Narrowing operators on template abstract domains

Gianluca Amato, Simone Di Nardo Di Maio, Maria Chiara Meo, and Francesca Scozzari

Dipartimento di Economia, Università di Chieti-Pescara, Italy

Abstract. In the theory of abstract interpretation, narrowing operators are used to improve the precision of the analysis after a post-fixpoint has been reached. This is especially true on numerical domains, since they are generally endowed with infinite descending chains which may lead to a non-terminating analysis in the absence of narrowing. We provide an abstract semantics which improves the analysis precision and show that, for a large class of numerical abstract domains over integer variables (such as intervals, octagons and template polyhedra), it is possible to avoid infinite descending chains and omit narrowing. Moreover, we propose a new family of narrowing operators for real variables which improves the analysis precision.

1 Introduction

Computing a static analysis in the framework of *abstract interpretation* [5, 6] typically amounts to solve a set of equations describing the program behavior. Given a program to be analyzed, we associate to each control point i of the program an unknown[1] x_i and an equation $x_i = \Phi_i(x_1, \ldots, x_n)$, where Φ_i is a monotone, state-transition operator . The unknowns $x_1, \ldots, x_n$ range over an *abstract domain* A, which encodes the property we want to analyze. An element of A is called *abstract object* and represents a set of concrete states.

We are interested in finding the (least) solution, over the domain A, of the set of equations $\Phi = (\Phi_1, \ldots, \Phi_n)$ associated to the program to be analyzed. The abstract interpretation framework ensures that any solution of the set of equation correctly approximates the concrete behavior of the program, and the smaller the solution, the more precise is the result of the analysis. In theory, the least solution of the system can be exactly computed as the limit of a Kleene iteration, starting from the least element of A^n. In practice, such a method can be unfeasible, since many abstract domains exhibit infinite ascending chains, and thus the computation may not terminate. Moreover, even for finite abstract domains, it may happen that the ascending chains are very long, and this method would result impractical.

[1] We use the terms *variable* to denote a variable in the program, and *unknown* to denote a variable in the data-flow equations.

The standard method to perform the analysis is to compute an approximation of the least solution of the system of equations using widening and narrowing operators [4, 7]. For specific abstract domains or for restricted classes of programs, we may find in the literature alternatives, such as acceleration operators [11] and strategy/policy iteration [3, 9, 10], but these methods are not generally applicable and their complexity may be impractical.

A widening, generally denoted by $\triangledown$, is a binary operator over the abstract domain A such that:

- it is an upper bound;
- when used in equations of the kind $x_i = x_i \triangledown \Phi(x_1, \ldots, x_n)$, it precludes the insurgence of infinite ascending chains for x_i.

The widening operator compares the value of x_i in the previous iteration with its value in the current iteration and, in some cases, returns an approximated value. Widening is used to ensure the termination of the analysis, while introducing a loss in precision. This is realized by replacing some of the original equations $x_i = \Phi_i(x_1, \ldots, x_n)$ with $x_i = x_i \triangledown \Phi_i(x_1, \ldots, x_n)$. The replacement may involve all unknowns or, more commonly, only the ones corresponding to loop heads. Applying widening in this way ensures the termination of a Kleene iteration, but we only get a post-fixpoint of the function $\Phi = (\Phi_1, \ldots, \Phi_n)$, instead of the least one.

Once we reach a post-fixpoint, we can start a new Kleene iteration, giving origin to a descending chain which improves the result of the analysis. However, due to infinite descending chains in the abstract domain, the descending iteration might not terminate. The next example[2] shows this phenomenon using the abstract domain $\mathsf{Int}_{\mathbb{Z}}$ of intervals over integer numbers [4], defined as:

$$\mathsf{Int}_{\mathbb{Z}} = \{[l, u] \subseteq \mathbb{Z} \mid l \leq u \in \mathbb{Z} \cup \{-\infty, \infty\}\} \cup \{\emptyset\},$$

where $\emptyset$ denotes the empty set of concrete states, i.e., an unreachable control point. The standard widening on intervals [4] is defined as follows:

$$\begin{aligned} \emptyset \triangledown I &= I \\ I \triangledown \emptyset &= I \\ [l_1, u_1] \triangledown [l_2, u_2] &= [l', u'] \end{aligned}$$

where

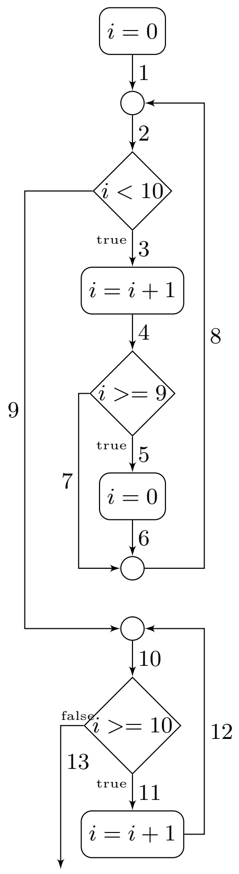
$$l' = \begin{cases} l_1 & \text{if } l_1 \leq l_2 \\ -\infty & \text{otherwise} \end{cases} \qquad u' = \begin{cases} u_1 & \text{if } u_1 \geq u_2 \\ +\infty & \text{otherwise} \end{cases}$$

Essentially, it works by preserving stable bounds and removing unstable ones. For instance, $[0, 3] \triangledown [0, 4] = [0, \infty]$. In this way, infinite ascending chains are precluded.

