

## Descending chains and narrowing on template abstract domains

Gianluca Amato · Simone Di Nardo Di Maio ·  
Maria Chiara Meo · Francesca Scozzari

the date of receipt and acceptance should be inserted later

**Abstract** A static analysis by abstract interpretation is typically composed of an ascending phase followed by a descending one. The descending phase is used to improve the precision of the analysis after that a post-fixpoint has been reached. Termination is often guaranteed by using narrowing operators, especially on numerical domains which are generally endowed with infinite descending chains. Under the hypothesis of dealing with reducible flow graphs, we provide an abstract semantics which improves the analysis precision and we show that, for a large class of numerical abstract domains over integer variables (such as intervals, octagons, template parallelotopes and template polyhedra), infinite descending chains cannot arise and we can safely omit narrowing. The abstract semantics is a slight variation of the standard one and can be easily implemented. We also provide an acceleration procedure which ensures termination of the descending phase without narrowing even with non-reducible graphs. Finally, we propose a new family of weak narrowing operators for real variables which improve the analysis precision.

**Keywords** static analysis · abstract interpretation · template domains · narrowing

**CR Subject Classification** D.2.4 · F.3.2

**Mathematics Subject Classification (2000)** 68Q60 · 68N30

---

Gianluca Amato  
Università di Chieti–Pescara, Italy  
Tel.: +39-085-4537686  
Fax: +39-085-4537639  
E-mail: gianluca.amato@unich.it

Maria Chiara Meo · Simone Di Nardo Di Maio · Francesca Scozzari  
Università di Chieti–Pescara, Italy

## 1 Introduction

Computing a static analysis in the framework of *abstract interpretation* [11,12] typically amounts to solving a system of equations

$$\begin{cases} x_1 = F_1(x_1, \dots, x_n) \\ \vdots \\ x_n = F_n(x_1, \dots, x_n) \end{cases} \quad (1)$$

describing the program behavior. Each index  $i \in \{1, \dots, n\}$  represents a control point of the program to be analyzed and each  $F_i$  is a monotone, state-transition operator. The unknowns<sup>1</sup>  $x_1, \dots, x_n$  associated to each control point  $i \in \{1, \dots, n\}$  range over an *abstract domain*  $V$ , which encodes the property we want to analyze. An element of  $V$  is called *abstract object* and represents a set of concrete states.

We are interested in finding the (least) solution, over the domain  $V$ , of the set of equations  $F = (F_1, \dots, F_n)$  associated to the program to be analyzed. The abstract interpretation framework ensures that any solution of the set of equations 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  $V^n$ . 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, making the approach impractical.

Therefore, the standard method to perform the analysis is to compute an over approximation of the least solution of the system of equations, using widening and narrowing operators [10, 13]. For specific abstract domains or for restricted classes of programs, we may find in the literature alternatives, such as acceleration operators [17] and strategy/policy iteration [9, 15, 16], but these methods are not generally applicable and their complexity may be impractical.

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

- it is an upper bound on  $V$ ;
- when used in equations of the kind  $x_i = x_i \nabla F_i(x_1, \dots, x_n)$ , it precludes the appearance 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  $x_i = F_i(x_1, \dots, x_n)$  with  $x_i = x_i \nabla F_i(x_1, \dots, x_n)$ . The replacement may involve all unknowns or, more commonly, only the ones corresponding to loop heads in the dependency graph of the equation system. Applying widening in this way ensures the termination of a Kleene iteration, but we only get a post-fixpoint of the function  $F = (F_1, \dots, F_n)$ , 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

<sup>1</sup> We use the terms *variable* to denote a variable in the program, and *unknown* to denote a variable in the data-flow equations.

next example<sup>2</sup> shows this phenomenon using the abstract domain  $\text{Int}_{\mathbb{Z}}$  of integer intervals, defined as:

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

where  $\emptyset$  denotes the empty set of concrete states, i.e., an unreachable control point. The standard widening on intervals [10] is defined as follows:

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

where

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

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

*Example 1* Consider the example program `unreachableLoop` 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  $\text{Int}_{\mathbb{Z}}$  with the standard widening. Therefore, we replace the second and the tenth equation in Fig. 1(c) with

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

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

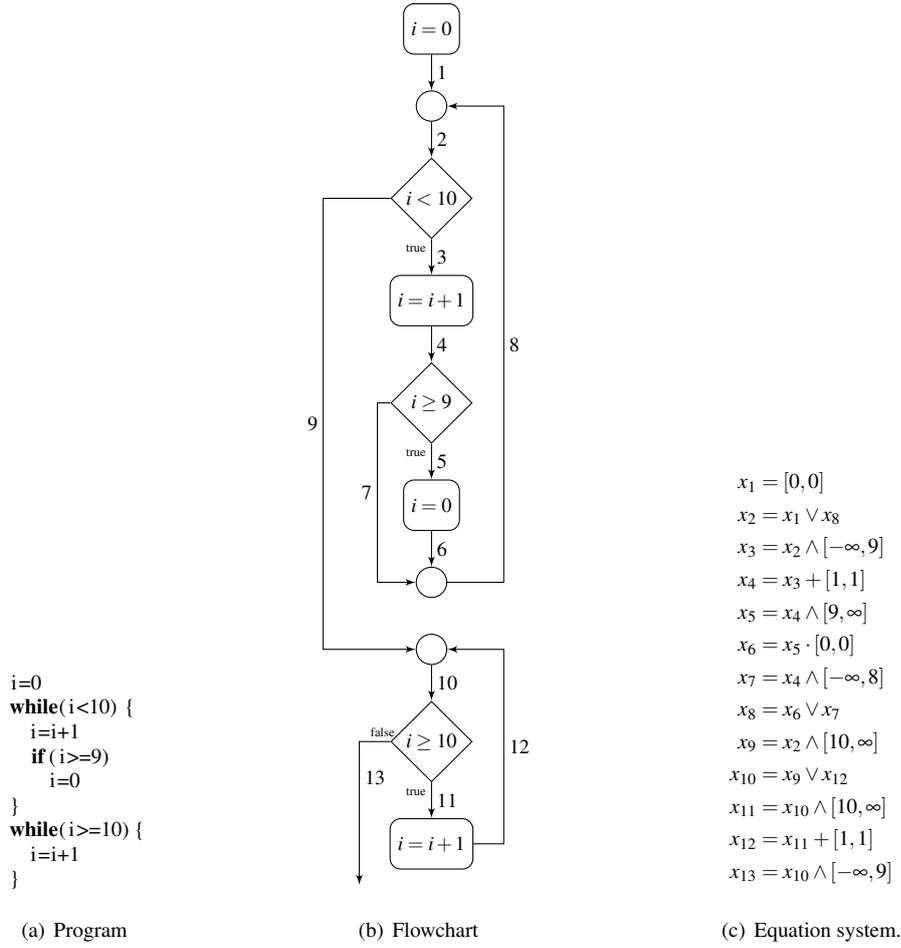
$$x_{10} := x_9 \vee x_{12} = \emptyset \vee [11, +\infty] = [11, +\infty]$$

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  $[11, +\infty]$ ,  $[12, +\infty]$ ,  $[13, +\infty]$ ,  $[14, +\infty]$ , ... which is an infinite descending sequence, whose limit is the empty set.  $\square$

<sup>2</sup> 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.



**Fig. 1** The example program `unreachableLoop`. Note that  $x_5$  appears in the right hand side of equation  $x_6$  since unreachability must be propagated.

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 appearance 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  $\Delta$ , is a binary operator on an abstract domain  $V$  such that:

- $a_1 \Delta a_2$  is only defined when  $a_2 \leq a_1$ ;
- it holds that  $a_2 \leq a_1 \Delta a_2 \leq a_1$ ;

- when used in equations of the kind  $x_i = x_i \Delta F_i(x_1, \dots, x_n)$ , it precludes the appearance of infinite descending chains for  $x_i$ .

The standard narrowing for intervals [10], for example, is defined as:

$$I \Delta \emptyset = \emptyset$$

$$[l_1, u_1] \Delta [l_2, u_2] = [l', u']$$

where

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

Essentially, it works by refining only unbounded extremes. For instance,  $[0, \infty] \Delta [0, 10] = [0, 10]$  but  $[0, 10] \Delta [0, 9] = [0, 10]$ . 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  $x_2 = x_2 \Delta (x_1 \vee x_8)$  and  $x_{10} = x_{10} \Delta (x_9 \vee x_{12})$  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  $x_{10} := [10, +\infty] \Delta [11, +\infty] = [10, +\infty]$ : 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, under certain conditions, narrowing may be omitted even in domains with infinite descending chains. We start by proving it for the domain of integer intervals and for structured programs. We progressively relax these assumptions and show that it holds in the more general case of integer template abstract domains [20] and non-structured programs. More generally, we show that this property holds for any domain which satisfies a newly defined *bottom chain condition*. In order to ensure termination, we need to introduce a new join operator (for structured programs) and a special acceleration procedure (in the general case). This is the first example of acceleration operator for descending chains. On template abstract domains with real bounds we cannot avoid using narrowing. However, inspired by the results on integers, we provide a family of narrowing operators which are more precise than the standard ones.

