When the Decreasing Sequence Fails

Nicolas Halbwachs, Julien Henry

When the decreasing sequence fails[⋆]

Nicolas Halbwachs, Julien Henry

Vérimag[⋆⋆], Grenoble University – France

Abstract. The classical method for program analysis by abstract interpretation consists in computing a increasing sequence with widening, which converges towards a correct solution, then computing a decreasing sequence of correct solutions without widening. It is generally admitted that, when the decreasing sequence reaches a fixpoint, it cannot be improved further. As a consequence, all efforts for improving the precision of an analysis have been devoted to improving the limit of the increasing sequence. In this paper, we propose a method to improve a fixpoint after its computation. The method consists in projecting the solution onto well-chosen components and to start again increasing and decreasing sequences from the result of the projection.

1 Introduction

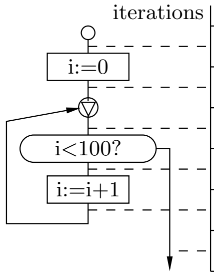
Program analysis by abstract interpretation [CC77] consists in computing an upper approximation of the least fixpoint of an abstract semantic function in a suitable abstract lattice of properties. When the abstract lattice is of infinite depth, the standard approach [CC76,CC77] consists in computing an *increasing sequence* whose convergence is forced using a *widening operator*; then, from the obtained limit of the increasing sequence, one can improve the solution by computing a *decreasing sequence*, by iterating the function without widening. The decreasing sequence may either stop at a fixpoint of the semantic function, or be infinite, but since all its terms are correct solutions, one can stop the computation after a fixed number of terms, or limit its length using a *narrowing operator*.

Of course, the precision of the result depends both on the ability of the widening operator to "guess" a precise limit of the increasing sequence, and on the information gathered during the decreasing sequence. Intuitively, the increasing sequence extrapolates the behaviour of the program from the first steps of its execution, while the decreasing sequence gathers information about the end of the execution of the program, its loops, or more generally, the way the strongly connected components of its control flow graph are left.

While significant efforts have been devoted to improving the precision of the limit of the increasing sequence (see §1.2 for a quick survey), little attention

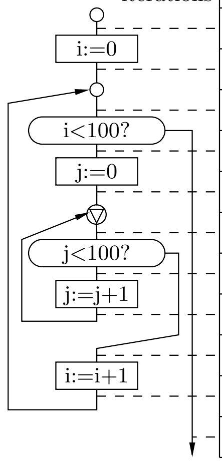
[⋆] This work has been partially supported by the ASOPT project of the "Agence Nationale de la Recherche" of the French Ministry of Research

[⋆⋆] Verimag is a joint laboratory of Université Joseph Fourier, CNRS and Grenoble-INP.



iterations	1	2	3↓
	$\top$		
	$[0,0]$		
	$[0,0]$	$[0,\infty]$	$[0,100]$
	$[0,0]$	$[0,99]$	
	$[1,1]$	$[1,100]$	
	$\bot$	$[100,\infty]$	$[100,100]$
		convergence	fixpoint

(a) A classical example where the decreasing sequence reaches the least fixpoint



iterations	1	2	3	4	5↓
	$\top$				
	$[0,0]$				
	$[0,0]$		$[0,1]$	$[0,\infty]$	
	$[0,0]$		$[0,1]$	$[0,99]$	
	$[0,0][0,0]$		$[0,1][0,0]$	$[0,99][0,0]$	
	$[0,0][0,0]$	$[0,0][0,\infty]$	$[0,\infty][0,\infty]$	$[0,\infty][0,\infty]$	$[0,\infty][0,100]$
	$[0,0][0,0]$	$[0,0][0,99]$	$[0,\infty][0,99]$		
	$[0,0][1,1]$	$[0,0][1,100]$	$[0,\infty][1,100]$		
	$\bot$	$[0,0][100,\infty]$	$[0,\infty][100,\infty]$		$[0,\infty][100,100]$
	$\bot$	$[1,1][100,\infty]$	$[1,\infty][100,\infty]$		$[1,\infty][100,100]$
	$\bot$	$\bot$		$[100,\infty][100,\infty]$	$[100,\infty][100,100]$
				convergence	fixpoint

(b) A nested loop prevents the decreasing sequence to get precise results

Fig. 1. Example 1 – nested loops

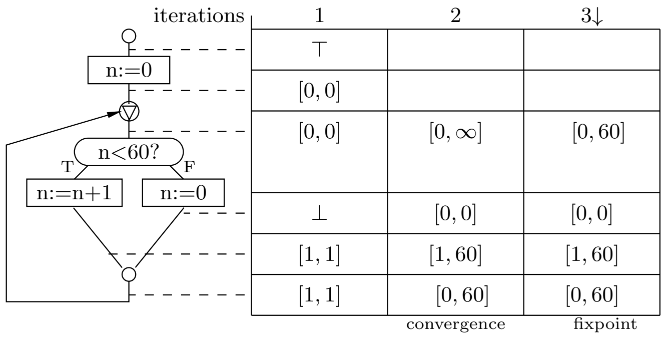
has been paid to the decreasing sequence. It is generally admitted that, when the decreasing sequence reaches a fixpoint, it cannot be improved. However, it appears that such a fixpoint can be far from the least fixpoint, so improving it may have a significant influence. More specifically, we shall see in §1.1 that slight modifications in a program may have a surprising influence on the amount of information gathered during the decreasing sequence.