[2] To the best of our knowledge, this is the first example in the literature which shows a program analysis iterating over an infinite descending sequence in an integer numerical domain.

```
i=0
while(i<10) {
  i=i+1
  if(i>=9)
    i=0
}
while(i>=10) {
  i=i+1
}
```

(a) Program



(b) Flowchart

$$
\begin{aligned}
x_1 &= [0, 0] \\
x_2 &= x_1 \vee x_8 \\
x_3 &= x_2 \wedge [-\infty, 9] \\
x_4 &= x_3 + [1, 1] \\
x_5 &= x_4 \wedge [9, \infty] \\
x_6 &= [0, 0] \\
x_7 &= x_4 \wedge [-\infty, 8] \\
x_8 &= x_6 \vee x_7 \\
x_9 &= x_2 \wedge [10, \infty] \\
x_{10} &= x_9 \vee x_{12} \\
x_{11} &= x_{10} \wedge [10, \infty] \\
x_{12} &= x_{11} + [1, 1] \\
x_{13} &= x_{10} \wedge [-\infty, 9]
\end{aligned}
$$

(c) Equation system

Fig. 1. The example program doubleLoop.

Example 1. Consider the example program doubleLoop in Fig. 1(a), and the corresponding flowchart and set of equations in Fig. 1(b) and 1(c). We perform the analysis using the integer interval domain $\mathsf{Int}_{\mathbb{Z}}$ with the standard widening. Therefore, we replace the second and the tenth equation in Fig.1(c) with

$$
\begin{aligned}
x_2 &= x_2 \mathbin{\nabla} (x_1 \vee x_8) \\
x_{10} &= x_{10} \mathbin{\nabla} (x_9 \vee x_{12}) \ .
\end{aligned}
$$

Note that these two equations correspond to the loop joins. We assume to follow a work-list based iteration sequence, although the result is analogous with other iteration schemas.

The first time x_2 is considered, we have $x_1 = [0, 0]$ and $x_2 = x_8 = \emptyset$. Widening does not trigger and x_2 gets updated to $x_2 := x_1 \vee x_8 = [0, 0]$. However, the second time x_2 is considered we have $x_8 = [1, 1]$, hence $x_1 \vee x_8 = [0, 1]$, which is widened to $[0, +\infty]$. This eventually leads to $x_9 := [10, +\infty]$, $x_{10} := [10, +\infty]$ and $x_{12} := [11, +\infty]$ which is a post-fixpoint and the result of the ascending phase of the analysis.

Starting from the post-fixpoint, we continue to evaluate the semantic equations, without applying neither widening nor narrowing, thus using the original equations $x_2 = x_1 \vee x_8$ and $x_{10} = x_9 \vee x_{12}$. We get a descending sequence, which turns out to be infinite. In fact, the first time x_2 is re-evaluated, we have

$$x_2 := x_1 \vee x_8 = [0, 0] \vee [0, 8] = [0, 8]$$

which leads to $x_9 := \emptyset$. When we evaluate the equations in the second while loop, we get

$$x_{10} := x_9 \vee x_{12} = \emptyset \vee [11, +\infty] = [11, +\infty]$$

and $x_{12} = [12, +\infty]$. At the second iteration we get

$$x_{10} := x_9 \vee x_{12} = \emptyset \vee [12, +\infty] = [12, +\infty]$$

and $x_{12} := [13, +\infty]$. It is immediate to see that, while keeping on iterating, the values computed at the control point x_{10} are $[11, +\infty]$, $[12, +\infty]$, $[13, +\infty]$, $[14, +\infty]$, ... in an infinite descending sequence, whose limit is the empty set. □

It is worth noting that, in the previous example, the existence of an infinite descending sequence depends on the fact that the second while loop is unreachable, although the initial ascending phase of the analysis computes a non-empty over approximation. This leads to a descending sequence whose limit is the empty set. This situation is not peculiar of our example. On the contrary, we will show that this is the only way infinite descending sequences may arise in the integer interval domain.

To avoid the insurgence of infinite descending chains, we may stop the descending iteration at an arbitrary step, still obtaining a post-fixpoint, or we may use a narrowing operator. Narrowing, generally denoted by $\vartriangle$, is a binary operator on a abstract domain A such that:

– $a_1 \vartriangle a_2$ is only defined when $a_2 \leq a_1$;
– it holds that $a_2 \leq a_1 \vartriangle a_2 \leq a_1$;
– when used in equations of the kind $x_i = x_i \vartriangle \Phi_i(x_1, \ldots, x_n)$, it precludes the insurgence of infinite descending chains for x_i.