The question whether narrowing can be omitted naturally arises since its very first definition. Actually, in many practical cases, the descending chain without narrowing terminates in a few steps. We believe that our result answers this question and, in addition, sheds some light on the behaviour of solvers on descending chains.

*Plan of the paper.* In Section 2 we recall some basic notions on equation systems. In Section 3 we deal with the case of integer intervals and structured programs. In Section 4 we generalize the results to any domain which satisfies the bottom chain condition and in Section 5 we prove that this condition is satisfied by all the template domains with integer bounds. Section 6 shows that, even if convergence is ensured, the length of the descending chain may be arbitrarily long. We generalize these results to non-structured programs in Section 7, while in Section 8 we deal with the case of abstract domains with real bounds. Finally, Section 9 concludes with some final remark and related work.

Sections 3, 5, 6 and 8 are based on results appeared in [2] in preliminary form and without proofs. The generalizations to domains satisfying the bottom chain condition and to non-structured programs in Sections 4 and 7 are entirely new. Moreover, the whole presentation has been formalized in a more general setting using flow graphs and proofs are provided for all the results.

## 2 Equation systems, flowcharts and dependencies

In this paper we consider the general problem of solving equation systems whose unknowns take values in a bounded join-semilattice  $(V, \perp, \vee)$ . Given a set  $X$  of unknowns, an *assignment* is a map  $\rho : X \rightarrow V$ . An *equation system* is a map  $F : (X \rightarrow V) \rightarrow (X \rightarrow V)$  from assignments to assignments. We will only consider equation systems with a finite number of unknowns. A post-fixpoint for  $F$  is an assignment  $\rho$  such that  $F(\rho) \leq \rho$ , where  $\leq$  is the standard pointwise extension of the ordering on  $V$  to assignments. In the setting of static analysis, solving an equation system means to find out a post-fixpoint for it. In many cases, an assignment which is bigger than a post-fixpoint also suffices. Determining a fixpoint (or even the least fixpoint) of the equation system is just an additional bonus.

In the previous section, we have derived an equation system from the flowchart of a program, by associating an unknown to each edge. Equation systems which arise in this way have a particular form: for each unknown  $x$ , the corresponding equation is either of the form  $x = x_1 \vee \dots \vee x_n$  or of the form  $x = E$  where  $x_1, \dots, x_n$  are other unknowns and  $E$  is an expression with at most one unknown.

In our theoretical treatment we will work with equation systems which are more general than those which arise from flowcharts, since each right hand side is of the form  $x = E_1 \vee \dots \vee E_n$  and each  $E_i$  is an expression which contains at most one unknown. We will derive these equation systems from graphs we will call semantic flow graphs, which are similar to flowcharts but more abstract since not tied to the syntax of a particular language.

**Definition 1 (Flow Graph)** A *flow graph* is a tuple  $\mathcal{G} = (N, E, \iota)$  where  $N$  is the set of nodes,  $E \subseteq N \times N$  is the set of edges and  $\iota \in N$  is the root node and has no incoming edges.

A *path* in the graph  $\mathcal{G} = (N, E, \iota)$  is a sequence  $\pi = n_0 \dots n_k$  with  $k \geq 0$  such that  $(n_{i-1}, n_i) \in E$  for each  $i \in \{1, \dots, k\}$ . We say that  $\pi$  starts from  $n_0$  and ends in  $n_k$ .

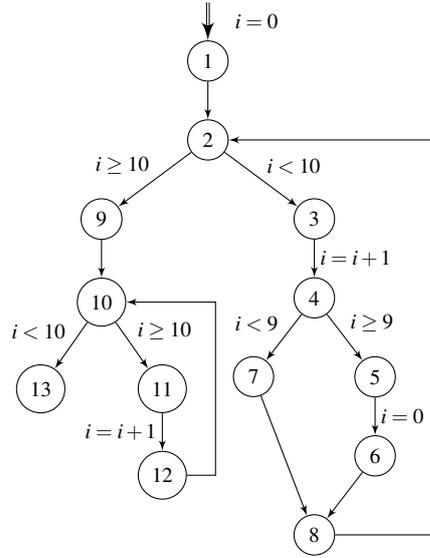
We say that the node  $n$  *dominates* the node  $m$  if each path from  $\iota$  to  $m$  passes through  $n$ . If  $n \neq m$  we say that  $n$  *strictly dominates*  $m$ . A *back-edge* is any edge  $(m, n)$  such that  $n$  dominates  $m$ .

A semantic flow graph (over the poset  $V$ ) is a flow graph with a distinguished value (the initial value for the root node) and whose edges are labeled with monotone functions (the state-transition functions). The labels and the distinguished value contain the semantic information that in a flowchart is conveyed by the shape and label of nodes.

**Definition 2 (Semantic Flow Graph)** A *semantic flow graph* over the poset  $V$  is a tuple  $\mathcal{G} = (N, E, \iota, \{f_e\}_{e \in E}, v_\iota)$  where  $(N, E, \iota)$  is a flow graph, each  $f_e : V \rightarrow V$  is a monotone function and  $v_\iota \in V$ .

**Definition 3 (Equation system derived from a semantic flow graph)** From any semantic flow graph  $\mathcal{G} = (N, E, \iota, \{f_e\}_{e \in E}, v_\iota)$  over the bounded join-semilattice  $V$ , we derive an *equation system*  $F_{\mathcal{G}}$  such that the unknowns of  $F_{\mathcal{G}}$  are the nodes of the graph and

$$F_{\mathcal{G}}(\rho)(n) = \begin{cases} v_\iota & \text{if } n = \iota \\ \bigvee_{(m,n) \in E} f_{(m,n)}(\rho(m)) & \text{otherwise.} \end{cases}$$



**Fig. 2** The semantic flow graph for the equation system in Figure 1(c)

The equation system  $F$  is called *strict* if each  $f_e$  is strict, i.e.  $f_e(\perp) = \perp$ . When we say that  $F$  is derived from the semantic flow graph  $\mathcal{G}$ , we mean that  $F = F_{\mathcal{G}}$ .

Usually, solvers of equations systems keep a current assignment  $\rho$  which is updated by selecting some unknown to be recomputed according to their corresponding equations. This is formalized by the following definition.

**Definition 4 (Update of an assignment)** Fixed a semantic flow graph  $\mathcal{G}$ , for each assignment  $\rho$  and  $X \subseteq N$ , the assignment

$$\rho^X(n) = \begin{cases} F_{\mathcal{G}}(\rho)(n) & \text{if } n \in X, \\ \rho(n) & \text{otherwise,} \end{cases}$$

is called the *update of  $\rho$  for  $X$* .

The following facts are well known and easy to check. However, since we use them extensively in the rest of the paper, we prefer to state them clearly.

**Proposition 1 (Properties of  $F_{\mathcal{G}}$ )** For any semantic flow graph  $\mathcal{G}$ , the equation system  $F_{\mathcal{G}}$  is monotone on assignments. Moreover, the function which maps an assignment  $\rho$  to its update  $\rho^X$  is monotone, too. Finally, if  $\rho$  is a post-fixpoint, then  $\rho^X$  is a post-fixpoint.

*Example 3* The equation system in Figure 1(c) is derived from the semantic flow graph in Figure 2. In this semantic flow graph, the functions labeling the edges are depicted by boolean expression and assignments in order to ease the comparison with the flowchart.  $\square$

Semantic flow graphs and flowcharts are related. Essentially, one may be obtained from the other by changing edges in nodes and vice-versa. In this paper, we prefer semantic

flow graphs since they are used by many libraries for solving fixpoint equations such as `Fixpoint`<sup>3</sup> and `ScalaFix`<sup>4</sup>.

### 3 Descending chains on intervals of integers

In this section we informally analyze how termination of the descending phase may be ensured, in the absence of narrowing, with the domain of integer intervals. In Section 4 we will formalize our reasoning and prove correctness results in a more general setting. Note that what is generally called *interval domain* in the literature is an extension of the domain we are considering here, and it would be better called *box domain* since its elements are boxes in  $\mathbb{R}^n$ . However, the box domain is a particular case of a template abstract domain, and will be dealt with in Section 5.

