Dissecting Widening: Separating Termination from Information

Graeme Gange[1], Jorge A. Navas[2], Peter Schachte[3], Harald Søndergaard[3], and Peter J. Stuckey[1]

[1] Faculty of Information Technology, Monash University, Melbourne, Australia
[2] SRI International, California, USA
[3] Computing and Information Systems, The University of Melbourne, Australia

Abstract. Widening ensures or accelerates convergence of a program analysis, and sometimes contributes a guarantee of soundness that would otherwise be absent. In this paper we propose a generalised view of widening, in which widening operates on values that are not necessarily elements of the given abstract domain, although they must be in a correspondence, the details of which we spell out. We show that the new view generalizes the traditional view, and that at least three distinct advantages flow from the generalization. First, it gives a handle on "compositional safety", the problem of creating widening operators for product domains. Second, it adds a degree of flexibility, allowing us to define variants of widening, such as delayed widening, without resorting to intrusive surgery on an underlying fixpoint engine. Third, it adds a degree of robustness, by making it difficult for an analysis implementor to make certain subtle (syntactic vs semantic) category mistakes. The paper supports these claims with examples. Our proposal has been implemented in a state-of-the-art abstract interpreter, and we briefly report on the changes that the revised view necessitated.

1 Introduction

A central problem in abstract interpretation is fixpoint finding: designing methods to find fixpoints of functions defined over certain mathematical structures, usually lattices, ideally producing results that are optimal in some sense. Here we shall assume that we are concerned with finding *least* fixpoints, or if that turns out to be difficult, fixpoints that are as small as we can manage.

Least fixpoints are usually found with Kleene's method, through repeated iteration starting from a smallest domain element. However, this iteration may not terminate, or may converge too slowly to be practical. *Widening* operators [8] serve a critical role in this, enforcing termination of Kleene iteration by jumping over infinite ascending chains, or simply accelerating analysis by somehow skipping long chain segments. Since widening may incur a loss of precision, the introduction of widening into an abstract interpretation engine becomes an art. The details of when and how to apply widening touch upon delicate trade-offs between the speed and the precision of program analysis.

A classical example is the widening operator often used with interval analysis [8,11]. This analysis determines for each program variable x at each program point, which values x could possibly take, in the form of an interval $[\texttt{lo}, \texttt{hi}]$.

For the program in Figure 1, an interval analysis can determine that, after the loop body has been executed 3 times, x is in the interval [3,6] and y is in [0,3]. Naive interval analysis, however, may not terminate, as it does not track the correspondence between x and y: after 100 iterations, it will see further (spurious) iterates $\{x \in [0, 100], y \in [0, 101]\}$, etc. Note that, as is common, interval analysis over-approximates the set of runtime states. In general, even if the least fixpoint is finitely reachable, it may take intolerably many iterations to reach.

```
x = 0; y = 0;
while(x < 100)
   if(a[x] > 0)
      y++; x++;
   else
      x += 2;
```

Fig. 1: Code snippet

Cousot and Cousot [8] introduced *widening* operations to cope with the problem of naive analysis being slow or non-terminating, and *narrowing* to improve on results (post-fixpoints) obtained after widening. Loosely, the idea behind widening is to introduce means for program analysis to skip long, possibly infinite, chains. For the interval analysis above, we might recognise that y's lower bound seems to remain unchanged in successive iterations, whereas its upper bound changes. Based on this we might move straight to the interval $[0, \infty]$ for y. Alternatively we might look for suggestions for less radical widening, provided by the surrounding program text, for example in the form of constants used in loop conditions. (We later discuss some of the variants of widening that have been proposed.) Narrowing may improve of the result of widening, although it is no panacea [26]. In this paper we are concerned exclusively with widening.

We propose a definition of widening which generalises the original concept in a small but critical way. We do not suggest that there is anything wrong with the original definition(s), and indeed a large number of useful and practical analysis tools have been built based on the standard view. However, *isolating* the termination aspect of widening from the task of finding upper bounds in an abstract domain has advantages, as we hope to show. By not conflating the two aspects, our definition of "isolated" widening covers some common constructions which are not true widenings in the classical sense. At the same time it simplifies certain implementation tasks, enabling compositional design of widening operators, and it eliminates certain pitfalls that surround the implementor.

In Section 2 we recapitulate the classical definition, and in Section 3 we demonstrate the pitfalls alluded to above. Section 4 defines the notion of *isolated* widening, and in Section 5 we demonstrate how isolated widenings resolve some common difficulties. Section 6 reports on our experience with the effort required to incorporate isolated widening in a generic abstract interpretation framework. Section 7 discusses related work and puts our proposal in the context of various forms of widening previously suggested, and Section 8 concludes. For proofs of the stated results, see Gange et al. [18].

2 Kleene iteration with widening

At an appropriate level of abstraction, a static analysis problem can be expressed as the search for solutions to an equation system

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} F_1(x_1, x_2, \ldots, x_n) \\ F_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ F_n(x_1, x_2, \ldots, x_n) \end{pmatrix} \tag{1}$$

This assumes a set $\texttt{Loc} = \{1, \ldots, n\}$ of program *locations* of interest, with $x_i \in \mathcal{C}$ representing (or approximating) the set of program states that may be observed at location i. F_i is the *transfer function* for location i, specifying how information pertaining to that location is computed from the information available at locations feeding into i. The *concrete domain* is assumed to be a partially ordered set $(\mathcal{C}, \subseteq)$, and each $F_i : \mathcal{C}^n \to \mathcal{C}$ is assumed to be monotone, with $\mathcal{C}^n$ ordered component-wise.

We say that x_i *depends* on x_j iff the definition of F_i mentions x_j. The *dependency graph* for (1) is the directed graph with $\texttt{Loc}$ as its set of nodes and an edge from i to j iff x_i depends on x_j.