The standard narrowing for intervals [4], for example, is defined as:

$$I \vartriangle \emptyset = \emptyset$$
$$[l_1, u_1] \vartriangle [l_2, u_2] = [l', u']$$

where

$$l' = \begin{cases} l_2 & \text{if } l_1 = -\infty \\ l_1 & \text{otherwise} \end{cases} \qquad u' = \begin{cases} u_2 & \text{if } u_1 = +\infty \\ u_1 & \text{otherwise} \end{cases}$$

Essentially, it works by refining only unbounded extremes. For instance, $[0, \infty] \vartriangle [0, 10] = [0, 10]$ but $[0, 10] \vartriangle [0, 9] = [0, 10]$. Let us reconsider Example 1 and show what happens when we use narrowing in the descending phase.

Example 2. Consider the same program, flowchart and equations of Example 1, together with the result of the analysis after the ascending phase. We now replace the equations for x_2 and x_{10} with $x_2 = x_2 \vartriangle (x_1 \vee x_8)$ and $x_{10} = x_{10} \vartriangle (x_9 \vee x_{12})$ and start a descending iteration.

When the second equation is first re-evaluated, the current value for x_2 is $[0, +\infty]$, hence the standard narrowing allows to change $+\infty$ into 8, and we have $x_2 := [0, 8]$ as for the case without narrowing. However, when x_{10} is evaluated for the first time in the decreasing sequence, we have $x_{10} := [10, +\infty] \vartriangle [11, +\infty] = [10, +\infty]$: the standard narrowing precludes further improvements on the second loop. The descending sequence terminates at the cost of a big loss of precision, since we are not able to detect anymore that control points 10–12 are unreachable. $\square$

In the rest of the paper, we will show that narrowing for the integer interval domain is superfluous, and may be removed upon adopting a slightly different semantic operator for loop joins which preserves unreachability. Moreover, we generalize this result to all the template abstract domains over integer variables.

Furthermore, we show that such a result can be used to design a more precise narrowing on template abstract domain over reals, exploiting the fact that we never get infinite descending chains of integer intervals.

2 Narrowing on intervals of integers

Example 1 shows an analysis which leads to an infinite descending chain of intervals. In particular, the chain is $[11, +\infty], [12, +\infty], [13, +\infty], \ldots$ and its limit is the empty set. It turns out that the only infinite descending chains of intervals are of the kind

$$[n_0, +\infty], [n_1, +\infty], [n_2, +\infty], \ldots$$

or

$$[-\infty, -n_0], [-\infty, -n_1], [-\infty, -n_2], \ldots$$

where $\{n_i\}_{i \in \mathbb{N}}$ is an infinite descending chain of integers. The limit of all these chains is the empty set.

Proposition 3. *Let $\{I_i\}_{i \in \mathbb{N}}$ be an infinite descending chain of integer intervals. Then $\sqcap_{i \in \mathbb{N}} I_i = \emptyset$.*

In the rest of the paper we assume to deal only with structured programs, whose flowchart is *reducible*. Intuitively, this means that every loop has a single well defined entry point.

Assume *loop* is the entry point of a loop and its corresponding equation is $x_{loop} = x_{in} \vee x_{back}$, where *in* is the edge in the flowchart which comes from outside the loop and *back* the back edge. Since in a reducible flowchart the entry point of a loop dominates all the nodes inside the loop, if control point *in* is unreachable (i.e., $x_{in} = \emptyset$ in the interval domain) the same holds for control point *loop*.

Therefore, we may change the abstract semantics of the program by replacing each equation corresponding to a loop join $x_{loop} = x_{in} \vee x_{back}$ with $x_{loop} = x_{in} \vee^{\emptyset} x_{back}$, where $\vee^{\emptyset}$ is a left-strict variant of the join operator defined as:

$$I_1 \vee^{\emptyset} I_2 = \begin{cases} \emptyset & \text{if } I_1 = \emptyset \\ I_1 \vee I_2 & \text{otherwise} \end{cases} \tag{1}$$

The new set of equations is correct (again, only on reducible flowcharts) and more precise. Moreover, during the descending phase of the analysis, narrowing is not required to achieve termination. Actually, assume that an infinite descending chain arises during the descending phase. Let *loop* be one of the outermost loop heads whose variable x_{loop} infinitely decreases. In the presence of left-strict joins, this leads to a contradiction. The equation of x_{loop} is $x_{loop} = x_{in} \vee^{\emptyset} x_{back}$. The value of x_{in} is definitively constant. Once it reaches its definitive value $\bar{x}_{in}$, we may have only two cases:

– if $\bar{x}_{in} = \emptyset$, then the first time x_{loop} is re-evaluated we have $x_{loop} := \emptyset$ and x_{loop} cannot descend anymore, contradicting our hypothesis;
– if $\bar{x}_{in} \neq \emptyset$, then $x_{loop} \geq \bar{x}_{in}$ always, and therefore it cannot descend infinitely, due to Proposition 3.