1.1 Motivating examples

Let's illustrate the problem with two very simple examples:

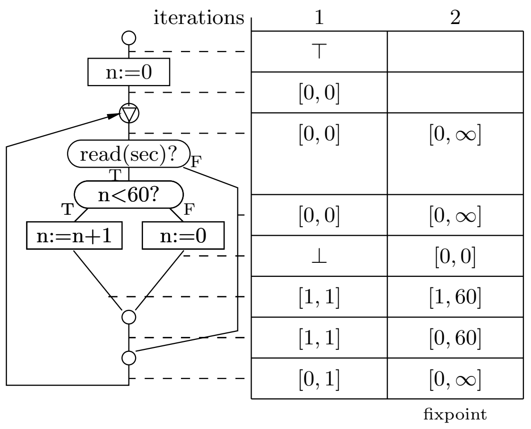
Example 1 is a classical example of what can be obtained by interval analysis [CC76]. Fig. 1.a shows the control-flow graph (CFG) of a very simple loop incrementing a variable i from 0 to 100. The increasing sequence consists of 2 iterations; at iteration 2, the widening is applied, and the sequence converges. Iteration 3 shows the descending sequence, which reaches a fixpoint in 1 step. The results are the best possible: $i \in [100, 100]$ at the end of the loop. Now, the CFG shown in Fig.1.b is obtained from the preceding one by nesting a second loop on another variable j within the first one. The increasing sequence converges after 4 steps. Again, the descending sequence reaches a fixpoint in 1 step, but now, the result for i is imprecise: $i \in [100, \infty]$ at the end. The reason is that the nested loop neither modifies nor tests the variable i; so, as soon as its interval has been widened to $[0, \infty]$, it will remain unchanged in the inside loop. Notice that it is also the case if we select both loop heads as widening nodes (see Note 1 in §2 on the selection of widening nodes).

Example 2: Our second example illustrates a situation which occurs commonly in reactive programs, cyclically sampling their environment. A first program (Fig. 2.a) is just an infinite loop with a counter modulo 60. It is properly analysed after 2 steps of increasing sequence and one descending step.



1	2	3↓
$\top$		
$[0,0]$		
$[0,0]$	$[0,\infty]$	$[0,60]$
$\bot$	$[0,0]$	$[0,0]$
$[1,1]$	$[1,60]$	$[1,60]$
$[1,1]$	$[0,60]$	$[0,60]$
	convergence	fixpoint

(a) An infinite loop with a counter modulo 60



1	2
$\top$	
$[0,0]$	
$[0,0]$	$[0,\infty]$
$[0,0]$	$[0,\infty]$
$\bot$	$[0,0]$
$[1,1]$	$[1,60]$
$[1,1]$	$[0,60]$
$[0,1]$	$[0,\infty]$
	fixpoint

(b) A loop counting the occurrences of "seconds"

Fig. 2. Example 2 – intermittent counting

Now, assume that we don't want to count all loop iterations, but only when some external event (e.g., a "second") is detected. This is done by the program of Fig. 2.b. As before, the increasing sequence converges after 2 steps, but now, the limit is a fixpoint, so there is no decreasing sequence, and the upper bound of the counter is missed. This second example can be simply explained as follows: let $(L, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ be the abstract lattice, and F be an abstract semantic function from L to L. Let $G = Id \sqcup F$ (i.e., $\lambda X.X \sqcup F(X)$). Then, it is easy to see that G has the same least fixpoint as F, but G is *extensive* (i.e., $\forall X \in L, X \sqsubseteq G(X)$). As a consequence, any postfixpoint of F is a fixpoint of G. So, while the limit of the increasing sequence with F may be a strict postfixpoint of F — which can be improved by a decreasing sequence —, this limit will be a fixpoint of G, meaning that there is no decreasing sequence with G. This is exactly what happens with our example, where the dummy branch in the loop of Fig. 2.b adds an identity term to the semantic function at the widening node.

1.2 Related works

Beside researches proposing new abstract domains, many existing works aim at fighting the imprecision of analysis, considered to be essentially due to the widening operation. Apart from proposals of systematic design of widening and narrowing operators [CZ11], one can distinguish at least three big tracks: (1) designing smart widening operators, generally dedicated to some specific domains; (2) avoiding or minimising the use of widening, by focusing either on some classes of programs or on some classes of abstract domains; (3) applying widening and narrowing using smart strategies.

Smart widening. Several proposals concern smart widening operators, especially for the polyhedra domain [CH78,BHRZ03]. Widening *up to* or *with thresholds* [Hal93,HPR97,BCC$^+$03,LCJG11] consists in choosing — generally from the conditions appearing in the program — some tentative limits to the widening. Widening *with landmarks* follows the same idea, but the selection of limits is made dynamically. Widening *with care set* [WYGI07] makes use of a proof objective. Some of these proposals can properly deal with some of our examples, mainly because they can reach a precise solution at the end of the increasing sequence. However, they are not independent of the considered abstract domain. Our method will work for any abstract domain, and is compatible with any widening operation.