Working directly with $\mathcal{C}$ is typically impractical, so analysis is performed on some alternate *abstract domain* $(\mathcal{D}, \sqsubseteq)$. $\mathcal{D}$ is related to $\mathcal{C}$ by a monotone *concretisation function* γ.[4] We say abstract state y *abstracts* concrete state x iff $x \subseteq \gamma(y)$. Similarly, we say abstract transfer function $F^\sharp$ *abstracts* transfer function F iff for all $y_1, \ldots, y_n$:

$$F(\gamma(y_1), \ldots, \gamma(y_n)) \subseteq \gamma(F^\sharp(y_1, \ldots, y_n)) \tag{2}$$

This ensures that, though $F^\sharp$ may not itself be monotone, it is an upper bound of the image (under γ) of the monotone F. So long as (2) holds, any (post-)fixpoint of $F^\sharp$ is a sound approximation of the sequence $[F(\bot), F^2(\bot), \ldots]$.

Circumstances under which (1) or its abstraction have a solution are well known. For example, $\mathcal{D}$ may be a complete lattice. Even so, simple iteration techniques such as Kleene iteration may fail to find a solution in finite time, if $\mathcal{D}$ has infinite ascending chains. Or, they may just be too slow to be practical, in the context of long ascending chains, even if these are finite. To solve this problem, Cousot and Cousot suggested the use of a *widening* operator.

Definition 1 (Widening [11]). A widening over domain $(\mathcal{D}, \sqsubseteq)$ is a binary operator $\triangledown : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ such that

- $\forall x, y \in \mathcal{D} : x \sqsubseteq x \triangledown y$
- $\forall x, y \in \mathcal{D} : y \sqsubseteq x \triangledown y$
- For all increasing chains $x_0 \sqsubseteq x_1, \ldots$, the increasing chain defined by $y_0 = x_0, \ldots, y_{i+1} = y_i \triangledown x_{i+1}, \ldots$ is not strictly increasing.

[4] Many variants of the concrete/abstract correspondence exist [10]. Here we deliberately adopt a relaxed formalisation which imposes few requirements on the domain.

Some later formulations of ∇ (e.g., [7]) impose a stricter condition:

$$\begin{array}{l}\text{For all } (a_i)\text{, the sequence } (a_i^{\nabla}) \text{ defined as:} \\ a_0^{\nabla} = a_0, \quad a_{n+1}^{\nabla} = a_n^{\nabla} \nabla a_{n+1} \quad \text{is ultimately stationary}\end{array} \tag{3}$$

The idea is to choose, judiciously, a set $W \subseteq \mathtt{Loc}$ of *widening points*, so that for every cycle C in the dependency graph for (1), $W \cap C \neq \emptyset$. (Such a set W always exists, but the aim is usually to choose a smallest possible set of widening points.) For each location $i \in W$, the equation for x_i in (1) is replaced by

$$x_i = x_i \nabla F_i(x_1, x_2, \ldots, x_n) \tag{4}$$

The impact on Kleene iteration is that the sequence of iterates for location $i \in W$ becomes, using (4),

$$x_i^0 = \bot, \quad x_i^{k+1} = x_i^k \nabla F_i(x_1^k, x_2^k, \ldots, x_n^k) \tag{5}$$

rather than $x_i^0 = \bot, x_i^{k+1} = F_i(x_1^k, x_2^k, \ldots, x_n^k)$ using (1). The properties of ∇ ensure the convergence of (5). Equation (5) also clearly shows the different roles of ∇'s arguments. The left argument holds "historical" information and we shall refer to it as the *widener*. The right argument holds "current" information, which may be weakened as a result of widening; we shall refer to it as the "widenee".

The widening operator ∇ lacks a property that is shared by other domain operators: ∇ is not required to be monotone. Moreover, unlike other upper bound operators used in the context of abstract interpretation, ∇ is not normally commutative, nor is it intended to be commutative. This is because its role in the system of recurrent equations is very different to other operators: Widening points are the only locations for which x_i is defined in terms of *its own* past values, in the history of iterations. At all other locations, x_i is defined in terms of the values obtained for neighbouring locations.

In spite of these anomalies, the classical formulation of ∇ leads to a sound analysis framework. There tends, however, to be a distinct mismatch between the formulation and the way widenings are constructed and used in practice. This mismatch manifests in a number of ways, requiring awkward choices in analysis engines. As we shall see, it occasionally causes unexpected non-termination.

Cousot and Cousot [11] demonstrate ∇'s lack of monotonicity in the context of interval analysis. Consider the complete lattice of intervals $(\mathcal{I}, \sqsubseteq)$ with

$$\mathcal{I} = \{\bot\} \cup \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{\infty\}, \ell \leq u\}$$

The ordering $\sqsubseteq$ is defined by $z \sqsubseteq z'$ iff $\mathsf{lo}(z') \leq \mathsf{lo}(z) \wedge \mathsf{hi}(z) \leq \mathsf{hi}(z')$, where

$$\mathsf{lo}(z) = \begin{cases} \infty & \text{if } z = \bot \\ x & \text{if } z = [x, y] \end{cases} \qquad \mathsf{hi}(z) = \begin{cases} -\infty & \text{if } z = \bot \\ y & \text{if } z = [x, y] \end{cases}$$

(see Figure 2, ignoring the dashed lines for now).