The considerations above hold for any numerical abstract domain A with a distinguished value denoting unreachability. In the following, we will refer to such a distinguished value as $\emptyset$, which is the common notation in all the numerical domains in the literature.

This discussion leads therefore to the following results.

Theorem 4. *Assume given a numerical abstract domain A with a distinguished value $\emptyset$ denoting unreachability. Assume we have a system of data-flow equations Φ generated by a structured program whose loop head nodes are of the form $x_{loop} = x_{in} \vee x_{back}$. Then, replacing $\vee$ with $\vee^{\emptyset}$ in all the loop heads, the new set of data-flow equations is still correct.*

Theorem 5. *In the hypothesis of Theorem 4, assume A is the abstract domain of integer intervals. Then every iteration strategy on the equations in Φ starting from a post-fixpoint of Φ leads to a finite sequence.*

Note that a descending sequence without narrowing always leads to a fixpoint of the equation system, instead of a post-fixpoint.

Some of the restrictions of Theorem 4 may be easily lifted. For example, if a loop join node has equation

$$x_{loop} = x_{in_1} \vee \cdots \vee x_{in_u} \vee x_{back_1} \vee \cdots \vee x_{back_v} \ ,$$

where all the edges in_i come from outside the loop and all the $back_j$'s are back edges, we may use left-strict join in this way:

$$x_{loop} = (x_{in_1} \vee \cdots \vee x_{in_u}) \vee^{\emptyset} (x_{back_1} \vee \cdots \vee x_{back_v}) \enspace .$$

Moreover, it is possible to extend Theorem 4 to non reducible flowcharts, provided we only apply the left-strict join to the loop heads that dominate the sources of the back edges.

When avoiding narrowing, we may find programs whose descending chain is arbitrarily long, but finite. The next example shows this phenomenon.

Example 6. Consider the example program doubleLoop2 in Fig. 2(a), and the corresponding flowchart and set of equations in Fig. 2(b) and 2(c). We first perform the analysis using the integer interval domain $\mathsf{Int}_{\mathbb{Z}}$ with the standard widening and narrowing and then we recompute the analysis without narrowing.

In the ascending phase we use widening on the join loops: $x_2 = x_2 \nabla (x_1 \vee^{\emptyset} x_4)$ and $x_6 = x_6 \nabla (x_5 \vee^{\emptyset} x_8)$. The post-fixpoint is:

$$\begin{array}{lll} x_1 = [0,0] & x_4 = [1,11] & x_7 = [-\infty,100] \\ x_2 = [0,\infty] & x_5 = [11,\infty] & x_8 = [-\infty,99] \\ x_3 = [0,10] & x_6 = [-\infty,\infty] & x_9 = [101,\infty] \end{array}$$

Now we start the descending phase with the standard narrowing, using the equations $x_2 = x_2 \vartriangle (x_1 \vee^{\emptyset} x_4)$ and $x_6 = x_6 \vartriangle (x_5 \vee^{\emptyset} x_8)$. When we first apply narrowing in the second equation, we get:

$$x_2 = x_2 \vartriangle (x_1 \vee^{\emptyset} x_4) = [0,\infty] \vartriangle [0,11] = [0,11]$$

and therefore $x_5 = [11,11]$. We now apply narrowing in the sixth equation:

$$x_6 = x_6 \vartriangle (x_5 \vee^{\emptyset} x_8) = [-\infty,\infty] \vartriangle [-\infty,99] = [-\infty,99]$$

and therefore we have $x_7 = [-\infty,99]$, $x_8 = [-\infty,98]$ and $x_9 = \emptyset$, which is the fixpoint.

We now recompute the descending phase without narrowing, using the equations

$$\begin{aligned} x_2 &= x_1 \vee^{\emptyset} x_4 \\ x_6 &= x_5 \vee^{\emptyset} x_8 \enspace . \end{aligned}$$

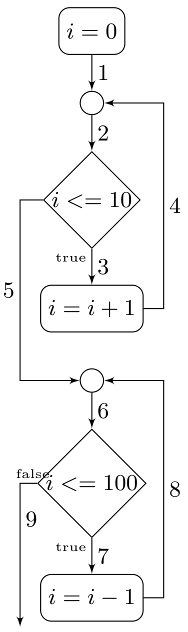
The first while loop behaves as before with $x_5 = [11,11]$. Now we enter the second while loop. The first iteration is the same as before using narrowing, and we get:

$$\begin{array}{ll} x_6 = [-\infty,99] & x_8 = [-\infty,98] \\ x_7 = [-\infty,99] & x_9 = \emptyset \end{array}$$

