Decoupling the Ascending and Descending Phases in Abstract Interpretation

Vincenzo Arceri$^{[0000-0002-5150-0393]1}$ and Isabella Mastroeni$^{[0000-0003-1213-536X]2}$ and Enea Zaffanella$^{[0000-0001-6388-2053]1}$

[1] University of Parma, Italy
{vincenzo.arceri,enea.zaffanella}@unipr.it
[2] University of Verona, Italy
isabella.mastroeni@univr.it

Abstract. Abstract Interpretation approximates the semantics of a program by mimicking its concrete fixpoint computation on an abstract domain $\mathbb{A}$. The abstract (post-) fixpoint computation is classically divided into two phases: the *ascending* phase, using widenings as extrapolation operators to enforce termination, is followed by a *descending* phase, using narrowings as interpolation operators, so as to mitigate the effect of the precision losses introduced by widenings. In this paper we propose a simple variation of this classical approach where, to more effectively recover precision, we *decouple* the two phases: in particular, before starting the descending phase, we replace the domain $\mathbb{A}$ with a more precise abstract domain $\mathbb{D}$. The correctness of the approach is justified by casting it as an instance of the A^{2}I framework. After demonstrating the new technique on a simple example, we summarize the results of a preliminary experimental evaluation, showing that it is able to obtain significant precision improvements for several choices of the domains $\mathbb{A}$ and $\mathbb{D}$.

Keywords: Abstract Interpretation, Static Analysis, Widening, Narrowing

1 Introduction

Abstract interpretation [17] is a framework for designing approximate semantics, with the aim of gathering information about programs in order to provide conservative/sound answers to questions about their run-time behaviors. In other words, the purpose of abstract interpretation is to formally design automatic program analyses by approximating program semantics for statically determining dynamic properties. The design of static analyzers consists in automatizing the computation of such approximations, and in this case the answer can only be partial or imprecise, due to the undecidability of program termination. Abstract/approximated semantics are computed by mimicking the monotonic (ascending) concrete semantics computation, obtained by Kleene iteration reaching fixpoint. Unfortunately, it is well known that Kleene fixpoint computation may not terminate. In the static analysis framework this issue has been tackled by introducing fixpoint accelerators, namely new operators (called widenings) built on the computational abstract domain, allowing to accelerate the fixpoint computation at the price of potentially reaching a post-fixpoint, namely at the price of losing precision in the answer. For this reason, it is common in static analysis to design another operator (called narrowing) performing a descending path in order to try to recover some precision by refining the reached post-fixpoint.

The precision of the result depends both on the ability of the widening operator to *guess* a limit of the increasing sequence, and on the information gathered during the decreasing phase. Intuitively, the increasing sequence extrapolates the behavior of the program from the first steps of its execution, while the decreasing sequence gathers information about the end of the execution of the program [13]. Moreover, a naive application of the classical approach may lead to an inadequate analysis, which is too expensive or too imprecise, meaning that there is a strong need for mechanisms that can effectively tune the precision/efficiency tradeoff. In order to improve this ratio, we could either improve efficiency (usually to the detriment of precision) by choosing a simpler (less precise) domain or by changing the fixpoint construction (e.g., replacing precise abstract operators with cheaper over-approximations), or improve precision (usually to the detriment of efficiency) by choosing a more precise/costly domain or again by changing the fixpoint construction (clearly in the opposite direction, see techniques discussed in Section 5).

In this work, we propose to combine these improvement approaches by choosing to use different domains (with different precision degrees) depending on the analysis phase: we use a potentially less precise domain in the fixpoint computation exploiting a widening operator for reaching a post-fixpoint in the ascending phase, and therefore potentially sensitively losing precision, and we use a more precise domain in the descending (narrowing) phase for trying to improve the gain of precision of such phase. The idea is rather simple but, to the best of our knowledge, it was never proposed before; also, since it is orthogonal with respect to similar approaches, it can be used in combination with them (rather than as an alternative to them). The intuition beyond the gain of precision without a relevant loss of efficiency is based on the idea that in the descending phase we do not need to use the more expensive operations. Such intuition is supported by our initial experimental evaluation, showing that the proposed approach is surely promising, being able to improve precision in a significant number of cases.

Paper structure. Section 2 gives basics in order theory, Abstract Interpretation, and the classical approach for static analysis by Abstract Interpretation. Section 3 presents our proposal for decoupling the ascending and descending phases with two different abstract domains. Section 4 reports a preliminary experimental evaluation of our approach. Section 5 discusses most related works. Section 6 concludes.

2 Background

Order theory. We denote by $\wp(S)$ the powerset of a set S. A poset $\langle L, \sqsubseteq_L \rangle$ is a set L equipped with a partial order $\sqsubseteq_L \in \wp(L \times L)$, i.e., a reflexive, transitive and anti-symmetric binary relation; in the following we will omit subscripts when clear from context. A poset is a join semi-lattice if, for each $l_1, l_2 \in L$, the lub (least upper bound) $l_1 \sqcup l_2$ belongs to L; similarly, it is a meet semi-lattice if the glb (greatest lower bound) $l_1 \sqcap l_2$ belongs to L; when both properties hold, we have a lattice $\langle L, \sqsubseteq, \sqcup, \sqcap \rangle$. A lattice is complete if $\forall X \subseteq L$, $\bigsqcup X$ and $\bigsqcap X$ belong to L; a complete lattice with bottom element $\bot$ and top element $\top$ is denoted $\langle L, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$. A poset $\langle L, \sqsubseteq \rangle$ satisfies the ascending chain condition (ACC) iff each infinite sequence $l_0 \sqsubseteq l_1 \sqsubseteq \cdots \sqsubseteq l_i \sqsubseteq \ldots$ of elements of L is not strictly increasing, i.e., $\exists k \geq 0, \forall j \geq k : l_k = l_j$. Dually the poset satisfies the descending chain condition (DCC) iff each infinite sequence $l_0 \sqsupseteq l_1 \sqsupseteq \cdots \sqsupseteq l_i \sqsupseteq \ldots$ of elements of L is not strictly decreasing, that is $\exists k \geq 0, \forall j \geq k : l_k = l_j$.

A function $f \colon L \to L$ on poset $\langle L, \sqsubseteq \rangle$ is monotone if, for all $l_1, l_2 \in L$, $l_1 \sqsubseteq l_2$ implies $f(l_1) \sqsubseteq f(l_2)$. We denote $\mathrm{post}(f)$ the set of post-fixpoints of f, i.e., those elements $x \in L$ satisfying $f(x) \sqsupseteq x$; similarly, $\mathrm{pre}(f)$ is the set of pre-fixpoints of f, satisfying $f(x) \sqsubseteq x$; the set of fixpoints of f, satisfying $f(x) = x$, is thus $\mathrm{fix}(f) = \mathrm{pre}(f) \cap \mathrm{post}(f)$. Given a function $f \colon L \to L$ we recursively define the iterates/iterations of f from $x \in L$ as $f^0(x) = x$ and $f^{i+1}(x) = f(f^i(x))$. The Kleene fixpoint theorem says that a continuous function $f \colon L \to L$ on a complete lattice $\langle L, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ has a least fixpoint $\mathrm{lfp}(f) \in L$, which can be obtained as the lub of the increasing sequence $f^0(\bot) \sqsubseteq f^1(\bot) \sqsubseteq \cdots \sqsubseteq f^i(\bot) \sqsubseteq \ldots$ [18].