For this domain, a natural widening operation is $\nabla_{\mathcal{I}}$ defined as follows:

$$\begin{array}{r}\bot \nabla_{\mathcal{I}} Y = Y \\ X \nabla_{\mathcal{I}} \bot = X \\ [x_\ell, x_u] \nabla_{\mathcal{I}} [y_\ell, y_u] = [\text{if } y_\ell < x_\ell \text{ then } -\infty \text{ else } x_\ell, \text{if } x_u < y_u \text{ then } \infty \text{ else } x_u]\end{array}$$

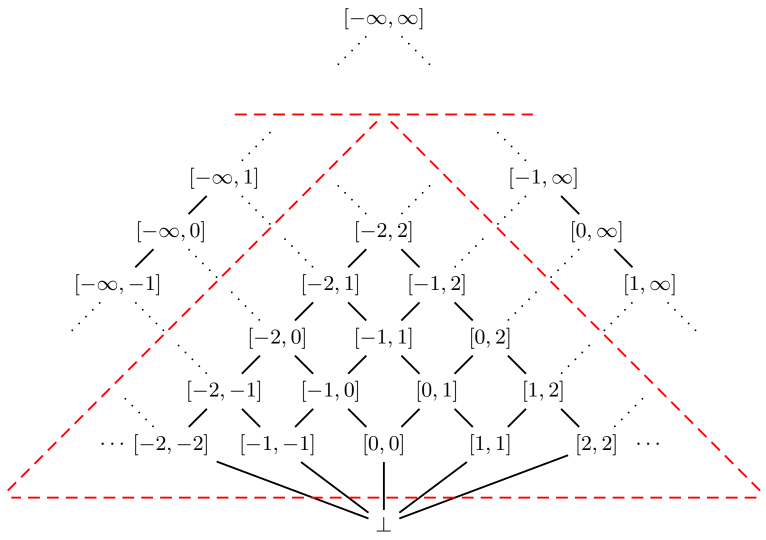

Fig. 2: The integer interval domain as a Hasse diagram. The role of the red dashed lines will be made clear in Section 4.

That is, unstable bounds get extrapolated, lower bounds to $-\infty$ and upper bounds to ∞. To show that ∇ fails to be monotone in the widener, Cousot and Cousot [11] note that $[0,1] \sqsubseteq [0,2]$ but $[0,1] \nabla_{\mathcal{I}} [0,2] = [0,\infty]$ whereas $[0,2] \nabla_{\mathcal{I}} [0,2] = [0,2]$. While this particular ∇ happens to be monotone in its widenee, the definition of widening does not enforce such monotonicity.

To see that ∇ may lack monotonicity in either argument position, consider the complete lattice $(\mathbb{N} \cup \{\infty\}, \sqsubseteq)$, with $\sqsubseteq$ defined $x \sqsubseteq y$ iff $x \leq y \vee y = \infty$ (with $\leq$ being the usual ordering on $\mathbb{N}$). Define widening on this lattice as follows:

$$x \nabla y = \begin{cases} \max(x,y) & \text{if } y = 2 \\ \infty & \text{otherwise} \end{cases}$$

Here $1 \sqsubseteq 2$ but $\infty = 0 \nabla 1 \not\sqsubseteq 0 \nabla 2 = 2$, so ∇ is not monotone in the widenee. Yet ∇ is an upper bound operator, and for every increasing chain $x_0, x_1, \ldots$, the increasing chain $y_0 = x_0, y_{i+1} = y_i \nabla x_{i+1}$ stabilises, so the classical requirements for a widening operation are satisfied.

It is worth highlighting the impact of (3). Definition 1 only guarantees convergence if the sequence of widenees is increasing. If the abstract transformer $F^\sharp$ relating successive iterations is monotone, this property is ensured. However, the monotonicity of $F^\sharp$ can be easily lost in a number of ways:

– If the abstraction $\mathcal{D}$ is not a join semi-lattice (e.g. [16, 17, 21]), there is no least-upper bound, thus successive values at control-flow join points may not strictly increase [15].
– If $F^\sharp$ is not the best abstraction of F, but some relaxation (as is common for non-linear operations), the sequence of iterates again may be non-monotone.

– A special case of the above is use of reduction operations to propagate information between multiple domains (discussed in Section 3.1). If the operators for channeling information between domains are not idempotent, and are not iterated to a fixpoint, monotonicity is again lost.

In any of these situations, a widening that only satisfies Definition 1 makes no guarantees of termination. Fortunately, the stabilisation-based widenings commonly used for numeric domains all satisfy (3), and for other cases, alternative, stricter, definitions of widening have been proposed [18](appendix 2).

3 Problems and pitfalls

To motivate a fresh look at widening, we first discuss some irregular properties of widening: an absence of compositionality, a lack of flexibility, and a certain lack of robustness. In Section 5 we return to these aspects, to show how a different view of widening can remove or mitigate some drawbacks.

3.1 Problems of compositionality

Abstract interpretation makes use of a number of domain product constructions. The reduced product of abstract domains [5,9] is a powerful concept, but difficult to implement in practice. Granger products [22] are a frequent compromise, equipping a pair of domains with 'reduction' operators to propagate information between them. In essence, both approaches take the quotient of $D_1 \times D_2$ under some equivalence relation $\equiv$. However, it is not in general safe to use widenings for D_1 and D_2 directly as a widening for $(D_1 \times D_2)_{/\equiv}$.

Example 1. Consider $D_1 = D_2 = \mathbb{N} \cup \{\infty\}$, with the usual ordering $\leq$, and define:

$$w \mathbin{\nabla_e} x = \begin{cases} w & \textbf{if } x \leq w \\ x & \textbf{if } x > w, w \text{ is even, and } x \text{ is odd} \\ x+1 & \textbf{if } x > w, w \text{ is even, and } x \text{ is even} \\ \infty & \textbf{otherwise} \end{cases}$$

$$w \mathbin{\nabla_o} x = \begin{cases} w & \textbf{if } x \leq w \\ x & \textbf{if } x > w, w \text{ is odd, and } x \text{ is even} \\ x+1 & \textbf{if } x > w, w \text{ is odd, and } x \text{ is odd} \\ \infty & \textbf{otherwise} \end{cases}$$

Note that ∇_e always converges: if the first value is even, the second iterate will become odd, after which the third increasing step will jump to ∞. ∇_o converges by analogous reasoning.