But now we are able to continue the descending phase, which is:

```
i=0
while(i<=10) {
  i=i+1
}
while(i<=100) {
  i=i−1
}
```

(a) Program



(b) Flowchart

$$
\begin{aligned}
x_1 &= [0, 0]\\
x_2 &= x_1 \vee^{\emptyset} x_4\\
x_3 &= x_2 \wedge [-\infty, 10]\\
x_4 &= x_3 + [1, 1]\\
x_5 &= x_2 \wedge [11, \infty]\\
x_6 &= x_5 \vee^{\emptyset} x_8\\
x_7 &= x_6 \wedge [-\infty, 100]\\
x_8 &= x_7 - [1, 1]\\
x_9 &= x_6 \wedge [101, \infty]
\end{aligned}
$$

(c) Equation system

Fig. 2. The example program doubleLoop2.

	2^{ns} descending iteration	3^{rd} d. i.	4^{th} d. i.	. . .	last d. i.
x_6	$[-\infty, 98]$	$[-\infty, 97]$	$[-\infty, 96]$	. . .	$[-\infty, 11]$
x_7	$[-\infty, 98]$	$[-\infty, 97]$	$[-\infty, 96]$	. . .	$[-\infty, 11]$
x_8	$[-\infty, 97]$	$[-\infty, 96]$	$[-\infty, 95]$	. . .	$[-\infty, 10]$

Note that, by continuing the descending phase till the fixpoint, we are able to detect that the guard in the second while loop is over dimensioned, since the variable i never reaches the value 100. □

2.1 Template abstract domains

The above result on intervals can be extended to the whole family of template abstract domains. We call template abstract domains those numerical domains where the coefficients of the allowed constraints are fixed in advance, before starting the analysis. Most important template abstract domains are the domain of intervals (also called box domain) [4], octagons [13] and template polyhedra [14]. Non-template abstract domains are, among others, polyhedra [8] and two-variable for linear inequality [15].

All the template abstract domains may be described using a fixed matrix which describes the constraints and any abstract object o is a subset of $\mathbb{R}^n$ (or $\mathbb{Z}^n$ if working with integer variables) of the form $o = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\}$ where A is the constraint matrix, $\boldsymbol{l}$ and $\boldsymbol{u}$ are, respectively, the lower and upper bounds.

A box is an abstract object where A is the identity matrix. Octagons are those objects where the coefficient matrix A allows constrains of the form $\pm x \pm y \leq c$. Finally, template polyhedra are those objects where the coefficient matrix A is arbitrary but fixed a priori.

Under the hypothesis of Theorem 4, it is possible to extend Theorem 5 to all the template abstract domains. In fact, given a narrowing operator on intervals, we can immediately define a corresponding component-wise narrowing operator on any template abstract domain. We first show that template abstract domains over integers enjoy a property similar to Prop. 3. Note that a template domain over integers only needs to have integer bounds, while the coefficients of the constraint matrix may be reals.

Proposition 7. *Let A be a template abstract domain over integers and $\{I_i\}_{i\in\mathbb{N}}$ be an infinite descending chain of objects $I_i \in A$. Then $\sqcap_{i\in\mathbb{N}} I_i = \emptyset$, where $\emptyset$ is a distinguished value of A denoting unreachability.*

Exploiting the above proposition and Theorem 4, we can prove a result analogue to Theorem 5 which, in presence of a left-strict join, allows us to avoid narrowing, still guaranteeing termination.

Theorem 8. *In the hypothesis of Theorem 4, assume A is a template abstract domain over integers. Then every iteration strategy on the equations in Φ starting from a post-fixpoint of Φ leads to a finite sequence.*

3 Narrowing on reals

The left-strict join we have introduced for integer domains may also be used with abstract domains over real variables. This improves the precision of the analysis, but does not ensure that the descending phase will terminate. This depends on the fact that, once we admit real variables, we can have infinite descending chains whose limit is not the empty set. Nonetheless, in this case the left-strict join may be exploited to define a narrowing more precise than the standard one.

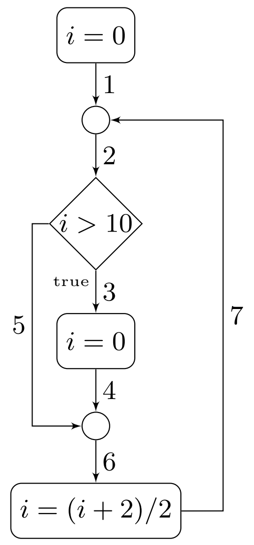
The next example shows that on the standard interval domain $\mathsf{Int}_{\mathbb{R}}$ for real variables, the descending phase of the analysis may lead to an infinite descending chain whose limit is not the empty set. We recall that

$$\mathsf{Int}_{\mathbb{R}} = \{[l,u] \subseteq \mathbb{R}^n \mid l \leq u \in \mathbb{R} \cup \{-\infty, \infty\}\} \cup \{\emptyset\}.$$