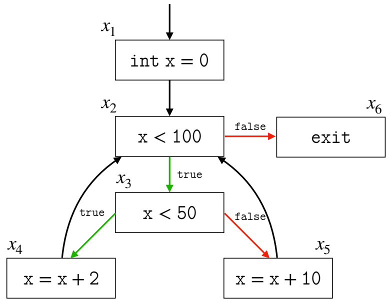
Abstract Interpretation (AI). Abstract Interpretation [17, 18] is a theory to soundly approximate program semantics, focusing on some run-time property of interest. In the classical setting, the concrete and the abstract semantics are defined over two complete lattices, respectively called the concrete domain C and the abstract domain A. A pair of monotone functions $\alpha : C \to A$ and $\gamma : A \to C$ forms a *Galois Connection* (GC) if $\forall c \in C, \forall a \in A : \alpha(c) \sqsubseteq_A a \Leftrightarrow c \sqsubseteq_C \gamma(a)$. If C and A are related by a GC, denoted $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$, then an abstract function $f_A \colon A \to A$ is a correct approximation of a concrete function $f_C \colon C \to C$ if and only if $\forall c \in C : \alpha(f_C(c)) \sqsubseteq_A f_A(\alpha(c))$ or equivalently $\forall a \in A : f_C(\gamma(a)) \sqsubseteq_C \gamma(f_A(a))$; the *best correct approximation* of f_C is $f_A^\sharp = (\alpha \circ f_C \circ \gamma)$.

Static program analysis via abstract interpretation. It is possible to represent a program of interest as a control-flow graph (CFG for short). A CFG is a graph $\langle N, E \rangle$ such that $N = \{n_1, n_2, \ldots, n_m\}$ is a finite set of nodes corresponding to the control points of the program, and $E \subseteq N \times N$ is a finite set of edges. It is possible to compute the CFG associated with a certain program with standard techniques [37].

Let us denote by A the abstract domain approximating the concrete domain C, used to analyze programs of interest. With each node $n \in N$ is associated a function transformer $f_n : A^m \to A$ capturing the effects of the node

```
int main() {
  int x = 0;
  while (x < 100)
  if (x < 50)
    x = x + 2;
  else
    x = x + 10;
}
```

(a)



(b)

$$x_1 = \top \qquad x_3 = x_2 \qquad x_4 = x_3 \qquad x_5 = x_3 \qquad x_6 = x_2$$
$$x_2 = \mathsf{Even} \sqcup (x_4 +_{\mathsf{Par}} \mathsf{Even}) \sqcup (x_5 +_{\mathsf{Par}} \mathsf{Even})$$

(c)

Fig. 1: (a) C function example, (b) associated CFG, (c) associated system of equations with the Par abstract domain.

n, i.e., the abstract semantics. Analyzing a given CFG $C = \langle N, E\rangle$, where $N = \{n_1, n_2, \ldots, n_m\}$ means to resolve the following system of equations[3]

$$F = \left\{ x_i = f_i(x_1, x_2, \ldots, x_m) \;\middle|\; i = 1, 2, \ldots, m \right\}$$

The goal of AI-based static analysis, using an abstract domain A, is to compute the least solution of the equation set F as the limit of a Kleene iteration on A, i.e., $x^{\mathrm{lfp}(F)} \triangleq (x_1^{\mathrm{lfp}(F)}, \ldots, x_m^{\mathrm{lfp}(F)})$ starting from the bottom elements of A, i.e., $\forall i \in [1, m].\, x_i = \bot$.

Example 1. Consider the C function of Figure 1a and the corresponding CFG, shown in Figure 1b. We intuitively describe the analysis of this program using the abstract domain Par [18, Example 10.1.0.3], tracking the parity of numerical variables:

$$\mathsf{Par} = \left\langle \{\bot, \top, \mathsf{Even}, \mathsf{Odd}\}, \sqsubseteq, \sqcup, \sqcap, \bot, \top \right\rangle,$$

where the partial order is defined by $\bot \sqsubseteq x \sqsubseteq \top$, for each $x \in \mathsf{Par}$. The system of equations is reported in Figure 1c; Note that the equation defining x_i is intuitively describing the values that are possibly *entering* the corresponding node of the CFG (also labeled x_i for convenience); for instance, the right hand side of the equation defining x_2 computes the lub of the abstract values *exiting* from nodes x_1, x_4 and x_5, respectively. For space reasons, we leave to intuition

[3] In general, the least fixpoint on the concrete domain C is not finitely computable. Hence, the idea is to compute an abstract fixpoint, over an abstract domain A, that correctly approximates the concrete one.

the abstract functions modeling the semantics of each CFG node (e.g., function $+_{\mathsf{Par}} \colon \mathsf{Par} \times \mathsf{Par} \to \mathsf{Par}$ modeling addition on the Par domain). The least solution for the system is $x_1 = \top$ and $x_i = \mathsf{Even}$ for $i = 2, \ldots, 6$. □

The ascending sequence over the system of equations F may fail to (finitely) converge for abstract domains that do not satisfy the ACC. A converge guarantee can be provided by widening operators, which over-approximate the least fixpoint solution $x^{\mathrm{lfp}(F)}$ by effectively computing a post-fixpoint of F. A widening $\nabla \colon A \times A \to A$ is an operator such that:

- for each $a_1, a_2 \in A$, $a_1 \sqsubseteq a_1 \nabla a_2$ and $a_2 \sqsubseteq a_1 \nabla a_2$;
- for all ascending sequences $a_0 \sqsubseteq \cdots \sqsubseteq a_{i+1} \sqsubseteq \ldots$, the ascending sequence $x_0 \sqsubseteq \cdots \sqsubseteq x_{i+1} \sqsubseteq \ldots$ defined by $x_0 = a_0$ and $x_{i+1} = x_i \nabla a_{i+1}$ is not strictly increasing.

In principle, widening can be applied to all equations of the system F, which however would lead to a gross over-approximation; following [12], it is sufficient that the widening is applied on one node in each cycle of the CFG; for instance, in Figure 1b we can use x_3 as the one and only widening point. We denote $\mathit{WP} \subseteq N$ the set of *widening points*, i.e., the nodes of the CFG where widening is applied, leading to the system of equations F^{∇}:

$$\begin{cases} x_i = x_i \nabla f_i(x_1, x_2, \ldots, x_m), & \text{if } i \in \mathit{WP}; \\ x_i = x_i \sqcup f_i(x_1, x_2, \ldots, x_m), & \text{otherwise.} \end{cases} \tag{1}$$

In order to mitigate the loss of precision introduced by widenings, the *ascending phase* computing the post-fixpoint x^{∇} of F can be followed by another Kleene iteration on the system F, starting from x^{∇} and descending towards a fixpoint of F (not necessarily the least one). If the abstract domain A does not satisfy the DCC, this descending sequence may fail to converge; a convergence guarantee can be obtained by using a narrowing operator $\Delta \colon A \times A \to A$, satisfying:

- for each $a_1, a_2 \in A$, $a_1 \sqsupseteq a_1 \Delta a_2 \sqsupseteq a_1 \sqcap a_2$;
- for all descending sequences $a_0 \sqsupseteq \cdots \sqsupseteq a_{i+1} \sqsupseteq \ldots$, the descending sequence $x_0 \sqsupseteq \cdots \sqsupseteq x_{i+1} \sqsupseteq \ldots$ defined by $x_0 = a_0$ and $x_{i+1} = x_i \Delta a_{i+1}$ is not strictly decreasing.