Avoiding widening. Other authors try to avoid the use of widening. *Acceleration techniques* [BGP97,WB98,CJ98,BFLP03,GH06] are dedicated to some classes of programs or loops, the effect of which can be exactly computed. Other approaches can be applied only with some kinds of domains — namely "weakly relational domains" [Min04] or "templates" [SSM04,SSM05] — in which *policy iteration* [SW04,CGG$^+$05,GS07] allows least fixpoints to be precisely computed. These methods generally solve our problem, but they are restricted either to some class of programs or to some abstract domains.

Widening strategies. An obvious way of improving the precision of the widening is to *delay* its application [Hal93,BCC$^+$03], i.e., applying it only after a fixed number of exact steps or intermittently, or applying it after some *loop unrolling* [Gou01,PGM03]. Some strategies adapt the application of the widening according to the discovery of *new feasible paths* [Hal93,HPR97,BCC$^+$03] in the program. In particular, [GR06,GR07] proposes a very clever strategy, called *lookahead widening*, where a succession of increasing-decreasing sequences are computed for more and more feasible paths of the program. *Stratified analysis* [ML11] is a succession of analyses concerning more and more variables, according to their dependencies. None of these strategies provides a general solution to our problem.

Anyway, while some of these methods can work on some of our examples, none of them specifically address the problem of improving the result of the decreasing sequence. We don't pretend our method is better than these works, but that it is different and complementary.

1.3 Contribution and summary

In this paper, we propose a method starting from the result of the decreasing sequence, and trying to improve it as follows: the solution will be projected on some of its components, whose propagation is likely to provide a more precise solution, according to some criteria. New increasing and decreasing sequences will be started from the result of the projection, providing a new solution which can be intersected with the previous one. The method is independent from the abstract domain. We'll show that it properly solves our running examples, and that in some cases, it can gather non trivial information about the end of program execution.

The paper is organised as follows: Section 2 introduce the necessary definitions and notations; our method is presented in Section 3 and illustrated on an example in Section 4; Section 5 proposes some ways for improving the performances and Section 6 gives some experimental results.

2 Definitions and notations

Abstract lattice. As said before, we assume that the analysis makes use of an abstract complete lattice $(L, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$. We assume this lattice to be of infinite depth. The lattice operations are supposed to be available, together with a widening operator ∇, and the interpretation of each program statement s as a function (predicate transformer) $f_s : L \mapsto L$.

Control-flow graph. A control-flow graph (CFG) is a graph (N, E), where
- the finite set $N = \{\nu_1, ..., \nu_k\}$ is made of 3 types of nodes: the start nodes, the junction nodes, and the statement nodes. With each statement node ν_i is associated a function $f_i : L \mapsto L$.

– $E \subseteq N \times N$ is the set of edges. Start nodes have no incoming edge, statement nodes have one incoming edge, junction nodes have several incoming edges.

Remark: for simplicity, each node has a single output. This means that the classical "test nodes" (used in Figures 1 and 2) are split into pairs of statement nodes, whose associated function returns an abstraction of the intersection of its argument with the condition of the test ("then" part) or its negation ("else" part). As an example, Fig. 3 shows the CFG of our example 1.b.

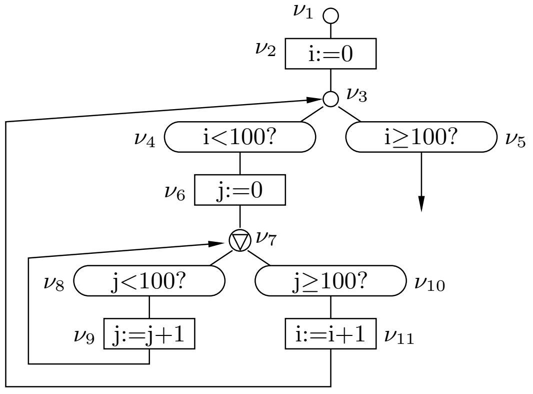


Fig. 3. CFG of the example 1.b

Semantic equations. The analysis will associate with each node ν_i of the CFG an abstract value $X_i \in L$, these abstract values being defined by a system of recursive equations:

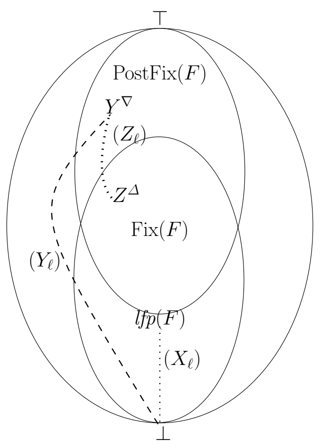
$$\forall i = 1..k,\ X_i = \begin{cases} \top & \text{if } \nu_i \text{ is a start node} \\ f_i(X_j) & \text{if } \nu_i \text{ is a statement node and } (\nu_j, \nu_i) \in E \\ \bigsqcup_{(\nu_j,\nu_i)\in E} X_j & \text{if } \nu_i \text{ is a junction node} \end{cases}$$

We'll often write this system of equations as a vectorial fixpoint equation: $\boldsymbol{X} = F(\boldsymbol{X})$ in the lattice L^k.