Example 9. Consider the example program realLoop in Fig. 3(a), and the corresponding flowchart and equations in Fig. 3(b) and 3(c). The ascending phase

```
i=0
while(true) {
  if (i>10) {
    i=0
  }
  i=(i+2)/2
}
```

(a) Program



(b) Flowchart

$$
\begin{aligned}
x_1 &= [0, 0] \\
x_2 &= x_1 \vee^{\emptyset} x_7 \\
x_3 &= x_2 \wedge [10, +\infty] \\
x_4 &= [0, 0] \\
x_5 &= x_2 \wedge [-\infty, 10] \\
x_6 &= x_4 \vee x_5 \\
x_7 &= (x_6 + [2, 2])/2
\end{aligned}
$$

(c) Equation system

Fig. 3. The example program realLoop.

using left-strict join and standard widening, i.e., $x_2 = x_2 \triangledown (x_1 \vee^{\emptyset} x_7)$, reaches a post-fixpoint in two iterations.

	1st ascending iteration	2nd ascending iteration
x_1	$[0, 0]$	$[0, 0]$
x_2	$[0, 0]$	$[0, 0] \ \triangledown^{\emptyset} \ [1, 1] = [0, +\infty]$
x_3	$\emptyset$	$[10, +\infty]$
x_4	$[0, 0]$	$[0, 0]$
x_5	$[0, 0]$	$[0, 10]$
x_6	$[0, 0]$	$[0, 10]$
x_7	$[1, 1]$	$[1, 6]$

We now start from the post fixpoint a descending iteration without applying narrowing, using the original equation $x_2 = x_1 \vee^{\emptyset} x_7$.

	1st descending iteration	2nd descending iteration
x_1	$[0, 0]$	$[0, 0]$
x_2	$[0, 0] \vee^{\emptyset} [1, 6] = [0, 6]$	$[0, 0] \vee^{\emptyset} [1, 4] = [0, 4]$
x_3	$\emptyset$	$\emptyset$
x_4	$[0, 0]$	$[0, 0]$
x_5	$[0, 6]$	$[0, 4]$
x_6	$[0, 6]$	$[0, 4]$
x_7	$[1, 4]$	$[1, 3]$

At the next iterations, we obtain:

$$x_2 = [0, 3] \qquad x_7 = \left[1, \frac{5}{2}\right]$$
$$x_2 = \left[0, \frac{5}{2}\right] \qquad x_7 = \left[1, \frac{9}{4}\right]$$

and so on, without terminating. The fixpoint, which is $x_2 = [0, 2]$ and $x_7 = [1, 2]$, is not the empty set. □

Exploiting Proposition 3, we can define a new narrowing operator on intervals for real variables which refines successive descending iterations at the nearest integer, since we cannot have an infinite descending chain whose bounds are all integers.

Definition 10 (Narrowing on reals). *We define a narrowing operator $\triangle^1$ on $\mathsf{Int}_{\mathbb{R}}$ as follows:*

$$I \triangle^1 \emptyset = \emptyset$$
$$[l_1, u_1] \triangle^1 [l_2, u_2] = [l', u']$$

where

$$l' = \begin{cases} l_2 & \textit{if } l_1 = -\infty \\ \max(l_1, \lfloor l_2 \rfloor) & \textit{otherwise} \end{cases}$$

$$u' = \begin{cases} u_2 & \textit{if } u_1 = +\infty \\ \min(u_1, \lceil u_2 \rceil) & \textit{otherwise} \end{cases}$$

The new narrowing $\triangle^1$ refines infinite bounds to finite values, as the standard one, and refines finite bounds only to new integer values. Since infinite descending sequences on integer template domains are precluded by the use of left-strict joins, the descending sequence terminates.

Theorem 11. *The operator $\triangle^1$ is a narrowing operator on template domains when the loop join is left-strict.*

In the next example we compare the standard narrowing with the new narrowing on reals $\triangle^1$.

Example 12. We compute the descending chain of Example 9 using the standard narrowing on intervals. We start from the post fixpoint and use the equation $x_2 = x_2 \triangle (x_1 \vee^{\emptyset} x_7)$. At the first descending iteration we get

$$x_2 = [0, +\infty] \triangle ([0, 0] \vee^{\emptyset} [1, 6]) = [0, +\infty] \triangle [0, 6] = [0, 6] \ .$$

Note that we get exactly the same value as in the first descending iteration without narrowing. Therefore, we compute for the other unknowns exactly the same values, in particular $x_7 = [1,4]$. It is immediate to see that this is a fixpoint for the computation using the standard narrowing, since no more unbounded values appear. In fact, we have that

$$x_2 = x_2 \vartriangle (x_1 \vee^{\emptyset} x_7) = [0,6] \vartriangle [0,4] = [0,6] \ .$$