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

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

or

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

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

**Proposition 2** *Let  $\{I_i\}_{i \in \mathbb{N}}$  be an infinite descending chain of integer intervals. Then  $\bigwedge_{i \in \mathbb{N}} I_i = \emptyset$ .*

*Proof* If  $\{I_i\}_{i \in \mathbb{N}}$  is an infinite descending chain of intervals and  $I_i = [l_i, u_i]$ , then either  $l_i$  or  $-u_i$  is an infinite ascending chain of integers. Therefore, either  $l_i \rightarrow +\infty$  or  $u_i \rightarrow -\infty$ . Without loss of generality, assume we are in the first case. Then, for each  $x \in \mathbb{R}$ , there is  $j \in \mathbb{N}$  such that  $l_j > x$ , hence  $x \notin \bigwedge_{i \in \mathbb{N}} I_i$ . We have  $\bigwedge_{i \in \mathbb{N}} I_i = \emptyset$ .  $\square$

In the rest of this section, we work with equation systems which have been generated by structured programs, i.e., programs whose control flow constructs are while, for and repeat loops, if-then(-else), break and continue. 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 flow graph which comes from outside the loop and *back* the back-edge. Since in a reducible graph 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  $x_{loop} = x_{in} \vee^\perp x_{back}$ , where  $\vee^\perp$  is a left-strict variant of the join operator defined as:

$$I_1 \vee^\perp I_2 = \begin{cases} \emptyset & \text{if } I_1 = \emptyset \\ I_1 \vee I_2 & \text{otherwise} \end{cases} \quad (2)$$

<sup>3</sup> <http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint/>

<sup>4</sup> <https://github.com/jandom-devel/ScalaFix>

The new set of equations is correct (only for structured programs) 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  $x_{loop} = x_{in} \vee^\perp x_{back}$ . 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 2.

The above considerations hold not only for integer intervals, but for any numerical abstract domain with a distinguished value  $\perp$  denoting unreachability.

Moreover, they can be easily lifted to  $n$ -ary loop join nodes. For example, if a loop join node has equation

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

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:

$$x_{loop} = (x_{in_1} \vee \dots \vee x_{in_u}) \vee^\perp (x_{back_1} \vee \dots \vee x_{back_v}) .$$

The next section is devoted to formalize the above reasoning in a formal, more general setting, and to prove the correctness of the approach.

#### 4 The general case

In the general case, we deal with an equation system  $F$  derived from a flow graph  $\mathcal{G}$ . The first step is to define a new equation system  $F^\perp$  which uses a left-strict variant of  $\vee$  and which is still correct.

**Definition 5** If  $F$  is a strict equation system derived from  $\mathcal{G}$ , we define the new equation system  $F^\perp$  as follows:

$$F^\perp(\rho)(n) = \begin{cases} v_l & \text{if } n = l \\ \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho(m)) \vee^\perp \bigvee_{e=(m,n) \in \text{BackE}} f_e(\rho(m)) & \text{otherwise,} \end{cases}$$

where  $\vee^\perp$  is a variant of  $\vee$  which is strict on the left argument and  $\text{BackE}$  is the set of all back-edges of  $\mathcal{G}$ .

The following theorem shows that if we have a correct solution for  $F^\perp$  (namely, an assignment which is greater than a post-fixpoint), that is also a correct solution for  $F$ . The opposite also holds. This means we can use  $F^\perp$  everywhere, instead of  $F$ .

**Theorem 1** *If  $F$  is a strict equation system derived from  $\mathcal{G}$  and  $\rho$  is a post-fixpoint of  $F^\perp$ , there is a  $\rho' \leq \rho$  which is a post-fixpoint of  $F$ . On the contrary, if  $\rho$  is a post-fixpoint of  $F$ , it is also a post-fixpoint of  $F^\perp$ .*

*Proof* Assume  $\rho$  is a post-fixpoint of  $F^\perp$ . Consider the set  $X$  of nodes which have no contributions from incoming non-backedge nodes, i.e.,

$$X = \left\{ n \in N \setminus \{t\} \mid \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho(m)) = \perp \right\}.$$

Note that if  $n \notin X$  then  $F^\perp(\rho)(n) = F(\rho)(n)$ . Let us define the new assignment  $\rho^\perp$  as follows:

$$\rho^\perp(n) = \begin{cases} \perp & \text{if there is } m \in X \text{ such that } m \text{ dominates } n; \\ \rho(n) & \text{otherwise} \end{cases}$$

We need to prove that  $\rho^\perp$  is a post-fixpoint of  $F$ , i.e.  $\rho^\perp(n) \geq F(\rho^\perp)(n)$  for all nodes  $n$ . There are several cases:

- if  $n$  is not dominated by any node in  $X$ , then  $\rho^\perp(n) = \rho(n) \geq F^\perp(\rho)(n) = F(\rho)(n) \geq F(\rho^\perp)(n)$ .
- if  $n \in X$ , then  $\rho^\perp(n) = \perp$ . We have

$$F(\rho^\perp)(n) = \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho^\perp(m)) \vee \bigvee_{e=(m,n) \in \text{BackE}} f_e(\rho^\perp(m)).$$

Since  $n \in X$ , then  $\bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho^\perp(m)) \leq \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho(m)) = \perp$ . If  $(m,n) \in \text{BackE}$ , then  $n$  dominates  $m$ , hence  $\rho^\perp(m) = \perp$ . As a consequence,  $F(\rho^\perp)(n) = \perp$ .

- if  $n \notin X$  is dominated by a node  $m \in X$  then  $\rho^\perp(n) = \perp$ . For any edge  $(l,n)$ , since  $m$  strictly dominates  $n$ , then  $m$  dominates  $l$  and therefore  $F(\rho^\perp)(n) = \perp$ .

For the second part of the theorem, let  $\rho$  be a post-fixpoint of  $F$ . Then  $\rho$  is a post-fixpoint of  $F^\perp$ , since  $F^\perp(\rho) \leq F(\rho)$ .  $\square$

The next definition generalizes and formalizes the property already seen on integer intervals, that all the infinitely descending chains converge to the bottom of the abstract domain.

**Definition 6 (Bottom chain condition)** We say that a poset  $V$  satisfies the *bottom chain condition* ( $\perp\text{CC}$ ) when, for each infinitely descending chain  $v_1 > v_2 > \dots > v_i > \dots$ , the greatest lower bound  $\bigwedge_i v_i$  exists and is equal to the least element  $\perp$  of  $V$ .

From Proposition 2 it turns out that the integer interval domain satisfies the  $\perp\text{CC}$ . We will show in Section 5 that any template domain satisfies the  $\perp\text{CC}$ .

**Definition 7 (Chaotic iteration sequence)** Given an assignment  $\rho$  for the equation system  $F$  derived from  $\mathcal{G}$ , a chaotic iteration sequence is a sequence of assignments  $\rho = \rho_0, \rho_1, \dots, \rho_i, \dots$  such that:

1. each  $\rho_{i+1}$  is obtained from  $\rho_i$  by selecting a set  $X_i \subseteq N$  and defining  $\rho_{i+1} = \rho_i^{X_i}$ ;
2. **fairness:** for each  $i \in \mathbb{N}$  and  $n \in N$ , if  $F(\rho_i)(n) \neq \rho_i(n)$ , then there exists  $j \geq i$  such that  $n \in X_j$ .

According to our definition, all the chaotic iteration sequences are infinite. Nonetheless, any solver for equation systems immediately stops when it recognizes that the sequence stabilizes, i.e., has reached a fixpoint. In this case, abusing terminology, we call it a finite sequence.

The next theorem shows what is one of the main results of this paper, namely that every chaotic iteration sequence on a domain which satisfies  $\perp\text{CC}$  for an equation system generated by a *reducible graph* eventually stabilizes. For the definition of reducible graph, depth-first ordering, forward, retreating and cross edge we refer to [1].

**Theorem 2** *Assume that  $V$  satisfies the  $\perp\text{CC}$ ,  $F$  is an equation system derived from  $\mathcal{G}$  and  $\rho$  is a post-fixpoint of  $F^\perp$ . If  $\mathcal{G}$  is reducible, every chaotic iteration sequence for  $F^\perp$  starting from  $\rho$  eventually stabilizes.*

*Proof* Let  $\rho = \rho_0, \rho_1, \dots, \rho_i, \dots$  be a chaotic iteration sequence. Assume the sequence does not stabilize, i.e., there is a node  $n$  such that the chain  $\{\rho_i(n) \mid i \in \mathbb{N}\}$  is infinitely descending. Consider a depth-first ordering of  $\mathcal{G}$  and assume without loss of generality that  $n$  is the first node in the ordering which does not stabilize. We recall that, for each  $i$ ,

$$F^\perp(\rho_i)(n) = \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho_i(m)) \vee^\perp \bigvee_{e=(m,n) \in \text{BackE}} f_e(\rho_i(m))$$

If  $(m, n) \in E \setminus \text{BackE}$ , it is not a back-edge. Since  $\mathcal{G}$  is reducible, it is not a retreating edge either, therefore  $m$  comes before  $n$  in the depth-first ordering. This means that, for a big enough  $i$ , the first half of the formula for  $F^\perp(\rho_i)(n)$  stabilizes to a value  $v$ . Assume the stable value is reached at iteration  $j$ . Since  $\{\rho_i(n)\}_i$  does not stabilize and  $V$  satisfies  $\perp\text{CC}$ , it should be that  $\bigwedge_{i \in \mathbb{N}} \rho_i(n) = \perp \geq v$ , hence  $v = \perp$ . This immediately implies that, when the node  $n$  is selected at an iteration  $l \geq j$ , we have  $\rho_l(n) = \perp$ , and therefore  $\{\rho_i(n)\}_i$  stabilizes against the original hypothesis.  $\square$

The special case of static analysis of structured programs on the abstract domain of integer intervals immediately follows from the above theorem.