As before, the application of narrowings can be limited to WP, leading to the system of equations F^{Δ} used during the *descending phase*:

$$\begin{cases} x_i = x_i \Delta f_i(x_1, x_2, \ldots, x_m), & \text{if } i \in \mathit{WP}; \\ x_i = x_i \sqcap f_i(x_1, x_2, \ldots, x_m), & \text{otherwise.} \end{cases} \tag{2}$$

In general, the descending sequence with narrowing will compute a post-fixpoint x^{Δ} of F (not necessarily a fixpoint), satisfying $x^{\Delta} \sqsubseteq x^{\nabla}$. A graphical representation of the ascending and descending phases over the abstract domain A is

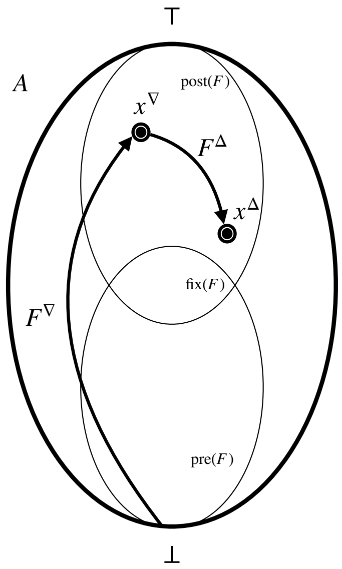


Fig. 2: The ascending and descending phases over abstract domain A.

reported in Figure 2. Note that a “glb-based” narrowing operator can be easily defined by computing the domain glb and forcing the descending sequence to stop as soon as reaching a fixed, finite number $k \in \mathbb{N}$ of iterations. For this reason, several abstract domains do not implement a proper narrowing operator.

Example 2. The domain of integral intervals [22] (or 1-dimension integral boxes) is an example of complete lattice satisfying neither the ACC nor the DCC:

$$\mathsf{Itv} = \{\bot, \top\} \cup \big\{ [\ell, u] \bigm| \ell, u \in \mathbb{Z}, \ell \leq u \big\} \cup \big\{ [-\infty, u] \bigm| u \in \mathbb{Z} \big\} \cup \big\{ [\ell, +\infty] \bigm| \ell \in \mathbb{Z} \big\},$$

where $\bot$ is the bottom element (denoting the empty interval), $\top = [-\infty, +\infty]$ is the top element (denoting $\mathbb{Z}$) and the partial order, lub and glb operators consistently model the usual containment relation. The interval widening operator [22] $\nabla \colon \mathsf{Itv} \times \mathsf{Itv} \to \mathsf{Itv}$ is defined, for each $x \in \mathsf{Itv}$, by $\bot \mathbin{\nabla} x = x \mathbin{\nabla} \bot = x$ and

$$[\ell_0, u_0] \mathbin{\nabla} [\ell_1, u_1] = [(\ell_1 < \ell_0 \mathbin{?} -\infty : \ell_0), (u_0 < u_1 \mathbin{?} +\infty : u_0)].$$

Similarly, the interval narrowing operator [22] $\Delta \colon \mathsf{Itv} \times \mathsf{Itv} \to \mathsf{Itv}$ is defined, for each $x \in \mathsf{Itv}$, by $\bot \mathbin{\Delta} x = x \mathbin{\Delta} \bot = \bot$ and

$$[\ell_0, u_0] \mathbin{\Delta} [\ell_1, u_1] = [(\ell_0 = -\infty \mathbin{?} \ell_1 : \ell_0), (u_0 = +\infty \mathbin{?} u_1 : u_0)].$$

Considering again the C function in Figure 1a, the corresponding system of equations for the domain Itv is shown in Figure 3a (where $+_{\mathsf{Itv}} \colon \mathsf{Itv} \times \mathsf{Itv} \to \mathsf{Itv}$ models addition on the Itv domain). The computation of the ascending and

$$\begin{aligned} &x_1 = \top && x_4 = x_3 \sqcap [-\infty, 49] \\ &x_2 = [0,0] \sqcup (x_4 +_{\mathsf{Itv}} [2,2]) \sqcup (x_5 +_{\mathsf{Itv}} [10,10]) && x_5 = x_3 \sqcap [50, +\infty] \\ &x_3 = x_2 \sqcap [-\infty, 99] && x_6 = x_2 \sqcap [100, +\infty] \end{aligned}$$

(a)

		ascending phase iter		descending phase iter	
N	WP	1st	2nd (= $x^{\nabla}_{\mathsf{Itv}}$)	1st	2nd (= $x^{\Delta}_{\mathsf{Itv}}$)
x_1		$\top$	$\top$	$\top$	$\top$
x_2		$[0,0]$	$[0,+\infty]$	$[0,+\infty]$	$[0,109]$
x_3	✓	$[0,0]$	$[0,+\infty]$	$[0,99]$	$[0,99]$
x_4		$[0,0]$	$[0,49]$	$[0,49]$	$[0,49]$
x_5		$\bot$	$[50,+\infty]$	$[50,99]$	$[50,99]$
x_6		$\bot$	$[100,+\infty]$	$[100,+\infty]$	$[100,109]$

(b)

Fig. 3: (a) Equations for CFG in Figure 1b using Itv, (b) interval results.

descending sequences is shown in Figure 3b, where in the 2nd column we have highlighted the only widening point x_3; in particular, the 4th and 6th columns show the post-fixpoint and the fixpoint obtained at the end of the ascending and the descending phases, respectively. □

Powerset domains. Many abstract domains (e.g., numerical domains whose elements are convex sets) are unable to precisely describe disjunctive information, thereby incurring significant precision losses whenever the abstract semantic construction needs to merge different control flow paths. To avoid these losses, it is possible to lift the domain using a disjunctive domain refinement operator [18]. In the following we will consider the finite powerset [6] of an abstract domain A, which is the join-semilattice $\mathsf{Set}_{\mathrm{fn}}(A) = \langle \wp_{\mathrm{fn}}(A), \sqsubseteq_{\mathrm{fn}}, \sqcup_{\mathrm{fn}}, \bot_{\mathrm{fn}} \rangle$, where:

- the carrier $\wp_{\mathrm{fn}}(A)$ is the set of the *finite* and *non-redundant* subsets of A (an element $a_1 \in A$ is redundant in $S \subseteq A$ iff $a_1 = \bot_A$ or $\exists a_2 \in S \,.\, a_1 \sqsubset_A a_2$);
- the partial order $S_1 \sqsubseteq_{\mathrm{fn}} S_2$ is defined by $\forall a_1 \in S_1, \exists a_2 \in S_2 \,.\, a_1 \sqsubseteq_A a_2$;
- the (binary) least upper bound $S_1 \sqcup_{\mathrm{fn}} S_2$ is computed by removing the redundant elements from the set union $S_1 \cup S_2$;
- the bottom element is $\bot_{\mathrm{fn}} = \emptyset$.

For space reasons we omit a more thorough discussion of powerset domains (e.g., the lifting of the abstract semantic operators defined on A), referring the interested reader to [6, 18].

3 Decoupling the Ascending and Descending Phases

In the previous section we have recalled the classical approach used in static analysis based on abstract interpretation, which can be summarized as follows:

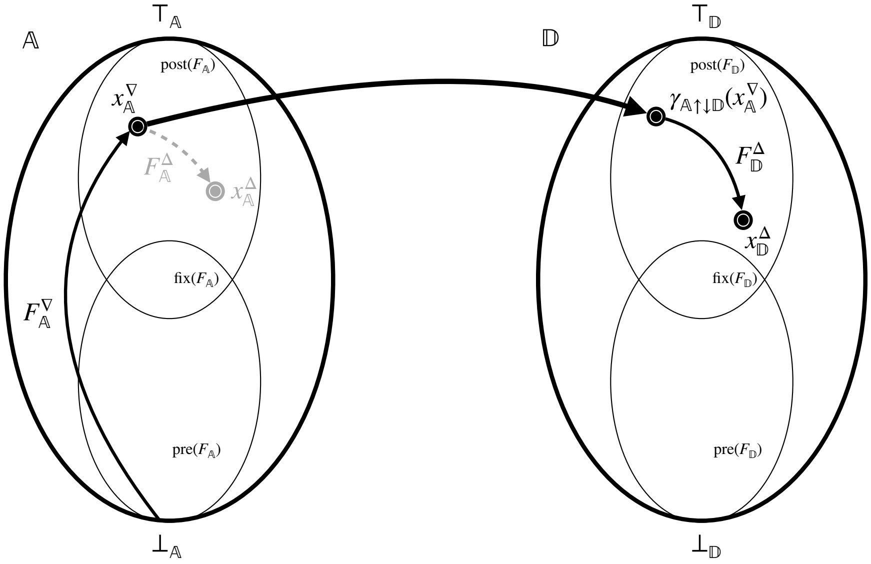

$\mathbb{A}$
$\top_{\mathbb{A}}$
$\mathrm{post}(F_{\mathbb{A}})$
$x_{\mathbb{A}}^{\nabla}$
$F_{\mathbb{A}}^{\Delta}$
$x_{\mathbb{A}}^{\Delta}$
$\mathrm{fix}(F_{\mathbb{A}})$
$F_{\mathbb{A}}^{\nabla}$
$\mathrm{pre}(F_{\mathbb{A}})$
$\bot_{\mathbb{A}}$
$\mathbb{D}$
$\top_{\mathbb{D}}$
$\mathrm{post}(F_{\mathbb{D}})$
$\gamma_{\mathbb{A}\uparrow\downarrow\mathbb{D}}(x_{\mathbb{A}}^{\nabla})$
$F_{\mathbb{D}}^{\Delta}$
$x_{\mathbb{D}}^{\Delta}$
$\mathrm{fix}(F_{\mathbb{D}})$
$\mathrm{pre}(F_{\mathbb{D}})$
$\bot_{\mathbb{D}}$

Fig. 4: The ascending and descending phases over $\mathbb{A}$ and $\mathbb{D}$, respectively.

(a) fix an abstract domain A such that $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ and a corresponding, correct system of abstract equations F_A; (b) approximate the concrete semantics by computing a post-fixpoint of F_A in the ascending phase (with widening); (c) improve the result in the descending phase (with narrowing). What is worth noting is that the two phases (b) and (c) are computed on the same domain A.

Moving from the observation that the only goal of the descending phase is to improve precision, we propose to *decouple* it from the ascending phase: that is, we compute the descending sequence on a different, more precise abstract domain, so as to increase the chances of a significant precision improvement. Clearly, the adoption of a more precise domain likely incurs some penalty in terms of the efficiency of the analysis; however, since in our proposal this domain is only used in the descending phase, it should be simpler to achieve a good tradeoff between precision and efficiency, because the descending phase can be stopped after any number of iterations and still provide a correct result.

In the following we will denote $\mathbb{A}$ and $\mathbb{D}$ the abstract domains used in the ascending and descending phases, respectively, and use the notation $\mathbb{A}{\upharpoonleft\!\downharpoonright}\mathbb{D}$ to refer to this decoupled approach. The correctness/precision relation between the concrete domain and the two abstract domains is formalized by requiring $C \underset{\alpha_{\mathbb{D}}}{\overset{\gamma_{\mathbb{D}}}{\leftrightarrows}} \mathbb{D} \underset{\alpha_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\overset{\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\leftrightarrows}} \mathbb{A}$; we also require that the concretization function $\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}$ is effectively computable.

Our decoupled approach is graphically represented in Figure 4. The (concrete) system of equations F_C is correctly approximated on domain $\mathbb{D}$ by the (abstract) system of equations $F_{\mathbb{D}}$, which is further approximated on domain $\mathbb{A}$ by the system of equations $F_{\mathbb{A}}$. We first compute a post-fixpoint $x_{\mathbb{A}}^{\nabla} \in \mathbb{A}$ using the system of equations $F_{\mathbb{A}}^{\nabla}$ (with widening); instead of descending on the same abstract domain, as done in Section 2, we transfer the post-fixpoint $x_{\mathbb{A}}^{\nabla}$ to the more precise domain $\mathbb{D}$, using the concretization function $\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}$ (which is computable); hence, the descending phase will use the system of equations $F_{\mathbb{D}}^{\Delta}$ (with narrowing) on domain $\mathbb{D}$, starting from $\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}(x_{\mathbb{A}}^{\nabla})$ and obtaining an improved post-fixpoint $x_{\mathbb{D}}^{\Delta} \in \mathbb{D}$.

The next lemma states that the post-fixpoint $x_{\mathbb{A}}^{\nabla} \in \mathbb{A}$ corresponds to a post-fixpoint for $F_{\mathbb{D}}$, necessary for starting the descending phase on $\mathbb{D}$.

Lemma 1. *Consider* $\mathbb{D} \underset{\alpha_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\overset{\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\leftrightarrows}} \mathbb{A}$ *and let* $F_{\mathbb{A}} \colon \mathbb{A} \to \mathbb{A}$ *be a correct approximation of* $F_{\mathbb{D}} \colon \mathbb{D} \to \mathbb{D}$. *Then,*

$$x_{\mathbb{A}}^{\nabla} \in \operatorname{post}(F_{\mathbb{A}}) \implies \gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}(x_{\mathbb{A}}^{\nabla}) \in \operatorname{post}(F_{\mathbb{D}}).$$

It would be desirable to prove that the final result $x_{\mathbb{D}}^{\Delta}$ obtained when using the decoupled approach $\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}$ systematically improves on the final result $x_{\mathbb{A}}^{\Delta}$ obtained by the classical approach, i.e., $x_{\mathbb{D}}^{\Delta} \sqsubseteq_{\mathbb{D}} \gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}(x_{\mathbb{A}}^{\Delta})$. However, in general this property does not hold, due to the use of different, unrelated, possibly non-monotonic narrowing operators on the domains $\mathbb{A}$ and $\mathbb{D}$. We can prove the desired result provided we force both domains to use the glb-based narrowing (with the same threshold value).

Proposition 1. *Consider* $\mathbb{D} \underset{\alpha_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\overset{\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}}{\leftrightarrows}} \mathbb{A}$ *and let* $F_{\mathbb{A}} \colon \mathbb{A} \to \mathbb{A}$ *be a correct approximation of* $F_{\mathbb{D}} \colon \mathbb{D} \to \mathbb{D}$*; let also* $x_{\mathbb{A}}^{\nabla} \in \operatorname{post}(F_{\mathbb{A}})$. *Then, for each* $k \in \mathbb{N}$*,*

$$F_{\mathbb{D}}^{k}(\gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}(x_{\mathbb{A}}^{\nabla})) \sqsubseteq_{\mathbb{D}} \gamma_{\mathbb{A}\upharpoonleft\downharpoonright\mathbb{D}}(F_{\mathbb{A}}^{k}(x_{\mathbb{A}}^{\nabla})).$$

Note that Lemma 1 and Proposition 1 are well-known results. Intuitively, the correctness of the decoupled approach is easily justified by viewing it as an instance of the A^{2}I framework [20]: starting from a classical analysis using the more precise domain $\mathbb{D}$, we further abstract part of its computation (the ascending phase), approximating it on domain $\mathbb{A}$.