In the reduced product $D_1 \times D_2$, the meaning of (x, y) is simply $\min(x, y)$. Consider what happens to the strictly increasing sequence $0, 1, 2, \ldots$

Sequence to stabilise:	0	1	2	3	4	$\ldots$
Result of ∇_π at iteration i:	$(0,0)$	$(1,\infty)$	$(\infty,2)$	$(3,\infty)$	$(\infty,4)$	$\ldots$
Reduced element at iteration i:	0	1	2	3	4	$\ldots$

When we map the two individual components back onto the quotient class, we regain information that was discarded by the previous widening. As a result, stabilisation is lost. □

Example 1 shows that traditional widening does not guarantee compositional stabilisation. Each of the widening operators in the example provides stabilisation, one in D_1, the other in D_2, and yet their natural composition does not provide stabilisation in the reduced product of D_1 and D_2. Each has the effect of undermining the other.

The lack of compositionality manifests itself in other ways.

Example 2. Widening with thresholds [25,26] is a common strategy for avoiding precision loss in widening. However, implementing widening with thresholds for a large number of numeric abstract domains is tedious, and it would be preferable to utilise an existing widening operator. Indeed, in Crab [14], a generic version of widening with thresholds was previously implemented as follows:

$$\begin{aligned} &s \mathbin{\nabla_{\mathcal{A}}^{T}} t = \\ &\quad (s \mathbin{\nabla_{\mathcal{A}}} t) \sqcap_{\mathcal{A}} (\textsc{from-interval}(\textsc{to-interval}(s) \mathbin{\nabla_{\mathcal{I}}^{T}} \textsc{to-interval}(t))) \end{aligned} \tag{6}$$

That is, given an arbitrary numerical abstract domain $\mathcal{A}$, extract interval approximations from $s, t \in \mathcal{A}$, using the function TO-INTERVAL that converts from $\mathcal{A}$ to $\mathcal{I}$. Then, apply (non-threshold) widening to the $\mathcal{A}$-operands, apply interval widening with thresholds to the $\mathcal{I}$-operands, convert from $\mathcal{I}$ to $\mathcal{A}$ using the function FROM-INTERVAL, and take the meet of the two results.

This sequence of operations *ought* to be innocuous. With the assumption that widening is applied in an unbroken sequence, and non-widening steps are not allowed after widening, the suggested solution should be safe: $\nabla_{\mathcal{I}}^{T}$ can be safely interleaved with increasing functions, so the interval component will eventually converge. And since the resulting intervals are stable, adding them back into $s \mathbin{\nabla_{\mathcal{A}}} t$ should have no effect. Nevertheless, this widening strategy would—on very rare occasions—cause non-termination.[5] □

It turns out that the problem in Example 2 is due to the call TO-INTERVAL(s), where we find the interval approximation of a previous iterate. To compute the tightest interval approximation of s, TO-INTERVAL must normalise s, that is, explicate the transitive closure of its representation. Not knowing that s is 'really' a widener, TO-INTERVAL helpfully modifies s in place, inadvertently breaking the termination conditions. There is nothing wrong with (6), read declaratively, and it is hard to blame the implementation of TO-INTERVAL. The root cause is that the termination invariant required by widening relies on invisible properties which are not captured at the semantic (or API) level. What was intended as a syntactic object (a set of constraints) is treated as a semantic object (a set of models). Clearly it would be nice to have a "widening API" that prevents this kind of category confusion.

[5] The bug was fixed on July 1st, 2018 in commit `https://github.com/seahorn/crab/commit/72ed05690bc2bbee19141f5513cb6a8e8ab3ce9a`.

3.2 Flexibility: Handling variants of widening

Widenings frequently achieve convergence by retaining only 'stable' information, and discarding everything else. This is excellent for attaining fast termination, but risks throwing away the properties we are attempting to infer.

The threshold widening discussed above is one of many different approaches to avoid excessive information loss in widening. Another common strategy to retain some information is *delayed widening* [3]: perform a bounded number of initial steps using join ($\sqcup$)—in the hope of obtaining stable invariants—before eventually resorting to widening.

These are simple and effective strategies, but implementing variants of widening is surprisingly messy [12]. For example, for delayed widening we need to track how many times each widening point has been processed, but where does this count belong? A common strategy is to remove this decision from the domain entirely, relying on the analysis engine to decide when to switch from "join" mode to "widen" mode. Alternatively, one can place an iteration counter somewhere with shared visibility. In any case, the traditional solutions involve disruptive changes to an underlying fixpoint engine, or a baroque redesign of abstract domains. A cleaner and less intrusive solution would be desirable.

3.3 Problems of fragility: Termination

Termination problems similar to that of Example 2 are easily provoked. Miné [28, 29] observed a related problem with his weakly relational domains: The implementation of these domains rely on transitive closure operations to make implicit binary relations explicit at certain points. For example, a set $\{x \leq y, y \leq z\}$ of constraints may be normalised as $\{x \leq y, y \leq z, x \leq z\}$. However, applying transitive closure to the post-widening state can result in non-termination, *even though the set of models is unchanged*. The root cause of the trouble [28, 29] (and in Example 2) is that the domain conflates two views of an abstract state: the lattice operations and abstract transformers see states as *semantic* objects, so two states with identical models are equivalent; closure is, viewed from that angle, a no-op. Widening, however, is non-semantic in its left argument: Once some constraint is discarded as unstable, it must not re-appear in future iterations. If transitive closure is applied between widening steps, we may re-infer relaxed forms of the discarded invariants, which may appear stable in the next iteration, breaking the invariant we need for termination.