**Corollary 1** *Assume we have a strict equation system  $F$  generated by a structured program whose loop head nodes are of the form  $x_{loop} = x_{in} \vee x_{back}$ .*

- *If the abstract domain has a distinguished value  $\perp$  denoting unreachability, replacing  $\vee$  with  $\vee^\perp$  in all the loop heads, the new set of equations is still correct.*
- *When using the abstract domain of integer intervals, every chaotic iteration sequence in the new set of equations starting from a post-fixpoint leads to a finite descending sequence.*

*Proof* The first point immediately derives from Theorem 1. Since the program is structured, it follows that the flow graph is reducible. By Proposition 2, the abstract domain of integer intervals satisfies Definition 6, thus we can apply Theorem 2, from which the thesis.

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

## 5 Template abstract domains

Template abstract domains are numerical domains where each abstract object is described by a finite set of linear constraints on the program variables, and the homogeneous coefficients of these constraints are fixed in advance, before starting the analysis. Most important template abstract domains are the domain of intervals (also called box domain) [10], octagons [19], template parallelotopes [3] and template polyhedra [20]. Non-template abstract domains are, among others, polyhedra [14], parallelotopes [4] and two-variables for linear inequality [21].

A template abstract domain (with integer bounds)  $\text{Template}_{\mathbb{Z}}^A$  is defined by an  $m \times n$  real matrix  $A$ , called the *constraint matrix*. An element of  $\text{Template}_{\mathbb{Z}}^A$  is a subset of  $\mathbb{R}^n$  of the form  $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{l} \leq A\mathbf{x} \leq \mathbf{u}\}$ , where  $\mathbf{l} \in \bar{\mathbb{R}}^n$  and  $\mathbf{u} \in \bar{\mathbb{R}}^n$  are, respectively, the lower and upper bounds. In the following, one such object will be denoted by  $[\mathbf{l}, \mathbf{u}]$ . Ordering is given by subset,  $\emptyset$  is the bottom element and the least upper bound  $[\mathbf{l}, \mathbf{u}] \vee [\mathbf{l}', \mathbf{u}'] = [\min(\mathbf{l}, \mathbf{l}'), \max(\mathbf{u}, \mathbf{u}')]$  where  $\min$  and  $\max$  are defined component-wise. Note that, differently from  $\text{Int}_{\mathbb{Z}}$ , an element of  $\text{Template}_{\mathbb{Z}}^A$  is a subset of  $\mathbb{R}^n$  and not of  $\mathbb{Z}^n$ : only the bounds are required to be integer. Also, the constraint matrix  $A$  may contain any real number.

A box is an abstract object where  $A$  is the identity matrix. Octagons are those objects where the coefficient matrix  $A$  allows constraints of the form  $\pm x \pm y \leq c$ . Parallelotopes, instead, are those objects whose matrix  $A$  is invertible.

Under the hypothesis of Theorem 1, it is possible to extend Corollary 1 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 with integer bounds enjoy a property similar to Proposition 2.

**Proposition 3** *Let  $\{I_i\}_{i \in \mathbb{N}}$  be an infinite descending chain of objects  $I_i \in \text{Template}_{\mathbb{Z}}^A$ . Then  $\bigwedge_{i \in \mathbb{N}} I_i = \emptyset$ .*

*Proof* If  $\{[\mathbf{l}^i, \mathbf{u}^i]\}_{i \in \mathbb{N}}$  is an infinite descending chain in  $\text{Template}_{\mathbb{Z}}^A$ , then there exists  $j$  such that either  $\{l_j^i\}_i$  or  $\{-u_j^i\}_i$  is an infinite ascending chain of integers. Therefore, either  $l_j^i \rightarrow +\infty$  or  $u_j^i \rightarrow -\infty$ . Without loss of generality, assume we are in the first case. Then, for each  $x \in \mathbb{R}$ , there is  $k \in \mathbb{N}$  such that  $l_k^i > x$ , hence  $x \notin \bigwedge_{i \in \mathbb{N}} [l_j^i, u_j^i]$ . Therefore  $\bigwedge_{i \in \mathbb{N}} [\mathbf{l}^i, \mathbf{u}^i] = \emptyset$ .  $\square$

Exploiting the above proposition and Theorem 1, we can prove a result analogous to Corollary 1 which, in presence of a left-strict join, allows us to avoid narrowing, still guaranteeing termination.

**Corollary 2** *Assume we have a strict equation system  $F$  generated by a structured program whose loop head nodes are of the form  $x_{loop} = x_{in} \vee x_{back}$ . When using  $\text{Template}_{\mathbb{Z}}^A$ , if we replace  $\vee$  with  $\vee^\perp$  in all the loop heads, every chaotic iteration strategy in the new set of equations starting from a post-fixpoint leads to a finite descending sequence.*

*Proof* The proof is analogous to that one of Corollary 1.

## 6 Finite, arbitrarily long descending chains

The use of  $F^\perp$  allows us to get rid of narrowing. As a consequence, however, we may find programs whose descending chain is finite but arbitrarily long. We now show this phenomenon.

Consider the example program `unreachableLoop2` in Fig. 3(a), and the corresponding flow graph and set of equations in Fig. 3(b) and 3(c). We first perform the analysis using the integer interval domain  $\text{Int}_{\mathbb{Z}}$  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:  $x_2 = x_2 \nabla (x_1 \vee^{\perp} x_4)$  and  $x_6 = x_6 \nabla (x_5 \vee^{\perp} x_8)$ . 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  $x_2 = x_2 \Delta (x_1 \vee^{\perp} x_4)$  and  $x_6 = x_6 \Delta (x_5 \vee^{\perp} x_8)$ . When we first apply narrowing in the second equation, we get:

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

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

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

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

$$\begin{array}{l} x_2 = x_1 \vee^{\perp} x_4 \\ x_6 = x_5 \vee^{\perp} x_8 \end{array} .$$

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, when 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:

	$2^{ns}$ 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.

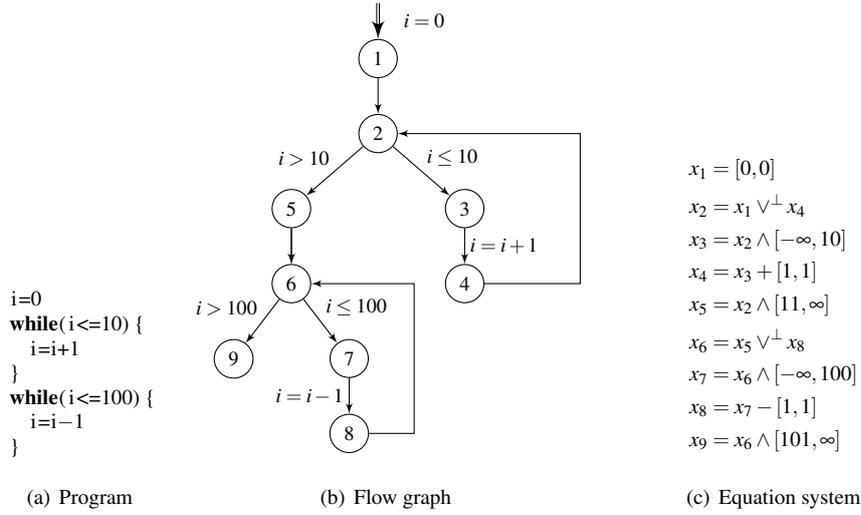


Fig. 3 The example program unreachableLoop2.

## 7 Non-reducible graphs

We first show that Theorem 2 does not hold when the flow graph is not reducible. Consider the program in Figure 4(a), where “?” denotes a non-deterministic boolean expression. The first while loop is the same which appears in the program unreachableLoop. The second while loop is unreachable but has two entry points. The subgraph composed of the nodes from 9 to 14 is not reducible, hence in the right hand side for  $x_{12}$  and  $x_{14}$  we cannot replace  $\vee$  with  $\vee^\perp$ . After the end of the ascending chain with the standard widening, we have  $x_{10} = x_{11} = x_{12} = x_{14} = [10, +\infty]$  and  $x_{13} = [11, +\infty]$ . During the descending chain, if we do not use narrowing,  $x_9, x_{10}$  and  $x_{11}$  become  $\emptyset$ . However, for  $x_{12}, x_{13}$  and  $x_{14}$  an infinite descending chain begins, whose limit is the empty set.

