BddApron library, in order to perform the experiments described in the next section.

Our second example is the lattice automata abstract domain defined in [9] for abstracting properties on unbounded sequences of elements, belonging to the domain $C = \wp\big((\mathbb{R}^n)^*\big)$. The motivation is the analysis of FIFO channels in communicating systems exchanging numerical data, see Fig. 8(a). The proposed abstract domain $\mathrm{Reg}(A^{\mathcal{N}})$ in is the set of *lattice automata*, that are finite automata, the transitions of which are labelled with elements of some numerical *atomic* lattice $A^{\mathcal{N}}$. Such an automaton recognizes words of atomic abstract elements; for instance, the first automaton of Fig. 8(c) recognizes the word $(i = 0 \wedge p = 1) \cdot (i = 0 \wedge p = 2)^2$. Widening on such automata proceeds in two steps: the first one bounds the size of the automata, the second one deals with the convergence of numerical properties labelling the transitions:

- If the two argument automata have a different structure, widening simplifies the structure by merging states and transitions.
- Otherwise, the labelled transitions of each automaton can be associated pairwise. Widening consists in applying the widening operator $\nabla^{\mathcal{N}}$ on such pairs of labels and in returning an automaton with same structure, see Fig. 8(c).

In practice, thresholds are not needed on the automata structure, so we propose to consider the domain $\mathrm{Reg}(\wp(A^{\mathcal{N}}))$ for propagating thresholds, and at the end of the inference to gather all the numerical thresholds corresponding to the different labels in a set $\mathcal{T} \subset A^{\mathcal{N}}$. Fig. 8(b) shows the automaton in $\mathrm{Reg}(\wp(A^{\mathcal{N}}))$ obtained by the sequence of Eqn. (8) with $N = 2$, from which one extracts the set $\mathcal{T}$. We then replace $\nabla^{\mathcal{N}}$ with $\nabla^{\mathcal{N}}_{\mathcal{T}}$ in the widening sketched above.

The automata recognizing languages on Fig. 8(b)(c) are labelled with abstract values on both variables i and p : this is because we use the non-standard semantics defined in [9] that relates the content of the FIFO queue q to the current value of variables manipulated by the process. Fig. 8(c) shows the last but one application of widening, which is the first application of widening in the analysis for which the two arguments have the same structure. The analysis



(a) Process pushing value p into queue q

(b) Inferred thresholds with $N = 2$

(c) Last but one widening with thresholds ($i{\leq}10$, $p{+}1{\leq}i$ and $i{\leq}p{+}2$ are preserved)

Figure 8: Inference of thresholds and widening with thresholds on lattice automata

Program	Guided vs Standard	Policy vs Guided	Thresholds vs Policy
test5	=	4/0	=
test6	0/4	6/−4	0/4
test7	=	9/0	−4/0
test8	=	4/0	=
test9	2/0	4/0	=
test1, test2, test3, test4: same results (simple examples)			

N1/N2 in column A vs B: number N1 of *additional* finite interval bounds and number $N2$ of *improved* finite interval bounds found by technique A compared to technique B, in all the program CFG; "=" indicates identical results.

Table 1: Comparison between standard, guided, policy iteration, and thresholds techniques using the box domain, on the examples of [4]

converges to the automaton on the right in Fig. 8(c) modified by removing the constraint $i{\leq}p{+}2$. On such an example, narrowing following widening without thresholds would not recover the information $p{\leq}9$ for all the elements in the FIFO queue, because in the abstraction the size of the FIFO queue is unbounded.

5 Experiments and concluding remarks

We implemented our inference technique for the BddApron logico-numerical abstract domain used by the ConcurInterproc tool, as described in Section 4.3.[2] We first consider the box abstract domain, and three alternative methods: (1) the **standard** Kleene iteration with widening and descending sequence, as described in Section 2; (2) the **guided** static analysis technique of [18]; (3) and the **policy** iteration technique of [4] mentioned in the introduction, which is able to converge to the least fixpoint under some assumptions; it is actually implemented only in the simpler Interproc tool [30]. Tab. 2 compares the results of the 4 methods on the examples of [4], which are purely numerical, by counting the total number of better bounds inferred by one technique over the other. On these tricky examples:

- Guided analysis is always better than standard analysis;

[2] These experiments are available and can be run with the online version of the analyzer, see [29].