Example 3. Miné [28] first identified the problem and gave a concrete example. Consider the sequence of iterates $p_i = \{|y-x| \leq i+1, |z-x| \leq i+1, |z-y| \leq 1\}$ for $i = 0, 1, 2, \ldots$. With the standard representation of difference constraints as directed graphs, we can depict the sequence as in the bottom row of Figure 3. To maintain precision, it is important to apply a transitive closure operation to constraint sets, to detect implied constraints (we say we "normalise" constraints). Now starting a sequence $\omega_0, \omega_1, \ldots$ from $\omega_0 = \{|y-x| \leq 1, |z-y| \leq 1\}$ is fine, except we run into trouble if we normalise the results of widening. Normalising

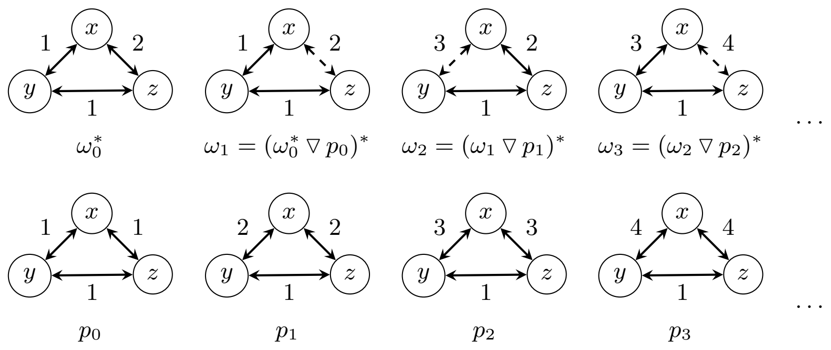
x
y
z
1
2
1
ω_0^*
$\omega_1 = (\omega_0^* \triangledown p_0)^*$
$\omega_2 = (\omega_1 \triangledown p_1)^*$
$\omega_3 = (\omega_2 \triangledown p_2)^*$
p_0
p_1
p_2
p_3
...

Fig. 3: An example of a divergent sequence of difference constraints, adapted from [28]. At each step, the edges between x and either y or z are discarded, but closure then recreates weaker relations (shown dashed), which are stable during the next iteration.

ω_0 yields $\omega_0^* = \{|y - x| \leq 1, |z - y| \leq 1, |z - y| \leq 2\}$. Under normalisation, the remainder of the widening sequence is shown in the top row of Figure 3. For example, ω_2 comes about as the closure of $\omega_1 \triangledown p_1 = \{|z - y| \leq 1, |z - x| \leq 2\}$, with closure adding the constraint $|y - x| \leq 3$. As can be seen, the widening sequence will not stabilise. The problem arises because closure—which makes no *semantic* difference—restores some constraints discarded by widening, breaking the very property that termination relied on. □

In Section 5 we return to the challenges discussed in this section. But first we introduce a different view of widening.

4 Isolated widening

In the classical treatment of abstract interpretation, the widening iterates inhabit the same type as our abstract domain. But as observed above, this view can mislead the implementor of an abstract domain. For the implementor it is critically important to isolate the effects of a *semantic* widenee from the *non-semantic* widener. As we show in Section 5.1, this isolation also aids compositionality.

To make the two aspects of widening transparent, we let the different roles of the widener and widenee be reflected in $\triangledown$'s argument types. That is, we relax the requirement that both arguments are of type $\mathcal{D}$ for some abstract domain $\mathcal{D}$. Instead, we take the widener to be an inhabitant of some partially ordered set $(\mathfrak{W}, \preceq)$ (not necessarily a lattice) which has the "ascending chain" property.

Definition 2 (Acc-poset). A partially ordered set $(S, \leq)$ satisfies the ascending chain condition iff, for every sequence $s_0 \leq s_1 \leq s_2 \leq \ldots$ of S, there is some $k \in \mathbb{N}$ such that $s_k = s_{k+1} = s_{k+2} = \ldots$. We refer to such a set as an *acc-poset.*

Example 4. Consider again the interval domain $(\mathcal{I}, \sqsubseteq)$ from Section 2. The full and dotted lines in Figure 2 show the lattice as a Hasse diagram. The natural widening operation $\nabla_{\mathcal{I}}$ defined in Section 2 is not monotone. However, consider an alternative ordering $\preceq$, defined by: $\bot \preceq d \preceq d$ for all $d \in \mathcal{I}$, and

$$[a, b] \preceq [c, d] \quad \textbf{iff} \quad (c = -\infty \wedge d \in \{b, \infty\}) \vee (d = \infty \wedge c \in \{-\infty, a\})$$

This results in an order illustrated in Figure 2 by considering edges that transitively cross a red dashed line. Nodes within one red bordered region are pairwise incomparable. Clearly $\preceq$ is a coarsening of $\sqsubseteq$: we have $i_1 \preceq i_2 \Rightarrow i_1 \sqsubseteq i_2$, but $\preceq$ makes certain elements of $\mathcal{I}$ incomparable which were comparable under $\sqsubseteq$.