Note that it would be possible to break the descending chain by recognizing that the node 9 dominates all the lower subgraph. Since  $x_9 = \emptyset$ , all the unknowns for the subgraph dominated by 9 may be set to  $\emptyset$ . We may turn this idea into a new procedure for the descending chain which ensures termination also for non-reducible graphs. However, something slightly more complex is required for the general case.

First of all, we need to extend the concept of dominance between nodes to the case of dominance between edges and nodes.

**Definition 8 (Edge dominance)** Given a flow graph  $\mathcal{G} = (N, E, t)$ , we say that a set of edges  $X \subseteq E$  dominates the node  $n$  if each path from  $t$  to  $n$  passes through at least one of the edges in  $X$ .

*Example 4* In the flow graph of Figure 4(b), edges (9, 10) and (9, 11) dominate nodes 12, 13 and 14.  $\square$

The following lemma formalizes the fact that we can set some unknown to  $\perp$  in an assignment, preserving the property of being a post-fixpoint, and thus improving the unreachability information.

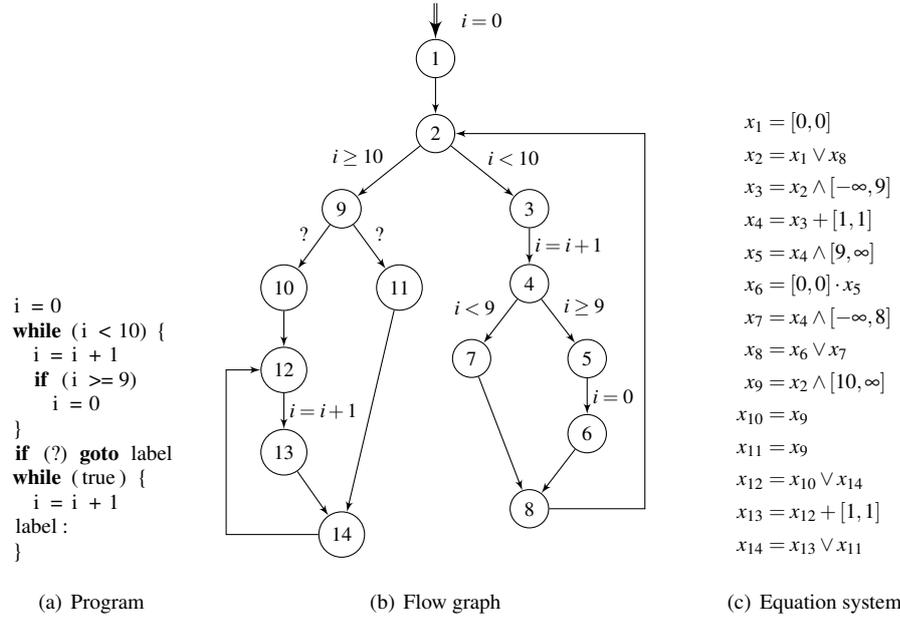


Fig. 4 The example program nonReducible.

**Lemma 1** *If  $F$  is a strict equation system derived from  $\mathcal{G}$  and  $\rho$  is a post-fixpoint of  $F$ , let  $X \subseteq E$  be a set of edges such that, for each  $(m, n) \in X$ ,  $f_{(m, n)}(\rho(m)) = \perp$ . Then, consider the assignment*

$$\rho^\perp(n) = \begin{cases} \perp & \text{if } X \text{ dominates } n \\ \rho(n) & \text{otherwise} \end{cases}$$

We have that  $\rho^\perp$  is a post-fixpoint of  $F$ .

*Proof* We prove that  $\rho^\perp(n) \geq F(\rho^\perp)(n)$  for each  $n \in N$ . We distinguish several cases:

- $X$  does not dominate  $n$ . By monotonicity of  $F$ , it is immediate that  $\rho^\perp(n) = \rho(n) \geq F(\rho)(n) \geq F(\rho^\perp)(n)$ .
- $X$  dominates  $n$ . By definition  $\rho^\perp(n) = \perp$  and  $F(\rho^\perp)(n) = \bigvee_{e=(m, n) \in E} f_e(\rho^\perp(m))$ . For each  $e = (m, n) \in E$ , either  $e \in X$ , hence  $f_e(\rho^\perp(m)) \leq f_e(\rho(m)) = \perp$ , by hypothesis, or  $X$  dominates  $m$ , hence  $\rho^\perp(m) = \perp$ . In both cases  $f_e(\rho^\perp(m)) = \perp$ , and therefore  $F(\rho^\perp)(n) = \perp$ .  $\square$

We can augment a chaotic iteration sequences with new steps which apply Lemma 1 to the current assignment.

**Definition 9 (Chaotic iteration sequence with bottom acceleration)** Given an assignment  $\rho$  for the equation system  $F$  derived from  $\mathcal{G}$ , a *chaotic iteration sequence with bottom acceleration* is a sequence of assignments  $\rho = \rho_0, \rho_0^\perp, \rho_1, \rho_1^\perp, \dots, \rho_i, \rho_i^\perp, \dots$  such that

1.  $\rho_{i+1} = \rho_i^X$  for some  $X \subseteq N$ ;
2.  $\rho_i^\perp$  is obtained from  $\rho_i$  by Lemma 1 by choosing  $X = \{(m, n) \in E \mid f_{(m, n)}(\rho(m)) = \perp\}$ .

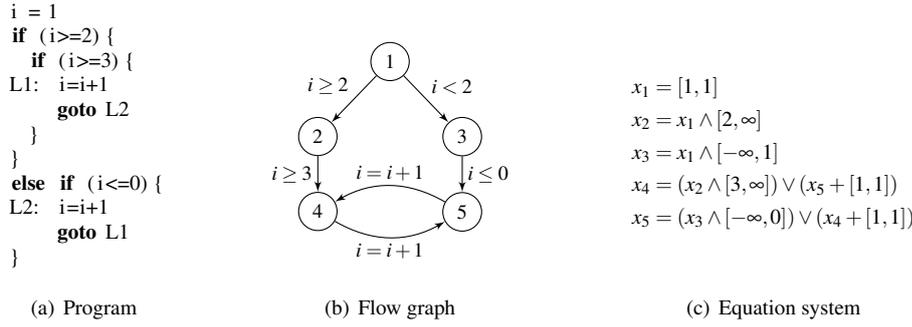


Fig. 5 The example program nonReducible2.

**Theorem 3** Assume that  $V$  satisfies the  $\perp$ CC,  $F$  is a strict equation system derived from  $\mathcal{G}$  and  $\rho$  is a post-fixpoint of  $F$ , any chaotic iteration sequence with bottom acceleration starting from  $\rho$  is ultimately stationary.

*Proof* First of all, note that each iteration sequence with bottom acceleration starting from  $\rho$  is a descending chain, since at each step we preserve the property of being in a post-fixpoint of  $F$ . Assume the sequence  $\rho = \rho_0, \rho_0^\perp, \rho_1, \rho_1^\perp, \dots, \rho_i, \rho_i^\perp, \dots$  is not ultimately stationary, i.e., there is a node  $n$  such that the chain  $\{\rho_i(n) \mid i \in \mathbb{N}\}$  is infinitely descending. Consider a depth-first ordering of  $\mathcal{G}$  and assume without loss of generality that  $n$  is the first node in the ordering which does not stabilize.

Since  $V$  satisfies the  $\perp$ CC we have  $\rho_i(n) \rightarrow \perp$  and therefore  $\rho_i(m) \rightarrow \perp$  for each edge  $(m, n) \in E$ . If  $m$  comes before  $n$  in the depth-first ordering, then the sequence  $\{\rho_i(m)\}_{i \in \mathbb{N}}$  should be ultimately stationary. This means that, if  $X$  is the set of all forward or cross edges pointing to  $n$ , there exists an index  $j$  such that  $f_{(m,n)}(\rho_j(m)) = \perp$  for each  $(m, n) \in X$ . Note that  $X$  dominates  $n$ , therefore  $\rho_j^\perp(n) = \perp$ , against the original hypothesis.  $\square$

*Example 5* Consider the program in Figure 4(a). During the descending chain, as soon as  $x_2$  becomes  $[0, 9]$  the acceleration step sets to  $\perp$  all unknowns from  $x_9$  to  $x_{14}$ .  $\square$

Although in this example we could just track node dominance and realize that unknowns from  $x_{10}$  to  $x_{14}$  may be set to  $\perp$  once  $x_9$  reaches  $\perp$ , this is not true in general, as the following example shows.