Program	CFG Size	Standard		Guided		Inf. of Thres.		Thresholds	
	#K/#F	Time	Prec.	Time	Prec.	Time	Av.nb.	Time	Prec.
Sequential, intraprocedural programs									
loop1	3/3	0.02	=	0.03	=	0.02	14	0.02	=
loop_nondet	3/4	0.03	B	0.03	B	0.02	12	0.04	A
loop_reset	4/6	0.01	B	0.01	B	0.01	6	0.02	A
loop2	4/5	0.06	B	0.09	B	0.02	12	0.08	A
gopanreps	4/6	0.06	B	0.09	A	0.04	16	0.08	A
loop2Bis	5/7	0.14	B	0.24	B	0.07	18	0.20	A
gopanrepsBis	5/8	0.29	B	0.49	B	0.28	39	0.85	A
nestedLoop	5/8	0.61	B	0.68	B	0.58	39	0.72	A
sipma91	7/11	0.35	B	0.42	B	0.57	33	0.37	A
car	3/4	0.06	=	0.07	=	0.01	14	0.06	=
Concurrent programs									
concurrent_loop	9/16	0.04	B	0.04	B	0.07	8	0.05	A
loop2_TLM	24/26	0.24	B	0.25	B	1.63	19	0.33	A^+
barrier_counter_2	61/108	1.71	B	1.91	B	2.09	18	4.90	A^+
barrier_counter_3	405/847	158.00	B	190.00	B	1553.00	78	1096.00	A^+
Programs with non-inlined procedure calls									
loop2_rec	15/18	0.25	B	0.42	B	1.88	28	0.47	A
gopanreps_rec	9/11	0.22	B	0.38	A	2.17	46	0.46	A
loop2Bis_rec	16/20	1.07	B	1.74	B	23.75	43	1.25	A
gopanrepsBis_rec	17/21	3.29	B'	9.23	A	651.00	82	9.86	B"
loop2_TLM_rec	34/38	0.86	B	0.86	B	17.76	20	1.97	A^+

#K/#F: size of the CFG, with #K the number of control nodes and #F the number of basic blocks; **Time:** running times in seconds, on a MacBook Air (Intel Core 2 Duo, 2.13 GHz); **Prec.:** relative precision: A is best, C is worse; A^+ indicates the proof of a specific property; **Av.nb.** : average number of inferred threshold constraints at each CFG node.

Table 2: Comparison between standard, guided and our technique (inference+analysis), using the convex polyhedra domain

- Policy iteration is better than guided analysis, with the exception of **test6**, where it infers 6 additional finite bounds, but where 4 of the other inferred bounds are less accurate. Widening with thresholds does strictly better than the other techniques here.
- **test7** is the only example for which widening with thresholds is less accurate than policy iteration, but still more accurate that guided analysis.

These experiments showed us the usefulness of considering also the constraint $x \geq 0$ when $x \leq 0$ is inferred. Typically, if we have a *inner* loop `while (x>=1) do x-`, the exit constraint $x \leq 0$ will be propagated to the *outer* loop head, whereas it is the constraint $x \geq 0$ which is relevant as a threshold at this point.

We then considered the convex polyhedra abstract domain combined in BddApron with finite-state variables, Tab. 1. Policy iteration could not be experimented, because it applies only to template-like domains as discussed in the introduction. For all but 5 of these examples, widening with thresholds is strictly more precise than the standard or guided analyses, and it is less precise than guided analysis for a single example. W.r.t. efficiency,

- For the sequential, simple examples, the additional cost can be considered moderate (at most a factor 2.0, including the inference time), even when the number of inferred thresholds is not so small;
- For concurrent programs, the additional complexity is higher and may be dramatic in some cases, typically **barrier_counter_3** for which the number of thresholds have an impact of the analysis time (factor 6.0 w.r.t. standard analysis, besides the inference time). The performance

problem here could be fixed easily by performing a thread-modular inference, which would infer the required thresholds on these examples (checked by manual inspection);

- For relational interprocedural analysis, we also have a performance problem, which results from the procedure return operation that implies a relation composition between abstract values; applied to disjunctions of constraints, the cost is quadratic! A manual inspection shows that many useless constraints are combined in this way. This problem desserves further investigations. Observe however that the technique infers the right thresholds, when for instance nested loops are implemented as tail recursive calls (**X_rec** versions of **X** examples).

To conclude, our technique is very successful w.r.t. precision, but needs efficiency improvements for concurrent and recursive programs.

Abstract acceleration [17] might be better than our technique, for instance on the **car** example, because it computes $\alpha \circ F^* \circ \gamma$ instead of the less precise $(\alpha \circ F \circ \gamma)^*$. It should combine efficiently with our technique. In particular we could use accelerated transitions to propagate thresholds created in outer loops. However abstract acceleration alone does not solve the **nestedLoop** example with 3 nested loops, and is hardly applicable if loops are transformed in tail-recursive calls. [31] gives some details about the use of thresholds in the Astrée analyzer. They infer thresholds only for single variables, and they consider intraprocedural programs (procedures are inlined).