Increasing sequence. Since the lattice L is of infinite depth, the Kleene sequence $\boldsymbol{X}_0 = \bot^k, \boldsymbol{X}_{\ell+1} = F(\boldsymbol{X}_\ell)$ may be infinite. The classical approach consists in computing the increasing sequence $\boldsymbol{Y}_0 = \bot^k, \boldsymbol{Y}_{\ell+1} = \boldsymbol{Y}_\ell \nabla F(\boldsymbol{Y}_\ell)$; from the properties of the widening operator, this sequence is guaranteed to converge after a finite number of steps towards a limit $\boldsymbol{Y}^\nabla$, which is a postfixpoint of F, i.e., $F(\boldsymbol{Y}^\nabla) \sqsubseteq \boldsymbol{Y}^\nabla$. Of course, the increasing sequence is computed in a chaotic way (cf. Figures 1 and 2), by propagating changes along the paths of the CFG, and since the widening operation loses information, it is only applied on a selected set W of *widening nodes* intersecting each loop of the CFG.

Note 1 (on the selection of widening nodes). The set W of widening nodes must be as small as possible, to minimise the number of applications of the widening operator. Since finding a minimal cutting set W is an NP-complete problem, the heuristic classically applied is the method of *strongly connected subcomponents* (SCSC) proposed by Bourdoncle [Bou93]: the method recursively uses Tarjan's algorithm [Tar72] to find the strongly connected components (SCC) of a directed graph, together with an entry node to each SCC. Entry nodes are the target of back edges, so *they are all junction nodes.* Bourdoncle's method consists in adding all SCC entry nodes to W, then removing them from the graph and recursively apply Tarjan's algorithm to the rest of each SCC. The result is a hierarchy of SCSC, each of which being cut by a junction node in W. An obvious improvement of this method (which we did not find published anywhere) consists in considering again the hierarchy of SCSC bottom-up, checking whether each SCSC is disconnected by the cut-points of its children. For instance, on the CFG of Fig. 3, a first application of Tarjan's algorithm finds one non-trivial SCC, $c_1 = \{\nu_3, \nu_4, \nu_6, \nu_7, \nu_8, \nu_9, \nu_{10}, \nu_{11}\}$, with entry node ν_3. Removing node ν_3 and applying again the algorithm provides the SCSC $c_2 = \{\nu_7, \nu_8, \nu_9\}$, with entry node ν_7, whose removal disconnects the graph. So, Bourdoncle's method provides $W = \{\nu_3, \nu_7\}$. Now, since the cut-point ν_7 of the leaf SCSC c_2 also disconnects the father SCSC c_1, it's enough to choose $W = \{\nu_7\}$, as done in Fig. 3.

Decreasing sequence. The limit Y^∇ of the increasing sequence is a post-fixpoint of F. If it is a strict post-fixpoint ($F(Y^\nabla) \sqsubset Y^\nabla$), it can be improved by computing a decreasing sequence $Z_0 = Y^\nabla, Z_{\ell+1} = F(Z_\ell)$. This sequence can be infinite, or reach a fixpoint of F. In practice, it generally reaches a fixpoint after very few steps. Anyway, since all of its terms are post-fixpoints of F, hence correct approximations of the least fixpoint, one can stop it after a fixed number of steps, or force its convergence using a *narrowing operator*. In the following, we'll note Z^Δ the last term of the descending sequence, and we'll generally assume that Z^Δ is a fixpoint; however, the results still hold if Z^Δ is a strict post-fixpoint.



The above figure classically illustrates the sequences in the abstract lattice: $(X\ell)$ is the (generally infinite) Kleene's sequence, (Y_ℓ) is the (finite) increasing sequence, leading to the post-fixpoint Y^∇, and (Z_ℓ) is the decreasing sequence of post-fixpoints providing a solution Z^Δ.

3 Improving a (post-)fixpoint solution

3.1 An intuition of the solution

Let's look again at example 1.b (see Fig. 3). At the widening node ν_7, during the decreasing sequence, we have $Z_7 = Z_6 \sqcup Z_9$. At the end of the decreasing sequence, we find $Z_7^\Delta = (i \in [0,\infty], j \in [0,100])$, while $Z_6^\Delta = (i \in [0,99], j \in [0,0])$ and $Z_9^\Delta = (i \in [0,\infty], j \in [1,100])$. Obviously, $i \in [0,99]$, found in Z_6^Δ, is a correct invariant, which is lost in Z_7^Δ because of the least upper bound with $i \in [0,\infty]$ imprecisely found in Z_9^Δ. Our idea is to start again a propagation of Z_6^Δ after resetting Z_9^Δ to $\bot$.