On the Galois Connection requirement. When formalizing our decoupled proposal, we have assumed that all the considered domains (concrete, ascending and descending) are related by GCs: this corresponds to an ideal situation where for each element x of the more precise domain (resp., each semantic transformer f) we can identify the corresponding best correct approximation on the less precise domain $\alpha(x)$ (resp., $\alpha \circ f \circ \gamma$). However, there are well-known abstract domains (e.g., the domain convex polyhedra [21] approximating sets of reals or the deterministic finite-state automata domain [4] approximating sets of strings) that cannot be related to the concrete domain using a GC. This is not a real concern because, as discussed at length in [19], one can adopt a slightly weaker theoretical framework and still ensure the correctness of the analysis. As a matter of fact, in the experimental evaluation we will implicitly relax the GC assumption.

	transfer	descending phase		
N	$\gamma_{\mathsf{Itv}\Uparrow\!\!\downarrow\mathsf{ISet}}(x^{\nabla}_{\mathsf{Itv}})$	1st iter	2nd iter	
x_1	$\{\top_{\mathsf{Itv}}\}$	$\{\top_{\mathsf{Itv}}\}$	$\{\top_{\mathsf{Itv}}\}$	
x_2	$\{[0,+\infty]\}$	$\{[0,0],[2,51],[60,+\infty]\}$	$\{[0,0],[2,2],[4,51],[60,61],[70,109]\}$	✓
x_3	$\{[0,+\infty]\}$	$\{[0,0],[2,51],[60,99]\}$	$\{[0,0],[2,2],[4,51],[60,61],[70,99]\}$	✓
x_4	$\{[0,49]\}$	$\{[0,0],[2,49]\}$	$\{[0,0],[2,2],[4,49]\}$	✓
x_5	$\{[50,99]\}$	$\{[50,51],[60,99]\}$	$\{[50,51],[60,61],[70,99]\}$	✓
x_6	$\{[100,+\infty]\}$	$\{[100,+\infty]\}$	$\{[100,109]\}$	

Fig. 5: Computing the descending phase on $\mathsf{Itv}\Uparrow\!\!\downarrow\mathsf{ISet}$.

Example 3. We now reconsider the C program in Figure 1a and show how the decoupled approach can be used to improve on the results computed by the classical analysis on domain Itv (see Example 2 and Figure 3b). To this end, while keeping the domain Itv for the ascending phase, we will compute the descending phase on the powerset domain $\mathsf{ISet} = \mathsf{Set}_{\mathrm{fn}}(\mathsf{Itv})$, i.e., we will adopt the combination $\mathsf{Itv}\Uparrow\!\!\downarrow\mathsf{ISet}$.

Before starting the descending phase, the post-fixpoint $x^{\nabla}_{\mathsf{Itv}}$ computed in the ascending phase using Itv (see the 4th column of Figure 3b) is transferred to ISet using $\gamma_{\mathsf{Itv}\Uparrow\!\!\downarrow\mathsf{ISet}} \colon \mathsf{Itv} \to \mathsf{ISet}$, obtaining the (singleton) sets of intervals shown in the 2nd column of Figure 5. Then the descending phase on ISet is started: note that we use the glb-based narrowing, with a threshold value $k = 2$ on the number of iterations; the results computed by the two iterations are shown in the 3rd and 4th columns of Figure 5.

It is now possible to perform a precision comparison of the results obtained on domain Itv using the classical approach (last column in Figure 3b) with respect to the results obtained with the $\mathsf{Itv}\Uparrow\!\!\downarrow\mathsf{ISet}$ combination (4th column of Figure 5): for convenience, in the last column we show a checkmark (✓) on the CFG nodes where we actually obtain a precision improvement. Note that the post-fixpoint computed on ISet is not a fixpoint: hence, the precision could be refined further by increasing the threshold value $k \in \mathbb{N}$. □

4 Experimental Evaluation

In order to obtain a preliminary experimental evaluation of the precision gains resulting from the proposed analysis technique, we have modified the open source static analysis tool PAGAI [32] to allow for decoupling the ascending and descending iteration phases; in particular, we have added program options to select a different abstract domain for the descending phase, as well as to select a threshold value for the number of descending iterations (this threshold is set to 3).[4] In our experiments we configured PAGAI to perform a *simple* static analysis:

[4] By design, PAGAI does not use proper narrowing operators to enforce the termination of the decreasing sequence; rather, it stops when the iteration count reaches the threshold value (or earlier, if a fixpoint is detected).

hence, we disregard more sophisticated approaches, such as *path focusing*, and we disabled those LLVM bitcode instrumentation passes that heavily modify the CFG in order to potentially detect overflows and other runtime errors.

PAGAI is interfaced with the Apron library [33], which provides several numeric domains, among which boxes [17] (Box), octagons [35] (Oct) and convex polyhedra [21] (Pol); these non-disjunctive domains all implement the corresponding "standard" widening operator. We extended Apron interface by adding a modified version of the PPLite library [10] which, besides its enhanced implementation of the domain of convex polyhedra [7, 8], also includes a prototype implementation of the finite powerset $\mathsf{PSet} = \mathsf{Set}_{\mathrm{fn}}(\mathsf{Pol})$ of convex polyhedra [6]; since this prototype is not (yet) provided with a widening operator, it can only be used in the descending phase of the analysis. Note that PAGAI features a variant analysis technique that is meant to compute disjunctive invariants, but this would yield an analysis which is quite different from the direct adoption of a powerset domain; for instance, one would be forced to choose in advance the maximal number of disjuncts that are allowed.

The experimental evaluation considers 35 C source files distributed with PAGAI, which are variants of benchmarks taken from the SNU real-time benchmark suite for worst-case execution time analysis. PAGAI can be configured to perform a precision comparison among two different abstract domains (DOM_1 and DOM_2): in this case, the analyzer records the invariant properties computed by the two domains for each widening point (WP); then it compares them and provides a final report made of four numbers, counting the widening points on which the invariant computed by the first domain is, respectively, equivalent (EQ), stronger (LT), weaker (GT) and uncomparable (UN) with respect to the invariant computed by the second domain. The results of the precision comparisons have been summarized in Tables 1 and 2; note that, for readability, the tables show the percentages of widening points, rather than absolute values.[5]

		% WP				
DOM_1	DOM_2	EQ	LT	GT	UN	ΔEQ
Box	Oct	**66.5**	0.7	32.4	0.4	
Box	Box↿↓Oct	83.6	0.4	**16.0**	0.0	
Box↿↓Oct	Oct	**73.0**	0.7	26.3	0.0	**6.4**
Box	Pol	**53.7**	3.6	37.0	5.7	
Box	Box↿↓Pol	76.5	0.4	**23.1**	0.0	
Box↿↓Pol	Pol	**59.8**	5.7	31.3	3.2	**6.0**
Oct	Pol	**69.4**	6.8	21.4	2.5	
Oct	Oct↿↓Pol	87.2	0.0	**12.8**	0.0	
Oct↿↓Pol	Pol	**72.2**	8.2	18.5	1.1	**2.8**

Table 1: Precision comparison for non-disjunctive domains.

[5] The total number of widening points is 281.

Consider first Table 1, which is meant to evaluate the effectiveness of the new approach when both abstract domains are non-disjunctive. Note that the rows in the table are divided in three groups (three rows per group); let us focus on the first group, which is evaluating the precision improvements obtained when using the abstract domain Box⇅Oct. The first row in the group provides the *baseline* for the precision comparison: in particular, the value in column EQ (highlighted in boldface) informs us that the domain Box achieves the same precision as Oct on 66.5% of the widening points; this means that only the remaining 33.5% of widening points are further *improvable*. The second row in the group, in particular the value in column GT, shows us that Box⇅Oct is able to improve the precision of Box on 16% of all widening points. The third row in the group, in particular the value in column EQ, informs us that Box⇅Oct is able to achieve the same precision of Oct on 73% of the widening points: this corresponds to an increase by 6.4% (reported in the column labeled ΔEQ) with respect to the baseline EQ value (in the first row).

It is worth stressing that the percentages highlighted in the second and third row of the group are computed with respect to the total number of widening points, which might mislead the reader towards an underestimation of the effectiveness of the approach. One should observe that a precision gain on 16.0% of *all* the widening points corresponds to a precision gain on almost one half (16.0/33.5 = 47.9%) of the *improvable* widening points. The same reasoning applies to the 6.4% value of ΔEQ, which corresponds to almost 20% of the improvable widening points.

Similar observations can be derived from the second and third group of rows in Table 1, where we evaluate the abstract domain combinations Box⇅Pol and Oct⇅Pol, respectively. For instance, the third group of rows in Table 1 informs us that Oct⇅Pol is able to improve precision on 12.8% of all the widening points with respect to Oct and that it increases by 2.8% the percentage of widening points on which the same precision as Pol is obtained.