Viewed with respect to $\preceq$, $\nabla_{\mathcal{I}}$ satisfies an interesting property:

$$w \preceq w' \wedge x \sqsubseteq x' \Rightarrow w \nabla_{\mathcal{I}} x \preceq w' \nabla_{\mathcal{I}} x'.$$

That is, $\nabla_{\mathcal{I}}$ *is* monotone after all, but with respect to different orderings of its left and right operands. But though $\preceq$ induces a lattice, $\nabla_{\mathcal{I}}$ does not coincide with the join operation over $(\mathcal{I}, \preceq)$, even though that operation does exist. In fact, $\nabla_{\mathcal{I}}$ is not an upper bound operation at all under $\preceq$, only under $\sqsubseteq$. For example, $[-10, 10] \nabla_{\mathcal{I}} [0, 3] = [-10, 10]$, but $[0, 3] \not\preceq [-10, 10]$. □

Example 4 provides motivation for the introduction of an isolated widening domain $\mathfrak{W}$, since the key to widening is really an independent acc-poset. The set $\mathfrak{W}$ is related to a given abstract domain $\mathcal{D}$, as specified in Definition 3 below. It is equipped with three operations:

$$\mathit{reflect} : \mathcal{D} \to \mathfrak{W} \qquad \mathit{reify} : \mathfrak{W} \to \mathcal{D} \qquad \mathbb{W} : \mathfrak{W} \times \mathcal{D} \to \mathfrak{W}$$

Here, *reflect* lifts an abstract state to initialize an ascending sequence, $\mathbb{W}$ computes successive (widening) steps in our (finite) ascending chain, and *reify* maps the current iterate back onto the abstract domain, in preparation for computing the next step in the sequence.

Definition 3 (Isolated widening). Let $(\mathcal{D}, \sqsubseteq)$ be an abstraction of poset $(\mathcal{C}, \subseteq)$ given by concretisation γ. The quintuple $\langle \mathfrak{W}, \preceq, \mathit{reflect}, \mathit{reify}, \mathbb{W} \rangle$ is an isolated widening (*I-widening*) for $(\mathcal{D}, \sqsubseteq)$ iff $(\mathfrak{W}, \preceq)$ is an acc-poset and the operators $\mathit{reflect} : \mathcal{D} \to \mathfrak{W}$, $\mathit{reify} : \mathfrak{W} \to \mathcal{D}$, and $\mathbb{W} : \mathfrak{W} \times \mathcal{D} \to \mathfrak{W}$, satisfy:

$$\forall x \in \mathcal{D}.\ \gamma(x) \subseteq \gamma(\mathit{reify}(\mathit{reflect}(x))) \tag{7}$$

$$\forall w \in \mathfrak{W},\ x \in \mathcal{D}.\ w \preceq (w \mathbin{\mathbb{W}} x) \tag{8}$$

$$\forall w \in \mathfrak{W},\ x \in \mathcal{D}.\ \gamma(x) \subseteq \gamma(\mathit{reify}(w \mathbin{\mathbb{W}} x)) \tag{9}$$

Equations (7)–(9) generalise the corresponding conditions for classical widenings. Indeed, if $\mathfrak{W} = \mathcal{D}$ and $\mathit{reflect}(x) = \mathit{reify}(x) = x$ then (7) is a tautology, and (8) and (9) ensure $\mathbb{W}$ is an upper bound operator. We do not require the sets $\mathfrak{W}$ and $\mathcal{D}$ to be identical, however. And, importantly, the left argument (the *widener*) and right argument (*widenee*) are generally subject to *different orderings*.

Theorem 1. Consider abstraction $(\mathcal{D}, \sqsubseteq)$ of domain $(\mathcal{C}, \subseteq)$ given by $\gamma : \mathcal{D} \to \mathcal{C}$. Let $\langle \mathfrak{W}, \preceq, \mathit{reflect}, \mathit{reify}, \mathbb{W} \rangle$ be an I-widening for lattice $(\mathcal{D}, \sqsubseteq)$. For every $x \in \mathcal{D}$, monotone function $f : \mathcal{C} \to \mathcal{C}$ and abstraction $f^\sharp : \mathcal{D} \to \mathcal{D}$ of f, the sequence given by

$$\begin{aligned} w_0 &= \mathit{reflect}(x) \\ w_i &= w_{i-1} \mathbin{\mathbb{W}} f^\sharp(\mathit{reify}(w_{i-1})) \end{aligned}$$

stabilises after k steps for some $k \in \mathbb{N}$, and $f^n(\gamma(x)) \sqsubseteq \gamma(\mathit{reify}(w_k))$ for all $n \in \mathbb{N}$.

Theorem 2. Consider a widening (in the sense of Definition 1) $\triangledown$ on poset $(\mathcal{D}, \sqsubseteq)$ abstracting $(\mathcal{C}, \subseteq)$, which satisfies (3). Let $\mathsf{id} : \mathcal{D} \to \mathcal{D}$ be the identity function, and let $\preceq$ be the relation given by:

$$u \preceq v \text{ iff } u = v \vee \exists x.\ u \triangledown x = v,$$

with $\preceq^*$ its transitive closure. Then $\langle \mathcal{D}, \preceq^*, \mathsf{id}, \mathsf{id}, \triangledown \rangle$ is an I-widening for $(\mathcal{D}, \sqsubseteq)$.

Theorem 2 states that each classical widening [11] induces an isolated widening. The I-widening derived from an arbitrary classical widening is not necessarily monotone. Indeed, for arbitrary fixed *reflect*, *reify*, and $\preceq$, there may be no *best* widening (though if $(\mathfrak{W}, \preceq)$ has a unique greatest element $\top$, there is always *some* monotone widening, namely $w \mathbin{\mathbb{W}} x = \top$).

5 Properties of isolated widening

We now return to the problems and pitfalls identified in Section 3. The aim is to show how the separation of concerns that was proposed in Section 4 can resolve some of the classical difficulties surrounding widening: *lack of compositionality* (as seen in product widening), *rigidity* (as seen in the difficulty of extending the definition of widening to cover variants such as delayed and threshold widening), and *fragility* (as seen in well-known implementation pitfalls).