*Example 6* Consider the equation system in Figure 5(c) over the domain of integer intervals. Consider the assignment  $\rho = \{x_1 \mapsto [1, 1], x_2 \mapsto \emptyset, x_3 \mapsto [1, 1], x_4 \mapsto [0, +\infty], x_5 \mapsto [0, +\infty]\}$ , which is a post-fixpoint. If we start a descending chain from  $\rho$ ,  $x_4$  and  $x_5$  assume all the values of the form  $[i, +\infty]$  for  $i \geq 0$ . Note that neither 2 nor 3 dominate either 4 or 5. However the set of edges  $\{(2, 4), (3, 5)\}$  dominates both 4 and 5. Therefore, a bottom accelerated descending chain immediately converges.  $\square$

The use of bottom acceleration is more effective than using the new equation system  $F^\perp$ , since it works for both reducible and non-reducible graphs. However, it is more difficult to implement, since it requires changes to the solver algorithm, while  $F^\perp$  may be generally implemented by just changing the equation system that we feed to the solver. Moreover, checking for dominance might slow down the computation in the vast majority of cases when there is no unreachable code, while using  $F^\perp$  has no adverse performance implications.

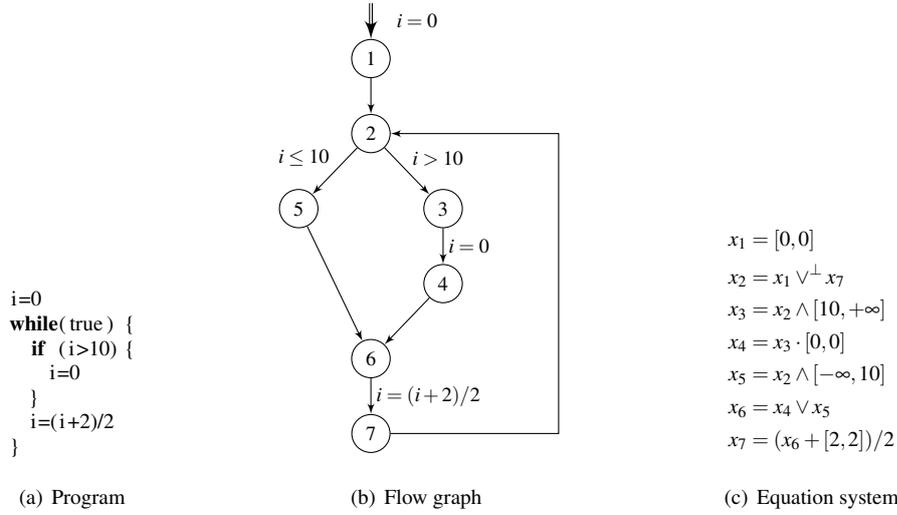


Fig. 6 The example program realLoop.

## 8 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 terminates. 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  $\text{Int}_{\mathbb{R}}$  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

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

*Example 7* Consider the example program realLoop in Fig. 6(a), and the corresponding flow graph and equations in Fig. 6(b) and 6(c). The ascending phase using left-strict join and standard widening, i.e.,  $x_2 = x_2 \vee (x_1 \vee^\perp x_7)$ , reaches a post-fixpoint in two iterations.

	1 <sup>st</sup> ascending iteration	2 <sup>nd</sup> ascending iteration
$x_1$	$[0, 0]$	$[0, 0]$
$x_2$	$[0, 0]$	$[0, 0] \vee [0, 1] = [0, +\infty]$
$x_3$	$\emptyset$	$[10, +\infty]$
$x_4$	$\emptyset$	$[0, 0]$
$x_5$	$[0, 0]$	$[0, 10]$
$x_6$	$[0, 0]$	$[0, 10]$
$x_7$	$[1, 1]$	$[1, 6]$

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

	1 <sup>st</sup> descending iteration	2 <sup>nd</sup> descending iteration
$x_1$	[0, 0]	[0, 0]
$x_2$	$[0, 0] \vee^\perp [1, 6] = [0, 6]$	$[0, 0] \vee^\perp [1, 4] = [0, 4]$
$x_3$	$\emptyset$	$\emptyset$
$x_4$	$\emptyset$	$\emptyset$
$x_5$	[0, 6]	[0, 4]
$x_6$	[0, 6]	[0, 4]
$x_7$	[1, 4]	[1, 3]

At the next iterations, we obtain:

$$\begin{array}{ll}
 x_2 = [0, 3] & x_7 = \left[1, \frac{5}{2}\right] \\
 x_2 = \left[0, \frac{5}{2}\right] & x_7 = \left[1, \frac{9}{4}\right]
 \end{array}$$

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

Taking advantage of the result in Proposition 2, we can define a new notion of *weak narrowing*. A weak narrowing is very similar to the standard narrowing, the only difference being that a descending chain built with weak narrowing might either stabilize or converge to  $\perp$ .

**Definition 10 (Weak narrowing)** A *weak narrowing* over the pointed poset  $(V, \perp)$  is a binary operator  $\Delta : V \times V \rightarrow V$  such that:

- $v_1 \Delta v_2$  is only defined when  $v_2 \leq v_1$ ;
- $v_2 \leq v_1 \Delta v_2 \leq v_1$ ;
- given any decreasing chain  $\{v_i\}_{i \in \mathbb{N}}$ , the sequence  $\{v'_i\}_{i \in \mathbb{N}}$  defined as  $v'_0 = v_0$  and  $v'_{i+1} = v'_i \Delta v_{i+1}$  is either ultimately stationary or converges towards  $\perp$ .

The following holds for weak narrowing.

**Theorem 4** Let  $F$  be an equation system derived from a reducible semantic flow graph  $\mathcal{G}$  and  $\rho$  a post-fixpoint of  $F^\perp$ . Let  $F_\Delta^\perp$  the equation system derived from  $F^\perp$  by applying the weak narrowing  $\Delta$  to each loop head. Every chaotic iteration sequence for  $F_\Delta^\perp$  starting from  $\rho$  eventually stabilizes.

*Proof* It is similar to the proof of Theorem 2. However, when node  $n$  is chosen, which is the least non-stabilizing node in a depth-first ordering of  $\mathcal{G}$ , two cases arise:

- if there are no back-edges pointing to  $n$ , then

$$F_\Delta^\perp(\rho_i)(n) = \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho_i(m))$$

and  $\{\rho_i(n)\}_i$  stabilizes since all the nodes which come before  $n$  in the depth-first ordering do stabilize;

– if there are back-edges pointing to  $n$ , i.e.,  $n$  is a loop head, then

$$F_{\Delta}^{\perp}(\rho_i)(n) = \rho_i(n) \Delta \left( \bigvee_{e=(m,n) \in E \setminus \text{BackE}} f_e(\rho_i(m)) \vee^{\perp} \bigvee_{e=(m,n) \in \text{BackE}} f_e(\rho_i(m)) \right)$$

Weak narrowing ensures that if the chain  $\{\rho_i(n)\}_i$  is infinitely descending, then  $\bigwedge_i \rho_i(n) = \perp$ . With the same reasoning in Theorem 2, we know that this case never occurs.  $\square$

Exploiting Proposition 2, we can define a weak narrowing on intervals of reals which refines successive descending iterations at the nearest integer, since we cannot have an infinite descending chain whose bounds are all integers.

**Definition 11 (Weak narrowing on reals)** We define a weak narrowing operator  $\Delta^1$  on  $\text{Int}_{\mathbb{R}}$  as follows:

$$\begin{aligned} I \Delta^1 \emptyset &= \emptyset \\ [l_1, u_1] \Delta^1 [l_2, u_2] &= [l', u'] \end{aligned}$$

where

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

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

Note that this operator may be immediately extended componentwise to template domains. The same also holds for the other weak narrowings we will introduce later in this section. We define them and prove they are weak narrowings only for intervals, for the sake of conciseness.

The new weak narrowing  $\Delta^1$  refines infinite bounds to finite values, as the standard narrowing, 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, any descending sequence terminates.

**Theorem 5** *The operator  $\Delta^1$  is a weak narrowing.*

*Proof* We need to show that for any  $[l_2, u_2] \leq [l_1, u_1]$ , it holds that  $[l_2, u_2] \leq [l_1, u_1] \Delta^1 [l_2, u_2] \leq [l_1, u_1]$  and any infinite descending chain of narrowing eventually stabilizes or converges to  $\emptyset$ .

Let  $[l', u'] = [l_1, u_1] \Delta^1 [l_2, u_2]$ . By Definition 11, we need to prove that  $u_2 \leq u' \leq u_1$  and  $l_1 \leq l' \leq l_2$ . If  $u_1 = +\infty$  then  $u' = u_2$  and  $u_2 \leq u' \leq u_1$ . Otherwise, since  $u_2 \leq u_1$  by definition of narrowing operator, it follows that  $u_2 = \min(u_1, u_2) \leq \min(u_1, \lceil u_2 \rceil) = u' \leq u_1$ . The proof for the lower bound is symmetric.