		% WP				Time (s)	
DOM$_1$	DOM$_2$	EQ	LT	GT	UN	DOM$_1$	DOM$_2$
Box	Box⇅PSet	52.3	0.4	**47.3**	0.0	6.20	6.76
Oct	Oct⇅PSet	56.2	0.0	**43.8**	0.0	12.70	8.93
Pol	Pol⇅PSet	64.8	0.0	**35.2**	0.0	7.03	7.53

Table 2: Precision comparison when using PSet in the descending phase.

In Table 2 we provide the summary for the precision comparisons between the three non-disjunctive domains Box, Oct and Pol and the corresponding enhanced combinations using the finite powerset of polyhedra PSet in the descending phase. Note that, in contrast with what we did in Table 1, in this case we cannot provide a *baseline* comparison with PSet because, as said before, this domain is missing a widening operator and hence cannot be used in the ascending phase of the analysis. The use of a powerset domain in the descending phase is of particular interest because it should be able to avoid the over-approximations that are incurred by the non-disjunctive domains when merging control flow paths. In fact, the values in column GT show us that the number of widening points where a precision improvement is obtained is significantly higher than those of Table 1, ranging from 35.2% to 47.3%. In summary, the results in Tables 1 and 2 provide an evidence that the adoption of a more precise abstract domain in the descending phase of the analysis is able to significantly improve precision. Intuitively, this is due to the fact that, by changing the abstract domain, we are potentially improving the precision of all the abstract semantic operators used in the descending phase (i.e., all operators except widening).

Note that we do not perform a proper efficiency comparison, because the considered benchmark suite seems inadequate to the purpose; also, PAGAI is a static analyzer meant to simplify experiments, rather than achieve maximum efficiency. Hence, we merely report in the last two columns of Table 2 the overall time spent on the 35 tests. A meaningful efficiency comparison will be the subject of future work.

A note on the relative precision of abstract domains. A non-expert but attentive reader may be wondering how it is possible that the more precise abstract domain Pol can sometimes compute weaker invariants when compared to the less precise domain Box (more generally, why column LT is not always zero). A first reason is that widening operators are not monotonic; another reason is that the two domains may be adopting different approximation strategies for some of the semantic operators (e.g., when modeling non-linear tests/assignments and when taking into account the integrality of program variables).

A technical note on the precision comparison. When comparing the invariants computed by different abstract domains, PAGAI calls a third-party model checking tool based on SMT (Satisfiability Modulo Theory), which also takes into account the integrality of program variables. Hence, when comparing the abstract elements of different domains, we are *not* counting those “dummy” precision improvements that are simply induced by the real relaxation step. As a concrete example, when x is an integral variable, the Box value $x \in [0, 2]$, the Pol value $\{0 \leq x \leq 2\}$ and the PSet value $\{ \{x = 0\}, \{x = 1\}, \{x = 2\} \}$ are all considered equivalent (note that the last two would not be considered equivalent when compared in the more precise domain PSet).

4.1 A detailed example

In Figure 6 we show a simplified version of one of the tests distributed with PAGAI; assuming $N > 1$, function `fib(N)` computes the $(N + 1)$-th element of the Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, \ldots$. In Table 3 we show the abstract values computed for the one and only widening point (whose position in the code is highlighted using a comment), first with the classical Box domain and then using the Box⇅PSet combination, i.e., using the finite powerset of polyhedra in

```
int fib(int N) {
  int P = 0, F = 1;
  for (int K = 2; K < N; ++K) {
    /* widening point */
    int tmp = F;
    F += P;
    P = tmp;
  }
  return F;
}

int main() { return fib(7); }
```

Fig. 6: A simplified version of `fibcall.c`

the descending phase. For this example, the threshold value for the number of downward iterations is set to 10.

When using the Box domain, the ascending phase ends on the 4th iteration: due to the use of widenings, the computed post-fixpoint has no upper bound for variable K; the upper bound 6 is easily recovered in the first iteration of the descending phase, which is then detected to be an abstract fixpoint on Box. When using the Box⇅PSet combination, the ascending phase is computed exactly as before but, before starting the descending phase, the post-fixpoint on Box is transferred to the PSet domain using the concretization function $\gamma\colon \mathsf{Box} \to \mathsf{PSet}$, obtaining a singleton set of polyhedra (see row labeled 'dsc/0'). Then the analysis proceeds by computing the descending iterates using the more precise domain PSet; the descending sequence is able to improve precision by computing several disjuncts, detecting the fixpoint on the 6th downward iteration.

It is worth stressing that, in this specific example, the descending sequence is able to reach a fixpoint on PSet only because function `fib` is called with a constant argument ($N = 7$). If instead the value of the argument was unknown, the descending sequence on PSet would be non-stabilizing, generating a new disjunct at each iteration. This is not a real issue because, as we already said, once started the descending phase the static analysis can be stopped at any iteration and still preserve correctness; a precision improvement with respect to the Box decreasing sequence is obtained even when computing a single downward iterate. Note that, for this detailed example, we have chosen the domain combination Box⇅PSet and the constant value $N = 7$ merely for exposition purposes, since the computed abstract values turn out to be simpler. For instance, if using the combination Box⇅Pol and stopping after the 3rd downward iteration, we would obtain as post-fixpoint the following abstract value:

$$\begin{aligned}\{2 \leq K \leq 6, 3K - 3P + 2F \geq 8, 7K - 7P - 13F \leq 1, K + 12P - 8F \leq 7,\\ K + 4P - 4F \leq 3, 3K - 16P + 8F \leq 14, 3K - 6P - 2F \leq 4\}.\end{aligned}$$

domain	phase/iter	abstract value
Box	asc/1	$P \in [0,0], F \in [1,1], K \in [2,2]$
Box	asc/2	$P \in [0,+\infty], F \in [1,1], K \in [2,+\infty]$
Box	asc/3	$P \in [0,+\infty], F \in [1,+\infty], K \in [2,+\infty]$
Box	asc/4	same value (detected post-fixpoint in Box)
Box	dsc/1	$P \in [0,+\infty], F \in [1,+\infty], K \in [2,6]$
Box	dsc/2	same value (detected fixpoint in Box)
PSet	dsc/0	$\{\ \{P \geq 0, F \geq 1, K \geq 2\}\ \}$
PSet	dsc/1	$\{\ \{P = 0, F = 1, K = 2\}, \{P \geq 1, F \geq P, 3 \leq K \leq 6\}\ \}$
PSet	dsc/2	$\{\ \{P = 0, F = 1, K = 2\}, \{P = 1, F = 1, K = 3\},$ $\{P + 1 \leq F \leq 2P, 4 \leq K \leq 6\}\ \}$
PSet	dsc/3	$\{\ \{P = 0, F = 1, K = 2\}, \{P = 1, F = 1, K = 3\},$ $\{P = 1, F = 2, K = 4\}, \{3P \leq 2F, F \leq 2P - 1, 5 \leq K \leq 6\}\ \}$
PSet	dsc/4	$\{\ \{P = 0, F = 1, K = 2\}, \{P = 1, F = 1, K = 3\},$ $\{P = 1, F = 2, K = 4\}, \{P = 2, F = 3, K = 5\},$ $\{3P + 1 \leq 2F, 3F \leq 5P, K = 6\}\ \}$
PSet	dsc/5	$\{\ \{P = 0, F = 1, K = 2\}, \{P = 1, F = 1, K = 3\},$ $\{P = 1, F = 2, K = 4\}, \{P = 2, F = 3, K = 5\},$ $\{P = 3, F = 5, K = 6\}\ \}$
PSet	dsc/6	same value (detected fixpoint in PSet)

Table 3: Abstract values computed using Box and Box$\Uparrow\!\Downarrow$PSet.