5.1 Compositionality: Domain products and widening

Example 1 exposed the dangers surrounding synthesis of widening operators for product domains. We now show that I-widening provides compositional stabilisation, also in the presence of reduction. We characterise direct, Granger, and reduced products as quotients of a direct product under a reduction operation.

Definition 4 (Product with reduction). Let $(\mathcal{C}, \subseteq)$ be a meet semi-lattice[6] with abstractions $(\mathcal{D}_1, \sqsubseteq_1)$ and $(\mathcal{D}_2, \sqsubseteq_2)$ given by γ_1 and γ_2. Let $\mathcal{D}_{1\times 2} = \mathcal{D}_1 \times \mathcal{D}_2$.

A function $\rho_i : \mathcal{D}_{1\times 2} \to \mathcal{D}_i$ is a *reduction operator* if it satisfies:

$$\rho_i(y_1, y_2) \sqsubseteq_i y_i \qquad i \in \{1, 2\} \tag{10}$$

$$\gamma_{1\times 2}(y_1, y_2) \sqsubseteq \gamma_i(\rho_i(y_1, y_2)) \qquad i \in \{1, 2\} \tag{11}$$

[6] The requirement for a meet semi-lattice is merely so $\gamma_{1\times 2}$ is expressible. With a different formalisation (taking γ as a relation $\gamma \subseteq \mathcal{C} \times \mathcal{D}$), we may take $(\mathcal{D}, \sqsubseteq)$ as a poset, and replace (11) by $(y_1, x) \in \gamma_1 \wedge (y_2, x) \in \gamma_2 \Rightarrow (\rho_i(\langle y_1, y_2 \rangle), x) \in \gamma_i$.

Equation (10) ensures ρ_i are decreasing, and (11) ensures ρ_i do not discard any sound concrete states. We use ρ to denote the pointwise application of (ρ_1, ρ_2):

$$\rho\langle y_1, y_2\rangle = \langle \rho_1\langle y_1, y_2\rangle, \rho_2\langle y_1, y_2\rangle\rangle.$$

From (10) and (11), we have that $\gamma_{1\times 2}(\rho(x)) = \gamma_{1\times 2}(x)$. A *product with reduction* is the quotient class $\mathcal{D}_{1\times 2|\rho}$.

In this view, $\rho = \mathsf{id}$ yields the direct product, and the reduced product is obtained with the tightest possible ρ. The continuum of Granger products sit between.

Theorem 3. Consider domain $(\mathcal{C}, \subseteq)$ with abstractions $(\mathcal{D}_1, \sqsubseteq_1)$, $(\mathcal{D}_2, \sqsubseteq_2)$ given by γ_1, γ_2 resp., and reduction operator ρ inducing domain $(\mathcal{D}_{1\times 2|\rho}, \sqsubseteq_{1\times 2|\rho})$.

Let $\langle \mathfrak{W}_1, \preceq_1, \mathit{reflect}_1, \mathit{reify}_1, \mathbb{W}_1\rangle$ be an I-widening for poset $(D_1, \sqsubseteq_1)$ and let $\langle \mathfrak{W}_2, \preceq_2, \mathit{reflect}_2, \mathit{reify}_2, \mathbb{W}_2\rangle$ be an I-widening for poset $(D_2, \sqsubseteq_2)$. Define

$$\begin{aligned}
\mathit{reflect}\langle x_1, x_2\rangle &= \langle \mathit{reflect}_1(x_1), \mathit{reflect}_2(x_2)\rangle\\
\mathit{reify}_\times\langle w_1, w_2\rangle &= \langle \mathit{reify}_1(w_1), \mathit{reify}_2(w_2)\rangle\\
\mathit{reify}\langle w_1, w_2\rangle &= \rho(\mathit{reify}_\times\langle w_1, w_2\rangle)\\
\langle w_1, w_2\rangle \mathbin{\mathbb{W}} \langle x_1, x_2\rangle &= \langle w_1 \mathbin{\mathbb{W}_1} x_1, w_2 \mathbin{\mathbb{W}_2} x_2\rangle
\end{aligned}$$

Then $\langle \mathfrak{W}_1 \times \mathfrak{W}_2, \preceq, \mathit{reflect}, \mathit{reify}, \mathbb{W}\rangle$ is an I-widening for domain $(\mathcal{D}_{1\times 2|\rho}, \sqsubseteq_{\sharp 1\times 2|\rho})$ with

$$\begin{aligned}
(x_1, x_2) \sqsubseteq (x_1', x_2') &\quad \text{iff} \quad x_1 \sqsubseteq_1 x_1' \wedge x_2 \sqsubseteq_2 x_2'\\
(w_1, w_2) \preceq (w_1', w_2') &\quad \text{iff} \quad w_1 \preceq_1 w_1' \wedge w_2 \preceq_2 w_2'
\end{aligned}$$

As a special case of Theorem 3, we may safely combine multiple widenings for the same domain, in the same manner. Indeed, recall our troublesome construction from Example 2. Using *reflect* and *reify*, this becomes perfectly safe: $\mathfrak{W}$ stores the stable relations and interval properties separately, and they are combined only upon calls to *reify*. The safe replacement for (6) becomes:

$$\begin{aligned}
\mathit{reflect}_{\mathcal{AI}}(s) &= \langle \mathit{reflect}_{\mathcal{A}}(s), \mathit{reflect}_{\mathcal{I}}(\text{TO-INTERVAL}(s))\rangle\\
\mathit{reify}_{\mathcal{AI}}(\langle r, i\rangle) &= \mathit{reify}_{\mathcal{A}}(r) \sqcap \mathit{reify}_{\mathcal{A}}(\text{FROM-INTERVAL}(\mathit{reify}_{\mathcal{I}}(i)))\\
\langle r, i\rangle \mathbin{\mathbb{W}_{\mathcal{AI}}} s &= \langle r \mathbin{\nabla_{\mathcal{A}}} s, i \mathbin{\nabla_{\mathcal{I}}^{T}} \text{TO-INTERVAL}(s)\rangle
\end{aligned}$$

5.2 Flexibility: Variations of widening

Another advantage of capturing termination aspects of widening via a separate domain $\mathfrak{W}$ is that it becomes possible to define variants of widening without any need for surgery to an underlying fixpoint engine. We now show how delayed widening can be implemented generically.