Given any chain  $\{[l_i, u_i]_{i \in \mathbb{N}}\}$ , where  $[l_{i+1}, u_{i+1}] \leq [l_i, u_i]$  for any  $i \in \mathbb{N}$ , we need to prove that the sequence:

$$\begin{aligned} [l'_0, u'_0] &= [l_0, u_0] \\ [l'_{i+1}, u'_{i+1}] &= [l'_i, u'_i] \Delta^1 [l_{i+1}, u_{i+1}] \end{aligned}$$

eventually stabilizes or converges towards  $\emptyset$ .

First assume that  $u_0 \neq +\infty$ . Thus for any  $i \in \mathbb{N}$  we have that  $u_i \neq +\infty$ . From Definition 11, it follows that, for any  $i \in \mathbb{N}$ ,  $u'_{i+1} = \min(u'_i, \lceil u_{i+1} \rceil)$ , from which it immediately follows that either  $u'_{i+1} = u_0$  or  $u'_{i+1} \in \mathbb{N}$ . Thus, either the sequence is constant (and is equal to  $u_0$ ) or there exists  $k \in \mathbb{N}$  such that  $u'_k \in \mathbb{N}$ . In the first case, there is nothing to prove, since the sequence stabilizes to  $u_0$ . In the second case, the narrowing sequence starting from  $u'_k$  is a sequence of integers, thus it converges towards  $-\infty$  and  $\bigwedge_i [l'_i, u'_i] = \emptyset$ .

Now we assume that  $u_0 = +\infty$ . In this case, either the narrowing sequence is stable to  $+\infty$ , or there exists  $k \in \mathbb{N}$  such that  $u_k \neq +\infty$ : we fall in the same case as  $u_0 \neq +\infty$ , just shifted by  $k$ .

The proof for lower bounds is symmetric. The extension to template domains is immediate.  $\square$

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

*Example 8* We compute the descending chain of Example 7 using the standard narrowing on intervals. We start from the post-fixpoint and use the equation  $x_2 = x_2 \Delta (x_1 \vee^\perp x_7)$ . At the first descending iteration we get

$$x_2 = [0, +\infty] \Delta ([0, 0] \vee^\perp [1, 6]) = [0, +\infty] \Delta [0, 6] = [0, 6] .$$

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

$$x_2 = x_2 \Delta (x_1 \vee^\perp x_7) = [0, 6] \Delta [0, 4] = [0, 6] .$$

We now recompute the descending chain of Example 7 using the weak narrowing on reals  $\Delta^1$  in Def. 11. 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

$$x_2 = x_2 \Delta^1 (x_1 \vee^\perp x_7) = [0, 6] \Delta^1 [0, 4] = [0, 4]$$

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

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

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

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

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

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

**Definition 12 ( $\delta$ -narrowing)** Let  $\delta \in \mathbb{R}$  such that  $\delta > 0$ . We define a new weak narrowing on intervals of reals:

$$I \Delta^\delta \emptyset = \emptyset$$

$$[l_1, u_1] \Delta^\delta [l_2, u_2] = [l', u']$$

where

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

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

The above weak narrowing produces a descending chain whose elements differ for a multiple of  $\delta$ , which is fixed in advance. It generalizes  $\Delta^1$  given in Definition 11. In fact, Def. 12 boils down to Def. 11 when  $\delta = 1$ .

**Proposition 4** Let  $\{[l_i, u_i]\}_{i \in \mathbb{N}}$  be an infinite descending chain of real intervals. Assume that there exists  $\delta > 0 \in \mathbb{R}$  and  $k \in \mathbb{N}$  such that, for any  $i \geq k$ , it holds that:

- either  $u_i = u_{i+1}$  or  $u_i - u_{i+1} \geq \delta$ ;
- either  $l_i = l_{i+1}$  or  $l_{i+1} - l_i \geq \delta$ .

Then  $\bigwedge_{i \in \mathbb{N}} [l_i, u_i] = \emptyset$ .

*Proof* We show the result when  $k = 0$ . The other case immediately follows by considering the sequence  $\{[l_i, u_i]\}_{i \geq k \in \mathbb{N}}$ . If  $\{[l_i, u_i]\}_{i \in \mathbb{N}}$  is an infinite descending chain of real intervals, then either  $\{l_i\}_{i \in \mathbb{N}}$  or  $\{-u_i\}_{i \in \mathbb{N}}$  is an infinite ascending chain of reals. Without loss of generality, assume we are in the first case. By hypothesis, it holds that either  $l_i = l_{i+1}$  or  $l_{i+1} - l_i \geq \delta$  for a fixed  $\delta > 0$ . Since the chain is infinitely descending,  $l_{i+1} - l_i \geq \delta$  infinitely many times, hence  $l_i \rightarrow +\infty$ . Then, for each  $x \in \mathbb{R}$ , there is  $j \in \mathbb{N}$  such that  $l_j > x$ , hence  $x \notin \bigwedge_{i \in \mathbb{N}} [l_i, u_i]$ . We have  $\bigwedge_{i \in \mathbb{N}} [l_i, u_i] = \emptyset$ .  $\square$

**Theorem 6** The operator  $\Delta^\delta$  is a weak narrowing.

*Proof* We need to show that for any  $[l_2, u_2] \leq [l_1, u_1]$ , it holds that  $[l_2, u_2] \leq [l_1, u_1] \Delta^\delta [l_2, u_2] \leq [l_1, u_1]$  and any infinite descending chain of narrowing either stabilizes or converges to  $\emptyset$ .

Let  $[l', u'] = [l_1, u_1] \Delta^\delta [l_2, u_2]$ . By Definition 13, it is immediate to see that either  $l' = l_1$ , or  $l' = l_2$ , or  $l' = \delta \lfloor l_2 / \delta \rfloor$ . Note that, for any  $\delta \in \mathbb{R}$ , we have that  $l_2 \geq \delta \lfloor l_2 / \delta \rfloor$ . Since  $l_1 \leq l_2$ , it follows that  $l_1 \leq \max(l_1, \delta \lfloor l_2 / \delta \rfloor) \leq l_2$ . In all cases,  $l_1 \leq l' \leq l_2$ . The symmetric reasoning holds for upper bounds.

Given any chain  $\{[l_i, u_i]\}_{i \in \mathbb{N}}$ , where  $[l_{i+1}, u_{i+1}] \leq [l_i, u_i]$  for any  $i \in \mathbb{N}$ , we need to prove that the sequence:

$$[l'_0, u'_0] = [l_0, u_0]$$

$$[l'_{i+1}, u'_{i+1}] = [l'_i, u'_i] \Delta^\delta [l_{i+1}, u_{i+1}]$$

either stabilizes or converges to  $\emptyset$ .

First assume that  $u_0$  is a multiple of  $\delta$ . From Definition 12, it follows that, for any  $i \in \mathbb{N}$ ,  $u'_{i+1} = \min(u'_i, \delta \lceil u_{i+1} / \delta \rceil)$  is a multiple of  $\delta$ . Therefore, either  $u'_{i+1} = u'_i$  or  $u'_i - u'_{i+1} \geq \delta$ .

The same eventually happens also when  $u_0$  is not a multiple of  $\delta$ , with the possible exception of the case when  $u'_i = u_0$  for each  $i \in \mathbb{N}$ . Similar considerations apply to the lower bounds. As a consequence, either  $\{[l_i, u_i]\}_i$  stabilizes, or it satisfies Proposition 4 and its limit is  $\emptyset$ .  $\square$

The next example applies the new weak narrowing  $\Delta^\delta$  to the program `realLoop`.

*Example 9* We compute the descending chain for the example program `realLoop` in Fig. 6(a) using  $\delta$ -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.  $\square$

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

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

$$I \Delta^{\delta^*} \emptyset = \emptyset$$

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

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 weak narrowing keeps iterating while the difference between two successive iterations is greater than  $\delta$ .

**Theorem 7** *The operator  $\Delta^{\delta^*}$  is a weak narrowing.*

*Proof* We need to show that for any  $[l_2, u_2] \leq [l_1, u_1]$ , it holds that  $[l_2, u_2] \leq [l_1, u_1] \Delta^{\delta^*} [l_2, u_2] \leq [l_1, u_1]$  and any infinite descending chain of narrowing either stabilizes or converge towards  $\emptyset$ .