3.2 Generalised sequences

Restarting an iteration from an arbitrary point requires some changes in the definition of the iteration sequences. We must ensure that a widened sequence starting form an arbitrary point (not necessarily a pre-fixpoint) is increasing; moreover, widening operators are generally designed under the assumption that their first operand is smaller than the second one. We introduce the following notations: Let F be a monotone function from L to L. Let $X \in L$. Then,

- we note $F^\nabla(X)$ the limit of the sequence $Y_0 = X$, $Y_{\ell+1} = Y_\ell \nabla (X \sqcup F(Y_\ell))$;
- we note $F^{\nabla\Delta}(X)$ the last term Z^Δ of a descending sequence (Z_ℓ) starting at $Z_0 = F^\nabla(X)$

Remarks:

- The second operand $X \sqcup F(Y_\ell)$ of the widening is always greater than the first one, and the increasing sequence (Y_ℓ) is indeed increasing. Obviously, $F^\nabla(X)$ is the classical approximation of the least fixpoint of the function $\lambda X.(X \sqcup F(Y))$, i.e., of the least fixpoint of F greater than X.
- Notice also that, with $X = \bot$, these definitions of sequences and limits match the classical ones recalled in §2.
- For any X, $F^{\nabla\Delta}(X)$ is a correct approximation of the least fixpoint of F, i.e., $\forall X \in L$, $F^{\nabla\Delta}(X) \sqsupseteq \mathit{lfp}(F)$.
- Neither F^∇ nor $F^{\nabla\Delta}$ is increasing. As a consequence, there can be some X such that $F^{\nabla\Delta}(X) \sqsubset F^{\nabla\Delta}(\bot)$, i.e., such that the limit obtained from X is more precise than the one computed by the classical iteration.

We address the problem of analysing an SCC of the graph, since the analysis of a complex graph considers each SCC in turn. We consider first the case of an SCC with only one widening node, before addressing the general case.

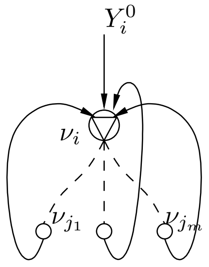
3.3 Case of a single widening node

We need some additional definitions:

Path transformers. Let ν_i, ν_j be two nodes. Let $\mathcal{P}_{i,j}$ denote the set of nodes belonging to an elementary path in the CFG going from ν_i to ν_j. Intuitively the *path transformer* from ν_i to ν_j is the function $F_{i,j} : L \mapsto L$, which, from an abstract value X associated with ν_i, provides the abstract value $F_{i,j}(X)$ corresponding to the propagation of X along the elementary paths from ν_i to ν_j. We have:

$$F_{i,j}(X) = \begin{cases} \bot & \text{if } \mathcal{P}_{i,j} = \emptyset \\ X & \text{if } \nu_i = \nu_j \\ f_j(F_{i,k}(X)) & \text{if } \nu_j \text{ is a statement node and } (\nu_k, \nu_j) \in E \\ \displaystyle\bigsqcup_{\substack{(\nu_k,\nu_j) \in E \\ \nu_k \in \mathcal{P}_{i,j}}} F_{i,k}(X) & \text{if } \nu_j \text{ is a junction node} \end{cases}$$

Now, let us consider an SCC with only one widening node, ν_i. ν_i is a junction node (from the way widening nodes are selected). The abstract value at ν_i depends on those at the preceding nodes in the SCC, and on abstract values propagated from outside (start nodes and preceding SCCs), which we note Y_i^0 since it is the first value $\neq \bot$ at node ν_i during the increasing sequence. Let $\nu_{j_1}, \ldots, \nu_{j_m}$ be the source nodes of incoming back edges to ν_i.The semantic equations considered during the descending sequence can be subsumed as:



$$Z_i = Y_i^0 \sqcup Z_{j_1} \sqcup \ldots \sqcup Z_{j_m} \quad , \quad Z_{j_\ell} = F_{i,j_\ell}(Z_i), \ell = 1..m$$

At the end of the sequence, we have $Z_i^{\Delta} \sqsupseteq Y_i^0 \sqcup Z_{j_1}^{\Delta} \sqcup \ldots \sqcup Z_{j_m}^{\Delta}$ (an equality if $\boldsymbol{Z}^{\Delta}$ is a fixpoint).

Projection according to improving components. The idea is to select a set S of nodes, such that the propagation of terms $\{Z_{j_\ell}^{\Delta} \,|\, \nu_{j_\ell} \in S\}$ is likely to improve the solution. More precisely, we define $\boldsymbol{Z^{\Delta}} \Downarrow S$ by

$$\left(\boldsymbol{Z^{\Delta}} \Downarrow S\right)_k = \begin{cases} Z_k^{\Delta} & \text{if } \nu_k \in S \\ \bot & \text{otherwise} \end{cases}$$

The choice of S should make $F^{\nabla\Delta}(\boldsymbol{Z^{\Delta}} \Downarrow S)$ more precise or incomparable with $\boldsymbol{Z^{\Delta}}$, so that $\boldsymbol{Z^{\Delta}} \sqcap F^{\nabla\Delta}(\boldsymbol{Z^{\Delta}} \Downarrow S)$ is an improved result.