We now recompute the descending chain of Example 9 using the narrowing on reals $\vartriangle^1$ in Def. 10. The first descending iteration is the same as for the standard narrowing, and we get $x_2 = [0,6]$ and $x_7 = [1,4]$. In the second descending iteration we have

$$x_2 = x_2 \vartriangle^1 (x_1 \vee^{\emptyset} x_7) = [0,6] \vartriangle^1 [0,4] = [0,4]$$

and $x_7 = [1,3]$. In the third descending iteration we have

$$x_2 = [0,4] \vartriangle^1 [0,3] = [0,3]$$

and $x_7 = [1, \frac{5}{2}]$. This is the fixpoint, since

$$x_2 = [0,3] \vartriangle^1 \left[0, \frac{5}{2}\right] = [0,3] \ .$$

In this case, we get a result strictly more precise than with the standard narrowing. □

It is worth noting that $\vartriangle^1$ could be easily generalized by rounding numbers at the multiple of any strictly positive constant value $\delta \in \mathbb{R}$.

Definition 13 (δ-narrowing). *Let $\delta \in \mathbb{R}$ such that $\delta > 0$. We define a new narrowing on intervals of reals:*

$$I \vartriangle^{\delta} \emptyset = \emptyset$$
$$[l_1, u_1] \vartriangle^{\delta} [l_2, u_2] = [l', u']$$

where

$$l' = \begin{cases} l_2 & \textit{if } l_1 = -\infty \\ \max(l_1, \delta\lfloor l_2/\delta \rfloor) & \textit{otherwise} \end{cases}$$

$$u' = \begin{cases} u_2 & \textit{if } u_1 = +\infty \\ \min(u_1, \delta\lceil u_2/\delta \rceil) & \textit{otherwise} \end{cases}$$

The above narrowing produces a descending chain whose elements differ for a multiple of δ, which is fixed in advance. Since the limit of these chains is still the empty set, it is immediate to see that $\vartriangle^{\delta}$ in the above definition is a narrowing operator on intervals of reals. It generalizes $\vartriangle^1$ given in Definition 10. In fact, Def. 13 boils down to Def. 10 when $\delta = 1$. Moreover, it can be easily generalized to template abstract domains.

Theorem 14. *For any $\delta \in \mathbb{R}$ such that $\delta > 0$, the operator $\vartriangle^\delta$ is a narrowing operators on template abstract domains when the loop join is left-strict.*

The next example applies the new narrowing $\vartriangle^\delta$ to the program realLoop.

Example 15. We compute the descending chain for the example program realLoop in Fig. 3(a) using δ-narrowing with $\delta = \frac{1}{100}$. We get the following values for x_2:

$$[0,6], [0,4], [0,3], \left[0, \frac{5}{2}\right], \left[0, \frac{9}{4}\right], \left[0, \frac{213}{100}\right], \left[0, \frac{207}{100}\right], \left[0, \frac{204}{100}\right], \left[0, \frac{202}{100}\right], \left[0, \frac{201}{100}\right]$$

where the last one is the fixpoint. □

As an alternative, instead of rounding bounds to a multiple of δ, we may refine bounds with the new value only if the difference w.r.t. the previous value is greater than a given δ. We call this δ*-narrowing.

Definition 16 (δ*-narrowing). *Let $\delta \in \mathbb{R}$ such that $\delta > 0$. We define a new narrowing on intervals of reals:*

$$I \vartriangle^{\delta *} \emptyset = \emptyset$$
$$[l_1, u_1] \vartriangle^{\delta *} [l_2, u_2] = [l', u']$$

where

$$l' = \begin{cases} l_2 & \text{if } l_1 = -\infty \text{ or } l_2 - l_1 \geq \delta \\ l_1 & \text{otherwise} \end{cases}$$

$$u' = \begin{cases} u_2 & \text{if } u_1 = +\infty \text{ or } u_1 - u_2 \geq \delta \\ u_1 & \text{otherwise} \end{cases}$$

The above narrowing keeps iterating while the difference between two successive iterations is greater than δ. Since the limit of any such descending chain is still the empty set, we can prove that $\vartriangle^{\delta *}$ is a narrowing operator under the same hypothesis of Th. 14

Theorem 17. *For any $\delta \in \mathbb{R}$ such that $\delta > 0$, the operator $\vartriangle^{\delta *}$ is a narrowing operators on template domains when the loop join is left-strict.*

The next example shows the narrowing $\vartriangle^{\delta *}$ in the program realLoop.

Example 18. We compute the descending chain for the example program realLoop in Fig. 3(a) using δ*-narrowing with $\delta = \frac{1}{100}$. We get the following values for x_2:

$$[0,6], [0,4], [0,3], \left[0, \frac{5}{2}\right], \left[0, \frac{9}{4}\right], \left[0, \frac{17}{8}\right], \left[0, \frac{33}{16}\right], \left[0, \frac{65}{32}\right], \left[0, \frac{129}{64}\right]$$

where the last one is the fixpoint. □

4 Conclusion and related work

We believe the main contribution of this paper is a deeper theoretical understanding of termination issues during descending iterations within the framework of static analysis by abstract interpretation. In details, we have:

- introduced a refined join operator for loop heads which improves precision by preserving unreachability;
- shown that, when using the new join operator with an integer template abstract domain, the descending phase of the analysis terminates even without using a narrowing operator;
- presented several improved (more precise) narrowings for template abstract domains over reals, to be used with the new join operator;
- shown, for the first time, examples of programs over integers and reals where the descending phase of the analysis is either infinite or arbitrarily long.