References

[1] Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. Journal of Logic Programming **13** (1992)

[2] Su, Z., Wagner, D.: A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'04. Volume 2988 of LNCS. (2004)

[3] Gawlitza, T., Leroux, J., Reineke, J., Seidl, H., Sutre, G., Wilhelm, R.: Polynomial precise interval analysis revisited. In: Efficient Algorithms. Volume 5760 of LNCS. (2009)

[4] Costan, A., Gaubert, S., Goubault, E., Martel, M., Putot, S.: A policy iteration algorithm for computing fixed points in static analysis of programs. In: Computer Aided Verification, CAV'05. Volume 3576 of LNCS. (2005)

[5] Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: European Symposium on Programming, ESOP'07. Volume 4421 of LNCS. (2007) 300–315

[6] Gaubert, S., Goubault, E., Taly, A., Zennou, S.: Static analysis by policy iteration on relational domains. In: European Symp. on Programming, ESOP'07. Volume 4421 of LNCS. (2007)

[7] Gawlitza, T., Seidl, H.: Precise relational invariants through strategy iteration. In: Computer Science Logic, CSL'07. Volume 4646 of LNCS. (2007) 23–40

[8] Halbwachs, N., Proy, Y., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design **11** (1997)

[9] Le gall, T., Jeannet, B.: Analysis of communicating infinite state machines using lattice automata. Technical Report 1839, IRISA (2007)

[10] Feuillade, G., Genet, T., Tong, V.V.T.: Reachability analysis over term rewriting systems. Journal of Automated Reasoning **33** (2004) 341–383

[11] Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: PLILP'92. Volume 631 of LNCS. (1992)

[12] Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: 2nd Int. Symp. on Programming, Dunod, Paris (1976)

[13] Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation **19** (2006)

[14] Bagnara, R., Hill, P.M., Ricci, E., Zafanella, E.: Precise widening operators for convex polyhedra. Science of Computer Programming (SCP) **58** (2005)

[15] Simon, A., King, A.: Widening polyhedra with landmarks. In: Programming Languages and Systems, APLAS'06. Volume 4279 of LNCS. (2006)

[16] Simon, A., Chen, L.: Simple and precise widenings for polyhedra. In: Programming Languages and Systems, APLAS'10. Volume 6461 of LNCS. (2010)

[17] Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Static Analysis Symposium, SAS'06. Volume 4218 of LNCS. (2006)

[18] Gopan, D., Reps, T.W.: Guided static analysis. In: Static Analysis Symposium, SAS'07. Volume 4634 of LNCS. (2007)

[19] Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Why does astrée scale up? Formal Methods in System Design **35** (2009)

[20] Knoop, J., Steffen, B.: The interprocedural coincidence theorem. In: Compiler Construction, CC'92. Volume 641 of LNCS. (1992)

[21] Jeannet, B., Serwe, W.: Abstracting call-stacks for interprocedural verification of imperative programs. In: Int. Conf. on Algebraic Methodology and Software Technology, AMAST'04. Volume 3116 of LNCS. (2004)

[22] Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: International Conference on Formal Methods in Programming and their Applications. Volume 735 of LNCS. (1993)

[23] Halbwachs, N.: A heuristics for inferring thresholds. (personal communication)

[24] Allamigeon, X., Gaubert, S., Goubault, E.: Inferring Min and Max Invariants Using Max-plus Polyhedra. In: Static Analysis Symposium, SAS'08. Volume 5079 of LNCS. (2008)

[25] Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: Computer Aided Verification, CAV'09. Volume 5643 of LNCS. (2009)

[26] Goubault, E., Putot, S.: A zonotopic framework for functional abstractions. CoRR **abs/0910.1763** (2009)

[27] Jeannet, B.: The BDDAPRON logico-numerical abstract domains library. (`http://www.inrialpes.fr/pop-art/people/bjeannet/bjeannet-forge/bddapron/`)

[28] Jeannet, B.: Relational interprocedural verification of concurrent programs. In: Software Engineering and Formal Methods, SEFM'09, IEEE (2009)

[29] Jeannet, B.: The ConcurInterproc interprocedural analyzer for concurrent programs. (`http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi`)

[30] Jeannet, B., Argoud, M., Lalire, G.: The Interproc interprocedural analyzer. (`http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`)

[31] Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Combination of abstractions in the astrée static analyzer. In: ASIAN. Volume 4435 of LNCS. (2008)

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399