5 Related Work

Widening operators are quite often necessary to enforce the stabilization of the ascending iteration sequence. Sometimes they are used even in abstract domains having no infinite ascending chains, to accelerate convergence, rather than enforcing it. In restricted cases, the use of widenings can be avoided even though the domain has infinite ascending chains: sometimes it is possible to apply fixpoint acceleration techniques [25] or strategy/policy iteration [23, 24] so as to compute the *exact* abstract fixpoint.

When widening operators are actually used, they also are one of the main sources of imprecision for the static analysis. As a consequence, many techniques try to mitigate the corresponding precision loss: [31] proposes the *widening up-to* technique, which tries to preserve precision by using a fixed set of constraints, used as *widening hints*; a similar approach (*widening with thresholds*) is used in [11]; in [5] a framework is proposed to improve the precision of any given widening operator using several heuristics, while still guaranteeing termination; other generic techniques include widening with landmarks [38], lookahead widening [26], guided static analysis [27], and stratified widening [36]. Note that all of the approaches above focus on the ascending sequence and hence are in principle orthogonal with respect to (i.e., they can be combined with) our proposal.

The computation of the descending sequence with narrowing is just another technique (as a matter of fact, the very first one proposed in the literature) to mitigate the imprecision of widenings. However, narrowings have received fewer attention,[6] and it is often believed that the descending sequence can hardly improve precision after a few iterations. Such a belief is probably justified when considering abstract domains whose elements are expressible as template polyhedra. In particular, for the case of template polyhedra with integral bounds (including integral boxes and octagons), [1] first shows that the abstract join operation can be safely replaced by its *left strict* variant; then they prove that, when using this join operator, the computed descending sequence cannot be infinite. However, as witnessed by the `fibcall.c` example shown in Figure 6, when adopting more precise domains the descending sequence can improve the precision of the analysis well beyond the first few iterations. This seems to be the case, in particular, for domains such as the finite powerset of polyhedra.

[2,3] propose a technique to intertwine the computation of widenings and narrowing (i.e., the computation of ascending and descending chains) during the analysis, aiming at improving the precision of the post-fixpoint computed when the CFG has nested loops. [13,29] propose a technique to improve the precision of the analysis by restarting, possibly several times, the abstract (ascending and descending) iteration sequence from a perturbation of the computed post-fixpoint. In the proposals recalled above the abstract domain is fixed during the runs of the analysis, i.e., the same domain is used in the ascending and descending iteration phases; hence, once again, these approaches are orthogonal with respect to the proposal of this paper. We plan to better investigate the potential synergies arising by integrating the intertwining of widening and narrowing of [3] (implemented for instance in IKOS [14] and SeaHorn [28]) with our decoupling of the ascending and descending phases: in practice, for the combination $\mathbb{A}{\uparrow}\!{\downarrow}\mathbb{D}$, besides using the concretization function $\gamma \colon \mathbb{A} \to \mathbb{D}$ to transfer the ascending post-fixpoint to domain $\mathbb{D}$ (as in the current proposal), we will also be using the abstraction function $\alpha \colon \mathbb{D} \to \mathbb{A}$ to transfer back the descending post-fixpoint whenever restarting the ascending phase on $\mathbb{A}$.

As mentioned previously, a formal justification for the correctness of our proposal is easily obtained by casting it as a meta-abstract interpretation (the so-called $\mathrm{A}^2\mathrm{I}$ framework [20]). The pre-analysis of the CFG proposed in [12] to reduce the number widening points can be interpreted as a very early instance of the offline $\mathrm{A}^2\mathrm{I}$ approach. More recently, [34] propose an offline pre-analysis to tailor the configuration of the static analysis tool to the specific program being analyzed. Online (i.e., dynamically computed) meta-analyses include, for instance, variable partitioning techniques [30,39] and the optimized implementation of semantic operators using *boxed polyhedra* [9]. While there certainly are static analysis tools that perform a *non-uniform* analysis (i.e., they use different abstract domains for different portions of the program being analyzed), to the best of our knowledge our approach is the first example of an analysis where the whole abstract domain (and not just one of its operators) is changed *during* the analysis of a single portion of code.

[6] Probably, this is due to the fact that the abstract domain glb operator implements a correct narrowing as soon as we can enforce a finite number of applications.

6 Conclusion

In this paper we have proposed a novel yet simple variation of the typical approach used in static analysis by abstract interpretation, where we decouple the ascending and descending phases of the abstract semantics computation. We use an abstract domain combination denoted $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$, meaning that the ascending phase uses an (ascending) abstract domain $\mathbb{A}$, while the descending phase uses a strictly more precise (descending) abstract domain $\mathbb{D}$. We have implemented our approach by extending the static analysis tool PAGAI and studied its effectiveness on several, different choices for $\mathbb{A}$ and $\mathbb{D}$ of classical numerical abstract domains, including boxes, octagons, convex polyhedra and sets of polyhedra. Our preliminary experimental results show that decoupling the ascending and descending phases in $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$ allows to obtain significant precision improvements when compared with a classical static analysis computing on $\mathbb{A}$. In particular, the choice of a disjunctive domain for the descending phase seems promising.

Even though this preliminary experimental evaluation is not adequate for assessing the impact on efficiency (in particular, scalability) of the proposed approach, we conjecture that the idea of using a more precise domain $\mathbb{D}$ only in the descending phase naturally leads to a more easily tunable efficiency/precision tradeoff. We would also like to stress that our approach is not really meant to be used *uniformly* on all the code being analyzed; rather, the idea is to selectively enable it on those portions of the program where a precision gain would be desirable, but scalability issues likely prevent to perform the whole analysis using the more precise (and usually less efficient) domain $\mathbb{D}$. As a consequence, an interesting problem that will be studied in future work is how to automatically identify those parts of the program where the decoupled approach is going to be more helpful. In particular, we plan to investigate the effectiveness of simple heuristics (e.g., suitable metrics on the CFG of a function) as well as more sophisticated approaches possibly based on machine learning techniques. Going even further, we could not only select *where* to enable the more concrete descending domain $\mathbb{D}$, but also *drive* the choice of the descending domain $\mathbb{D}$. In particular, we can observe that precision of static analysis is an intensional property, namely it depends on the way the program is written [15,16]. This implies that, we can drive the choice of the descending domain depending on the syntactic characteristics of expressions (guards and assignments) that, in the program, we effectively aim to analyze, since precisely these expressions are the program elements determining the precision of the analyzer [16].

By studying the results of the experimental evaluation, one can also observe that, in a high percentage of cases, the analysis with $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$ is able to produce the same analysis results of the more precise domain $\mathbb{D}$ (e.g., $\mathsf{Box}{\uparrow\!\downarrow}\mathsf{Oct}$ obtains the same results of Oct in 73% of the widening points for the considered benchmarks). This suggests an alternative usage of the decoupled approach, starting from rather different motivations: instead of improving the precision of a classical analysis on $\mathbb{A}$ using the more precise combination $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$ (as discussed above), one may try to improve the efficiency of a classical analysis on $\mathbb{D}$ by adopting the less precise combination $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$. In such a context, one would be interested in identifying those portions of the program where the decoupled approach is anyway as precise as the classical approach using $\mathbb{D}$; once again, from a practical point of view, this problem can be addressed using heuristics and/or machine learning techniques. The same problem can also be addressed from a more theoretical point of view, leading to the following research question: *"For a program P and an abstract domain $\mathbb{D}$, which is the less precise domain $\mathbb{A}$ such that the decoupled approach $\mathbb{A}{\uparrow\!\downarrow}\mathbb{D}$ yields the same results of $\mathbb{D}$ on P?"*