Both the new join and the improved narrowings may be easily applied to existent analyzers with little effort. In the case of structured program, they only require a single check in the abstract join in order to make it strict.

The new join operator may be used systematically with structured programs, since it improves both precision and speed at the same time. The same cannot be said for the new narrowings over reals or for the idea of not using narrowing at all with integer domains. In this case, we may get better precision, as shown in Example 9, but at the expense of a greater computational cost, since the analysis of the loops might be repeated several times. The good point is that we increase the computational cost only when we improve precision w.r.t. the standard narrowing.

The impact of the repeated computations of loops might be probably reduced by delaying analysis of the inner loops until outer loops are stabilized, so that a long descending sequence in a loop does not force to repeatedly analyze the inner loops. However the impact of the new narrowing on the precision and performance of the analysis on realistic test cases will be the topic of a future work.

Only a few papers in the literature deal with narrowing and the descending phase of the analysis. In [12], the authors try to recover precision by restarting the analysis after that a post-fixpoint has been reached. In [1] and [2], the authors propose to combine widening and narrowing during the analysis, resulting in multiple intertwined ascending and descending phases. Moreover, [1] also proposes to restart (part of) the analysis when the abstract value associated to the exit node of a loop is refined during the descending phase. Our left-strict join operator may be viewed as a variant of the restarting policy in [1], where restart is triggered only when unreachability is detected. However, while in the previous work restarting is a feature of the equation solver, here it is realized directly at the semantic level.

Mostly, our work is orthogonal to the ones cited above: the new operators we have defined may be used within these frameworks to get more precise results.

The idea of avoiding narrowing in the descending phase is used in many papers, with the proviso of bounding the number of descending iterations to ensure termination. In this paper we show that, under certain conditions and ignoring performance issues, we do not need to bound the number of iterations.

References

1. Gianluca Amato and Francesca Scozzari. Localizing widening and narrowing. In Francesco Logozzo and Manuel Fähndrich, editors, *Static Analysis. 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013, Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 25–42, Berlin Heidelberg, 2013. Springer.
2. Kalmer Apinis, Helmut Seidl, and Vesal Vojdani. How to combine widening and narrowing for non-monotonic systems of equations. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 377–386, New York, NY, USA, 2013. ACM.
3. A. Costan, Stephane Gaubert, Eric Goubault, Matthieu Martel, and Sylvie Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 462–475, Berlin Heidelberg, 2005. Springer.
4. Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130, Paris, France, 1976. Dunod.
5. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM Press, New York, NY, USA, January 1977.
6. Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 269–282. ACM Press, New York, NY, USA, January 1979.
7. Patrick Cousot and Radhia Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In Maurice Bruynooghe and Martin Wirsing, editors, *Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP '92 Leuven, Belgium, August 26–28, 1992, Proceedings*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer, Berlin Heidelberg, 1992. Invited paper.
8. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–97, New York, NY, USA, January 1978. ACM Press.
9. Thomas Martin Gawlitza and David Monniaux. Invariant generation through strategy iteration in succinctly represented control flow graphs. *Logical Methods in Computer Science*, 8(3), 2012.
10. Thomas Martin Gawlitza and Helmut Seidl. Solving systems of rational equations through strategy iteration. *ACM Transactions on Programming Languages and Systems*, 33(3):1–48, April 2011.

11. Laure Gonnord and Nicolas Halbwachs. Combining widening and acceleration in linear relation analysis. In Kwangkeun Yi, editor, *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006. Proceedings*, volume 4134 of *Lecture Notes in Computer Science*, pages 144–160, Berlin Heidelberg, 2006. Springer.
12. Nicolas Halbwachs and Julien Henry. When the decreasing sequence fails. In Antoine Miné and David Schmidt, editors, *Static Analysis, 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings*, volume 7460 of *Lecture Notes in Computer Science*, pages 198–213, Berlin Heidelberg, 2012. Springer.
13. Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, March 2006.
14. Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In Radhia Cousot, editor, *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005. Proceedings*, volume 3385 of *Lecture Notes in Computer Science*, pages 25–41. Springer, Berlin Heidelberg, January 2005.
15. Axel Simon, Andy King, and Jacom M. Howe. Two variables per linear inequality as an abstract domain. In Michael Leuschel, editor, *Logic Based Program Synthesis and Transformation 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17–20, 2002. Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 71–89. Springer, Berlin Heidelberg, 2003.