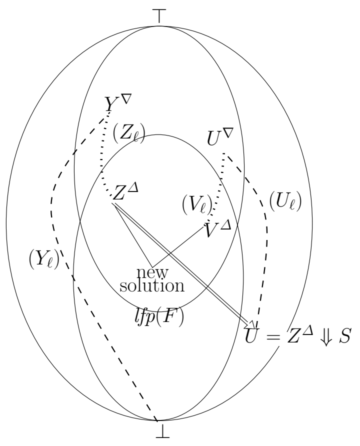
For instance, and following the intuition of §3.1, in our example 1.b (Fig. 3), we should choose $S = \{\nu_6\}$ and restart increasing and decreasing sequences from $\boldsymbol{Z^{\Delta}} \Downarrow S$, i.e., a vector $\boldsymbol{U}$ such that

$$U_k = (\text{if } k = 6 \text{ then } ([0, 99], [0, 0]) \\ \text{else if } k = 1 \text{ then } \top \text{ else } \bot)$$

With this choice, $F^{\nabla\Delta}(\boldsymbol{U})$ is the best possible result (the least fixpoint of F), shown by the opposite vector.

	i	j
ν_3	[0,100]	
ν_7	[0,99]	[0,100]
ν_9	[0,99]	[1,100]
ν_{11}	[1,100]	[100,100]
ν_5	[100,100]	

The opposite figure shows the new sequences: the classical solution Z^{Δ} is projected on $U = Z^{\Delta} \Downarrow S$ (generally not a pre-fixpoint), from which a new increasing sequence (U_ℓ) and a new decreasing sequence (V_ℓ) provide a new limit V^{Δ}. The improved solution is the greatest lower bound $Z^{\Delta} \sqcap V^{\Delta}$.



Choice of improving components. A component $Z^{\Delta}_{j_\ell}$ is likely to improve the solution if it is strictly smaller than Z^{Δ}_i. It may happen for two reasons:

- either some "initial states" in Y^0_i have been left on the paths from ν_i to $\nu_{j\ell}$; this case is not interesting since Y^0_i clearly consists of states which won't be shown to be unreachable. They will belong to any solution.
- or, during its propagation along these paths, Z^{Δ}_i has been "truncated" by some condition; this is the interesting case which can add some information.

As a consequence, ν_{j_ℓ} will be selected in S if

$$Y^0_i \sqcup Z^{\Delta}_{j_\ell} \sqsubset Z^{\Delta}_i \quad \text{(Criterion 1)}$$

Moreover, it is useless to propagate again Y^0_i, which already provided the existing result Z^{Δ}. So, ν_{j_ℓ} will be selected in S only if

$$Z^{\Delta}_{j_\ell} \not\sqsubseteq Y^0_i \quad \text{(Criterion 2)}$$

Now, let's consider our example 2.b. At convergence, no predecessor of the widening node satisfy our criteria. However, the widening node is preceded by a succession of two junction nodes, the first one being associated with an obviously interesting invariant: $n \in [0, 60]$. A simple change in the CFG taking into account the associativity of junction, would bring this node in the predecessors of the widening node. So, it can be useful to look for "improving nodes" further upstream the widening node. We change our selection process as follows:

A node ν_j will be selected in S if

- it precedes a junction node (possibly ν_i) (C0)
- $Y^0_i \sqcup F_{j,i}(Z^{\Delta}_j) \sqsubset Z^{\Delta}_i$ (C1)
- $F_{j,i}(Z^{\Delta}_j) \not\sqsubseteq Y^0_i$ (C2)

The criterion (C0) above comes from the fact that, during the descending sequence, only junction nodes lose information. The other two criteria generalise our preceding Criteria 1 and 2, by allowing the candidate node ν_j not to be an immediate predecessor of ν_i. Note that, in our example 2.b, the selected node ν_j is such that $F_{j,i} = Id$. However, our criteria allow also some statement nodes to be on the path from ν_j to ν_i.

3.4 General case

The case of an SCC with several widening nodes is very similar. Only the definition of $\mathcal{P}_{i,j}$ needs to be modified: it denotes the set of nodes belonging to an elementary path in the CFG going from i to j *without going through a widening node*: only the extremities ν_i and/or ν_j can belong to W. Our criteria for selecting the improving nodes are essentially unchanged: a node ν_j will be selected in S if

– it precedes a junction node (C0)
– there exists a widening node ν_i such that
  - $Y_i^0 \sqcup F_{j,i}(Z_j^{\Delta}) \sqsubset Z_i^{\Delta}$ (C1)
  - $F_{j,i}(Z_j^{\Delta}) \not\sqsubseteq Y_i^0$ (C2)

and the definition of $Z^{\Delta} \Downarrow S$ is unchanged.

4 A more illustrative example

Our two running examples are properly analysed with the proposed method. They would be also solved with a smart choice of "thresholds" [Hal93,BCC+03,LCJG11] for limiting the widening during the increasing sequence. Let us consider now another example to show that our method can discover constraints that don't appear as conditions in the program. The example is a rather ad-hoc modification of Example 1.b by variable change.

```
$i := 0$;
while $i < 4$ do {
  $j := 0$;
  while $j < 4$ do { $i := i + 1; j := j + 1;$}
  $i := i - j + 1$;
}
```

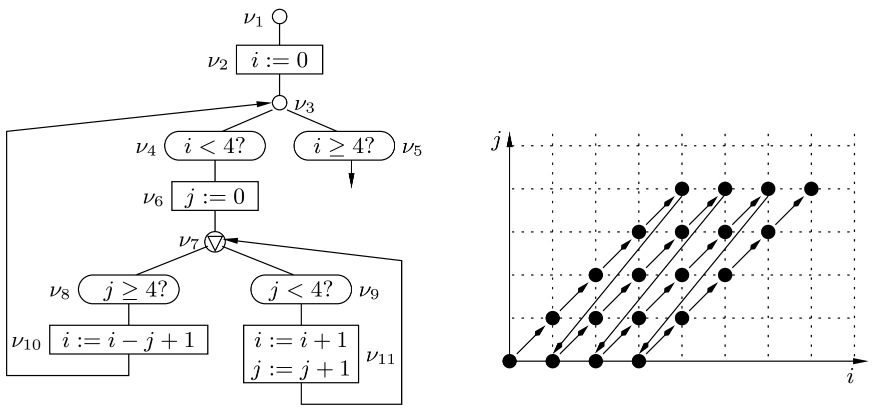


Fig. 4. Example 3

Fig. 4 shows the corresponding CFG together with the set of variable states traversed by the execution at ν_7. Fig. 5 shows the abstract values at widening node ν_7, during a polyhedra analysis:

– at the end of the classical increasing sequence, we get $(0 \leq j \leq i)$;
– the decreasing sequence reaches a fixpoint Z^{Δ} in one step, giving $(0 \leq j \leq i, j \leq 4)$;
– at node ν_6, we have $Z_6^{\Delta} = (0 \leq i \leq 3, j = 0)$, which satisfies all our criteria; we start again from $Z^{\Delta} \Downarrow \{\nu_6\}$;
– after a new increasing sequence, we get $(0 \leq j \leq i \leq j + 3)$;
– the decreasing sequence converges in one step, giving the best possible polyhedral invariant $(0 \leq j \leq i \leq j + 3, j \leq 4)$.

Notice that the constraint $i \leq j + 3$ doesn't appear in the program, so it could not be chosen as a "threshold" for widening.

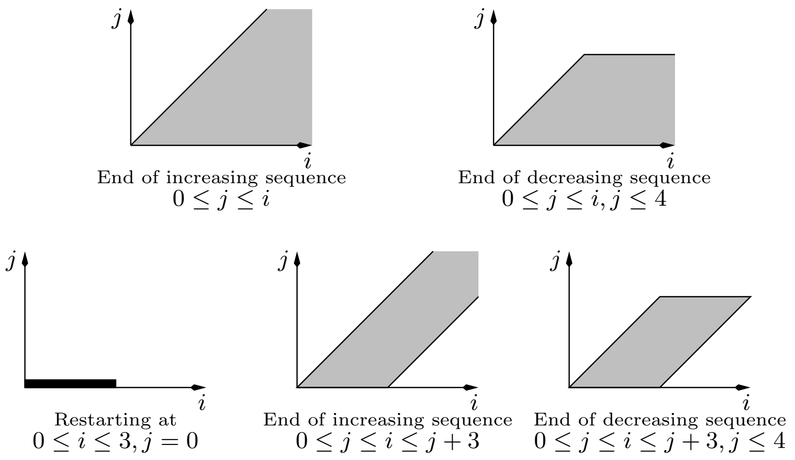


Fig. 5. Analysis of Example 3

5 Some improvements

Of course, our method involves the computation of new iteration sequences, which may look expensive. We propose here two simple improvements to limit the cost of this computation.

The first one is obvious: since the result, in general, is the greatest lower bound of the classical solution Z^{Δ} and the new limit V^{Δ}, one can intersect with Z^{Δ} each term of the new increasing sequence. In our Example 3, this would force the convergence of the new increasing sequence directly on the fixpoint $(0 \leq j \leq i \leq j + 3, j \leq 4)$, without need of a decreasing sequence.

The second improvement is a compromise: it saves computation, but may lose precision (although we did not find any example where it happens). The selected components $\{Z_j^{\Delta} \mid \nu_j \in S\}$ are intended to lead to improvements of other components, but in general, they are not supposed to be improved by the new iteration. With this idea in mind, we can limit the computation of the new iteration to the part of the CFG which doesn't influence only the components in S. More precisely, a component at node ν_k must be computed again only if there is a path from ν_k to a widening node which doesn't intersect S. In our Example 3, this would limit the new iteration to the subgraph $\{\nu_7, \nu_9, \nu_{11}\}$, since any path from $\{\nu_8, \nu_{10}, \nu_3, \nu_4\}$ to the widening node ν_7 goes through the selected node ν_6.

6 Experimental results

Our technique has been implemented inside our prototype static analyser, called Pagai, which computes numerical invariants in programs expressed in the LLVM internal representation [LA04]. In this representation, a function is a graph of basic blocks. The analyser takes as input such an LLVM file (that can be obtained from a C, C++, Fortran program by llvm-gcc or clang), and outputs for each basic block a numerical inductive invariant over the variables that are live at the head of this block. We can choose among several abstract domains : convex polyhedra, octagons, intervals, etc. through the Apron library [JM09]. Since Pagai is an intra-procedural analyser, we can apply function inlining to obtain more precise results. We can also apply some LLVM optimisation passes, such as loop unrolling, or promoting memory variables to registers (mem2reg), in order to increase precision.

In the current state of our implementation, we only choose as improving components basic-blocks that are direct predecessors of a widening point, i.e., we don't apply our criteria C0 and C1 in their full generality. We apply only the first improvement proposed in §5: during the new increasing sequence, we intersect our new result with Z^{Δ} at each step.

We compared our technique with the classical abstract interpretation with standard widening/narrowing, on a variety of benchmarks.