References

1. Amato, G., Di Nardo Di Maio, S., Meo, M., Scozzari, F.: Descending chains and narrowing on template abstract domains. Acta Informatica **55**(6), 521–545 (2018). https://doi.org/10.1007/s00236-016-0291-0
2. Amato, G., Scozzari, F.: Localizing widening and narrowing. In: Logozzo, F., Fähndrich, M. (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 (2013). https://doi.org/10.1007/978-3-642-38856-9_4
3. Amato, G., Scozzari, F., Seidl, H., Apinis, K., Vojdani, V.: Efficiently intertwining widening and narrowing. Sci. Comput. Program. **120**, 1–24 (2016). https://doi.org/10.1016/j.scico.2015.12.005
4. Arceri, V., Mastroeni, I., Xu, S.: Static analysis for ecmascript string manipulation programs. Appl. Sci. **10**, 3525 (2020). https://doi.org/10.3390/app10103525
5. Bagnara, R., Hill, P., Ricci, E., Zaffanella, E.: Precise widening operators for convex polyhedra. Sci. Comput. Program. **58**(1-2), 28–56 (2005). https://doi.org/10.1016/j.scico.2005.02.003
6. Bagnara, R., Hill, P., Zaffanella, E.: Widening operators for powerset domains. Int. J. Softw. Tools Technol. Transf. **8**(4-5), 449–466 (2006). https://doi.org/10.1007/s10009-005-0215-8
7. Becchi, A., Zaffanella, E.: A direct encoding for NNC polyhedra. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 230–248. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_13
8. Becchi, A., Zaffanella, E.: An efficient abstract domain for not necessarily closed polyhedra. In: Podelski, A. (ed.) Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11002, pp. 146–165. Springer (2018). https://doi.org/10.1007/978-3-319-99725-4_11
9. Becchi, A., Zaffanella, E.: Revisiting polyhedral analysis for hybrid systems. In: Chang, B. (ed.) Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11822, pp. 183–202. Springer (2019). https://doi.org/10.1007/978-3-030-32304-2_10
10. Becchi, A., Zaffanella, E.: PPLite: zero-overhead encoding of NNC polyhedra. Inf. Comput. **275**, 104620 (2020). https://doi.org/10.1016/j.ic.2020.104620
11. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Cytron, R.,

Gupta, R. (eds.) Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11, 2003. pp. 196–207. ACM (2003). https://doi.org/10.1145/781131.781153

12. Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: Bjørner, D., Broy, M., Pottosin, I. (eds.) Formal Methods in Programming and Their Applications, International Conference, Akademgorodok, Novosibirsk, Russia, June 28 - July 2, 1993, Proceedings. Lecture Notes in Computer Science, vol. 735, pp. 128–141. Springer (1993). https://doi.org/10.1007/BFb0039704
13. Boutonnet, R., Halbwachs, N.: Improving the results of program analysis by abstract interpretation beyond the decreasing sequence. Formal Methods Syst. Des. **53**(3), 384–406 (2018). https://doi.org/10.1007/s10703-017-0310-y
14. Brat, G., Navas, J., Shi, N., Venet, A.: IKOS: A framework for static analysis based on abstract interpretation. In: Giannakopoulou, D., Salaün, G. (eds.) Software Engineering and Formal Methods - 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8702, pp. 271–277. Springer (2014). https://doi.org/10.1007/978-3-319-10431-7_20
15. Bruni, R., Giacobazzi, R., Gori, R., Garcia-Contreras, I., Pavlovic, D.: Abstract extensionality: on the properties of incomplete abstract interpretations. Proc. ACM Program. Lang. **4**(POPL), 28:1–28:28 (2020)
16. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F.: A logic for locally complete abstract interpretations. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021. pp. 1–13. IEEE (2021)
17. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977. pp. 238–252 (1977)
18. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979. pp. 269–282 (1979)
19. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Log. Comput. **2**(4), 511–547 (1992). https://doi.org/10.1093/logcom/2.4.511
20. Cousot, P., Giacobazzi, R., Ranzato, F.: A^2I: abstract^2 interpretation. Proc. ACM Program. Lang. **3**(POPL), 42:1–42:31 (2019). https://doi.org/10.1145/3290355
21. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Aho, A., Zilles, S., Szymanski, T. (eds.) Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978. pp. 84–96. ACM Press (1978). https://doi.org/10.1145/512760.512770
22. Cousot, P., Cousot, R.: Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In: Bruynooghe, M., Wirsing, M. (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 (1992). https://doi.org/10.1007/3-540-55844-6_142
23. Gaubert, S., Goubault, E., Taly, A., Zennou, S.: Static analysis by policy iteration on relational domains. In: Nicola, R.D. (ed.) Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007,

Braga, Portugal, March 24 - April 1, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4421, pp. 237–252. Springer (2007). https://doi.org/10.1007/978-3-540-71316-6_17

24. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: Nicola, R.D. (ed.) Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4421, pp. 300–315. Springer (2007). https://doi.org/10.1007/978-3-540-71316-6_21
25. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Yi, K. (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 (2006). https://doi.org/10.1007/11823230_10
26. Gopan, D., Reps, T.: Lookahead widening. In: Ball, T., Jones, R. (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4144, pp. 452–466. Springer (2006). https://doi.org/10.1007/11817963_41
27. Gopan, D., Reps, T.: Guided static analysis. In: Nielson, H., Filé, G. (eds.) Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4634, pp. 349–365. Springer (2007). https://doi.org/10.1007/978-3-540-74061-2_22
28. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.: The seahorn verification framework. In: Kroening, D., Pasareanu, C. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9206, pp. 343–361. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_20
29. Halbwachs, N., Henry, J.: When the decreasing sequence fails. In: Miné, A., Schmidt, D. (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 (2012). https://doi.org/10.1007/978-3-642-33125-1_15
30. Halbwachs, N., Merchat, D., Gonnord, L.: Some ways to reduce the space dimension in polyhedra computations. Formal Methods Syst. Des. **29**(1), 79–95 (2006). https://doi.org/10.1007/s10703-006-0013-2
31. Halbwachs, N., Proy, Y., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: Charlier, B.L. (ed.) Static Analysis, First International Static Analysis Symposium, SAS'94, Namur, Belgium, September 28-30, 1994, Proceedings. Lecture Notes in Computer Science, vol. 864, pp. 223–237. Springer (1994). https://doi.org/10.1007/3-540-58485-4_43
32. Henry, J., Monniaux, D., Moy, M.: PAGAI: A path sensitive static analyser. Electron. Notes Theor. Comput. Sci. **289**, 15–25 (2012). https://doi.org/10.1016/j.entcs.2012.11.003
33. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 661–667. Springer (2009). https://doi.org/10.1007/978-3-642-02658-4_52
34. Mansur, M., Mariano, B., Christakis, M., Navas, J., Wüstholz, V.: Automatically tailoring abstract interpretation to custom usage scenarios. In: Silva, A., Leino, K.

(eds.) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12760, pp. 777–800. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_36

35. Miné, A.: The octagon abstract domain. High. Order Symb. Comput. **19**(1), 31–100 (2006). https://doi.org/10.1007/s10990-006-8609-1
36. Monniaux, D., Guen, J.L.: Stratified static analysis based on variable dependencies. Electron. Notes Theor. Comput. Sci. **288**, 61–74 (2012). https://doi.org/10.1016/j.entcs.2012.10.008
37. Nielson, F., Nielson, H., Hankin, C.: Principles of program analysis. Springer (1999). https://doi.org/10.1007/978-3-662-03811-6
38. Simon, A., King, A.: Widening polyhedra with landmarks. In: Kobayashi, N. (ed.) Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Sydney, Australia, November 8-10, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4279, pp. 166–182. Springer (2006). https://doi.org/10.1007/11924661_11
39. Singh, G., Püschel, M., Vechev, M.: Fast polyhedra abstract domain. In: Castagna, G., Gordon, A. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 46–59. ACM (2017). https://doi.org/10.1145/3009837.3009885