Let  $[l', u'] = [l_1, u_1] \Delta^{\delta^*} [l_2, u_2]$ . By Definition 13, it is immediate to see that either  $l' = l_1$  or  $l' = l_2$ , and symmetrically for the upper bounds, from which we have that  $[l_2, u_2] \leq [l', u'] \leq [l_1, u_1]$

Given any chain  $\{[l_i, u_i]_{i \in \mathbb{N}}\}$ , where  $[l_{i+1}, u_{i+1}] \leq [l_i, u_i]$  for any  $i \in \mathbb{N}$ , we need to prove that the sequence:

$$[l'_0, u'_0] = [l_0, u_0]$$

$$[l'_{i+1}, u'_{i+1}] = [l'_i, u'_i] \Delta^{\delta^*} [l_{i+1}, u_{i+1}]$$

either stabilizes or converge towards  $\emptyset$ . The proof proceed as for Theorem 6.  $\square$

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

*Example 10* We compute the descending chain for the example program `realLoop` in Fig. 6(a) using  $\delta^*$ -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.  $\square$

## 9 Conclusion and related work

We believe that 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:

- defined a left-strict join operators for loop heads, which improves precision by preserving unreachability;
- proved that, when using the new join operator on a domain satisfying the bottom chain condition (such as any template abstract domain with integer bounds), the descending phase of the analysis terminates even without using a narrowing operator, provided the equation system comes from a reducible flow graph;
- defined an acceleration operation for descending iterations which immediately propagates unreachability; this is the first instance of an accelerator operator for the descending phase of the analysis;
- proved that, when using acceleration on an abstract domain satisfying the bottom chain condition, narrowing is not needed even for non-reducible graphs;
- provided several improved (more precise) weak narrowings for template domains with real bounds, 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 weak narrowings may be easily implemented in existent analyzers with little effort, since they only require a single check in the abstract join in order to make it left-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 weak narrowings over reals or for the idea of avoiding narrowing at all with integer domains. In this case, we may get better precision, as shown in Example 7, 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 cost of the repeated computations in 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 weak narrowings 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 [5, 7, 6] the authors propose to combine widening and narrowing during the analysis, resulting in multiple intertwined ascending and descending phases. In [5] the alternation between ascending and descending phases is driven by a hierarchical ordering [8] of the equation system, while in [7, 6] the alternation is driven by a choice of priorities for the

unknowns. Both these approaches, when paired with favourable ordering or priorities, are able to completely analyze the upper part (the first loop) of the program `unreachableLoop` in Fig. 1(a) before starting the analysis of the lower part. In this case, the infinite descending chain does not arise even with the standard join operator. However, it is not clear if a favourable ordering of equations or priorities exist for any program. This seems unlikely, especially for programs with complex nested loops or non reducible equation systems.

Moreover, [5] 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 [5], where restart is triggered only when unreachability is detected. For instance, in the example program `unreachableLoop` in Fig. 1(a), the restarting policy triggers a full analysis (widening and narrowing phases) of the second loop with an initial assignment which maps every unknown of the second loop to bottom. However, while in the previous work restarting is a feature of the equation solver, here it is realized directly at the semantic level.

In [18] the authors try to recover precision by restarting the analysis after that a post-fixpoint has been reached. An heuristic chooses a set of predecessors for each join node. The analysis is then restarted from an initial value which is equal to the result of the previous analysis for the chosen nodes, and bottom for all the others. For the example program `unreachableLoop` in Fig. 1(a) this does not help since, at the end of the first analysis,  $x_9$  is bottom anyway.

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.

While acceleration in the ascending phase (for numerical domains) has been introduced already in [17], the new acceleration operator is, up to our knowledge, the only example of acceleration to be used during the descending phase of the analysis.

## References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.: *Compilers: Principles, Techniques and Tools*, second edn. Addison Wesley (2006)
2. Amato, G., Di Maio, S.D.N., Meo, M.C., Scozzari, F.: Narrowing operators on template abstract domains. In: N. Bjørner, F. de Boer (eds.) *FM 2015: Formal Methods*, 20th International Symposium, Oslo, Norway, June 24–26, 2015, *Proceedings, Lecture Notes in Computer Science*, vol. 9109, pp. 57–72. Springer, Berlin Heidelberg (2015). DOI 10.1007/978-3-319-19249-9\_5
3. Amato, G., Parton, M., Scozzari, F.: Discovering invariants via simple component analysis. *Journal of Symbolic Computation* **47**(12), 1533–1560 (2012). DOI 10.1016/j.jsc.2011.12.052
4. Amato, G., Scozzari, F.: The abstract domain of parallelotopes. In: J. Midtgaard, M. Might (eds.) *Proceedings of the Fourth International Workshop on Numerical and Symbolic Abstract Domains, NSAD 2012, Electronic Notes in Theoretical Computer Science*, vol. 287, pp. 17–28. Elsevier (2012). DOI 10.1016/j.entcs.2012.09.003
5. Amato, G., Scozzari, F.: Localizing widening and narrowing. In: F. Logozzo, M. Fähndrich (eds.) *Static Analysis. 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20–22, 2013, Proceedings, Lecture Notes in Computer Science*, vol. 7935, pp. 25–42. Springer, Berlin Heidelberg (2013). DOI 10.1007/978-3-642-38856-9\_4
6. Amato, G., Scozzari, F., Seidl, H., Apinis, K., Vojdani, V.: Efficiently intertwining widening and narrowing. *Science of Computer Programming* **120**, 1–24 (2016). DOI 10.1016/j.scico.2015.12.005

7. Apinis, K., Seidl, H., Vojdani, V.: 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, pp. 377–386. ACM, New York, NY, USA (2013). DOI 10.1145/2491956.2462190
8. Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: D. Bjørner, M. Broy, I.V. Pototosin (eds.) Formal Methods in Programming and Their Applications, International Conference Academgorodok, Novosibirsk, Russia June 28 – July 2, 1993 Proceedings, *Lecture Notes in Computer Science*, vol. 735, pp. 128–141. Springer, Berlin Heidelberg (1993). DOI 10.1007/BFb0039704
9. Costan, A., Gaubert, S., Goubault, E., Martel, M., Putot, S.: A policy iteration algorithm for computing fixed points in static analysis of programs. In: K. Etessami, S.K. Rajamani (eds.) Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6–10, 2005. Proceedings, *Lecture Notes in Computer Science*, vol. 3576, pp. 462–475. Springer, Berlin Heidelberg (2005). DOI 10.1007/11513988\_46
10. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, pp. 106–130. Dunod, Paris, France (1976)
11. Cousot, P., Cousot, R.: 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, pp. 238–252. ACM Press, New York, NY, USA (1977). DOI 10.1145/512950.512973
12. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 269–282. ACM Press, New York, NY, USA (1979). DOI 10.1145/567752.567778
13. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: M. Bruynooghe, M. Wirsing (eds.) Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP '92 Leuven, Belgium, August 26–28, 1992, Proceedings, *Lecture Notes in Computer Science*, vol. 631, pp. 269–295. Springer, Berlin Heidelberg (1992). DOI 10.1007/3-540-55844-6\_101. Invited paper
14. Cousot, P., Halbwachs, N.: 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, pp. 84–97. ACM Press, New York, NY, USA (1978). DOI 10.1145/512760.512770
15. Gawlitza, T.M., Monniaux, D.: Invariant generation through strategy iteration in succinctly represented control flow graphs. *Logical Methods in Computer Science* **8**(3) (2012). DOI 10.2168/LMCS-8(3:29)2012
16. Gawlitza, T.M., Seidl, H.: Solving systems of rational equations through strategy iteration. *ACM Transactions on Programming Languages and Systems* **33**(3), 1–48 (2011). DOI 10.1145/1961204.1961207
17. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: K. Yi (ed.) Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29–31, 2006. Proceedings, *Lecture Notes in Computer Science*, vol. 4134, pp. 144–160. Springer, Berlin Heidelberg (2006). DOI 10.1007/11823230\_10
18. Halbwachs, N., Henry, J.: When the decreasing sequence fails. In: A. Miné, D. Schmidt (eds.) Static Analysis, 19th International Symposium, SAS 2012, Deauville, France, September 11–13, 2012. Proceedings, *Lecture Notes in Computer Science*, vol. 7460, pp. 198–213. Springer, Berlin Heidelberg (2012). DOI 10.1007/978-3-642-33125-1\_15
19. Miné, A.: The octagon abstract domain. *Higher-Order and Symbolic Computation* **19**(1), 31–100 (2006). DOI 10.1007/s10990-006-8609-1
20. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: R. Cousot (ed.) Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17–19, 2005. Proceedings, *Lecture Notes in Computer Science*, vol. 3385, pp. 25–41. Springer, Berlin Heidelberg (2005). DOI 10.1007/b105073
21. Simon, A., King, A., Howe, J.M.: Two variables per linear inequality as an abstract domain. In: M. Leuschel (ed.) Logic Based Program Synthesis and Transformation 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17–20, 2002. Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 2664, pp. 71–89. Springer, Berlin Heidelberg (2003). DOI doi://10.1007/3-540-45013-0\_7