The benchmark from the Mälardalen WCET research group[1] contains interesting programs such as sorts, matrix transformations, fft, etc. These programs have been instrumented with a variable that counts the number of instructions being executed. These 98 functions have been analysed, using the polyhedra abstract domain. For 69 functions, the results of the new method are the same as the classical one, with a negligible time overhead (1.008 factor). For the other 29 functions, the new methods gives better results at 35% of widening points, with a 1.76 overhead factor. So, on this benchmark, not only the results are better for a significant subset of functions, but the new method costs almost nothing when it doesn't improve the results.

However, these encouraging experimental conclusions should not be overestimated: on a benchmark made of various highly used GNU functions (e.g., a2ps, gawk, gnuchess, gnugo, grep, gzip, lapack, make, sed and tar), the results are improved only at 4.14% of the widening points, with an overhead factor of 1.56 even on non improved functions.

[1] `www.mrtc.mdh.se/projects/wcet/benchmarks.html`

7 Conclusion

A claim of the present paper is that the information about the end of executions can be as rich, complex and useful than the one derived from their beginning. To permit a better gathering of this information, we presented a method to improve the solution obtained by classical analysis: this solution is projected on some of its components, and the result of the projection is used to start a new pair of increasing and decreasing sequences.

The method is independent of the abstract lattice, it is compatible with any smart widening operator and any iteration strategy.

Acknowledgement: We are indebted to Laure Gonnord and David Monniaux for having put our attention on the problem — and specially on Examples 1 and 2 — and for helpful discussions.

References

[BCC$^+$03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation, PLDI 2003*, pages 196–207, San Diego (Ca.), June 2003.

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast acceleration of symbolic transition systems. In *CAV'03*, pages 118–121, Boulder (Colorado), July 2003. LNCS 2725, Springer-Verlag.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using Presburger arithmetic. In *Computer Aided Verification, CAV'97*. LNCS 1254, Springer Verlag, June 1997.

[BHRZ03] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin.

[Bou93] F. Bourdoncle. Efficient chaotic iterations strategies with widening. In *International Conference on Formal Methods in Programming and their applications*, pages 128–141. LNCS 735, Springer Verlag, 1993.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *2nd Int. Symp. on Programming*. Dunod, Paris, 1976.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

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

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, Vancouver (B.C.), 1998. LNCS 1427, Springer Verlag.

[CZ11] A. Cortesi and M. Zanioli. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures*, 37(1):24–42, 2011.

[GH06] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In Kwangkeun Yi, editor, *13th International Static Analysis Symposium, SAS'06*, Seoul, Korea, August 2006. LNCS 4134, Springer Verlag.

[Gou01] Eric Goubault. Static analyses of the precision of floating-point operations. In *SAS*, pages 234–259, 2001.

[GR06] D. Gopan and T. Reps. Lookahead widening. In *CAV'06*, Seattle, 2006.

[GR07] Denis Gopan and Thomas W. Reps. Guided static analysis. In *SAS 2007*, pages 349–365. LNCS 4634, Springer Verlag, 2007.

[GS07] T. Gawlitza and H. Seidl. Precise fixpoint computation through strategy iteration. In *ESOP 2007*, pages 300–315. LNCS 4421, Springer-Verlag, March 2007.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag.

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[JM09] B. Jeannet and A. Miné. Apron: a library of numerical abstract domains for static analysis. In *CAV'09*, pages 661–667, Grenoble, France, July 2009. LNCS 5643, Springer Verlag.

[LA04] C. Lattner and V. Adve. LLVM: a compilation framework fopr lifelong program analysis & transformation. In *CGO'04*, pages 75–86, Washington, DC, August 2004. IEEE Computer Society.

[LCJG11] L. Lakhdar-Chaouch, B. Jeannet, and A. Girault. Widening with thresholds for programs with complex control graphs. In *Automated Technology for Verification and Analysis, ATVA 2011, Taipei, Taiwan*, pages 492–502. LNCS 6996, Springer Verlag, 2011.

[Min04] A. Miné. *Weakly relational numerical abstract domains.* PhD thesis, Ecole Polytechnique, 2004.

[ML11] D. Monniaux and J. Le Guen. Stratified static analysis based on variable dependencies. In *Third International Workshop on Numerical and Symbolic Abstract Domains*, Venice, September 2011.

[PGM03] S. Putot, E. Goubault, and M. Martel. Static analysis-based validation of floating-point computations. In *Numerical Software with Result Verification*, pages 306–313. LNCS 2991, Springer Verlag, 2003.

[SSM04] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constraint-based linear relations analysis. In *International Symposium on Static Analysis, SAS'2004*, pages 53–68. LNCS 3148, Springer Verlag, 2004.

[SSM05] S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In R. Cousot, editor, *6th International Conference on Verification, Model-checking, and Abstract Intepretation, VMCAI'05*, Paris, January 2005. LNCS 3385, Springer Verlag.

[SW04] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In *TACAS'04*, pages 280–295, Barcelona, 2004.

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*, pages 88–97, Vancouver, June 1998. LNCS 1427, Springer-Verlag.

[WYGI07] C. Wang, Z. Yang, A. Gupta, and F. Ivančić. Using counterexamples for improving the precision of reachability computation with polyhedra. In *CAV 2007*, pages 352–365. LNCS 4590, Springer-Verlag, July 2007.