We simply define a widening combinator, which lifts an isolated widening $\mathfrak{W}$ to a new isolated widening $k\mathfrak{W}$ for some given k, as follows. The values in the widening domain are from a discriminated union with constructors $U : \mathfrak{W} \to k\mathfrak{W}$ and $P : (\mathbb{N} \times \mathcal{D}) \to k\mathfrak{W}$. Let $\sqcup$ be any upper bound operator on $\mathcal{D}$ (for lattices, this will be the least upper bound). The operations are defined by:

$$\begin{aligned}
\mathit{reflect}_{k\mathfrak{W}}(s) &= P(k, s) & P(0, s) \mathbin{\mathbb{W}_{k\mathfrak{W}}} s' &= U(\mathit{reflect}_{\mathfrak{W}}(s \sqcup s'))\\
\mathit{reify}_{k\mathfrak{W}}(P(l, s)) &= s & P(l, s) \mathbin{\mathbb{W}_{k\mathfrak{W}}} s' &= P(l-1, s \sqcup s'), l > 0\\
\mathit{reify}_{k\mathfrak{W}}(U(w)) &= \mathit{reify}_{\mathfrak{W}}(w) & U(w) \mathbin{\mathbb{W}_{k\mathfrak{W}}} s' &= U(w \mathbin{\mathbb{W}_{\mathfrak{W}}} s')
\end{aligned}$$

The ordering $\preceq^{-}_{k\mathfrak{W}}$ is defined as:

$$\begin{aligned}
P(l,s) \preceq^{-}_{k\mathfrak{W}} P(l',s') &\Leftrightarrow (l = l' \land s = s') \lor (l > l' \land s \sqsubseteq s') \\
P(l,s) \preceq^{-}_{k\mathfrak{W}} U(w) &\Leftrightarrow \gamma(s) \subseteq \gamma(\mathit{reify}_{\mathfrak{W}}(w)) \\
U(w) \preceq^{-}_{k\mathfrak{W}} U(w') &\Leftrightarrow w \preceq_{\mathfrak{W}} w' \\
U(w) \preceq^{-}_{k\mathfrak{W}} P(k,s) &\Leftrightarrow \mathit{false}
\end{aligned}$$

The ordering of the new domain $\preceq_{k\mathfrak{W}}$ is defined as the transitive closure of $\preceq^{-}_{k\mathfrak{W}}$.

Proposition 1. Consider domain $(\mathcal{C}, \subseteq)$ with abstraction $(\mathcal{D}, \sqsubseteq)$ equipped with upper bound operator $\sqcup$, and let $\langle \mathfrak{W}, \preceq_{\mathfrak{W}}, \mathit{reflect}_{\mathfrak{W}}, \mathit{reify}_{\mathfrak{W}}, \mathbb{V}_{\mathfrak{W}} \rangle$ be an I-widening for $(\mathcal{D}, \sqsubseteq)$. Then $\langle k\mathfrak{W}, \preceq_{k\mathfrak{W}}, \mathit{reflect}_{k\mathfrak{W}}, \mathit{reify}_{k\mathfrak{W}}, \mathbb{V}_{k\mathfrak{W}} \rangle$ is an I-widening for $(\mathcal{D}, \sqsubseteq)$.

The I-widening framework makes it easy to define delayed widening because it allows the separation of widening control from whatever abstract domain we may be using. Other variations of delayed widening are also simple to encode. For example, we can define $P(l,s) \mathbin{\mathbb{V}_{k\mathfrak{W}}} s' = P(l,s)$ when $s \sqcup s' = s$ which may lead to more accurate widening.

5.3 Convergence

Recall the problem of non-convergent DBMs (or octagons), outlined in Section 3.3. The underlying problem is a subtle difference in the interpretation of a *state* between the (normal) domain operations and the widening operation.

Viewed in terms of $\sqsubseteq$ and $\preceq$, it is clear where things go astray in Example 3. Computing $\omega_0^* \mathbin{\triangledown} p_0$ yields the *set of constraints* $\omega = \{|z - y| \leq 1, |y - x| \leq 1\}$. This is safe, as $\omega_0^* \preceq \omega$, and $p_0 \sqsubseteq \mathit{reify}(\omega)$. But performing transitive closure on ω yields a result $\omega_1 = \omega \cup \{|z - x| \leq 2\}$. And although ω_1^* is still an upper bound of ω_0^* and p_0 with respect to $\sqsubseteq$, $\omega_0^* \not\preceq \omega_1$—so the ascending chain of wideners is broken.

The normal lattice operators view Octagons as the *quotient class* of sets of octagon constraints under entailment: two sets of constraints are *equivalent* iff they have the same set of models. The termination argument views Octagon states as sets of constraints: at each iteration, the *number of constraints* decreases, so the iteration process terminates. These are, very subtly, different sets, which are equipped with different partial orders, let us call them $\sqsubseteq_8$ and $\preceq_8$, defined by

$$\begin{aligned}
S \sqsubseteq_8 T \quad &\text{iff} \quad \forall_{c_T \in T}.\ S \Rightarrow c_T \\
S \preceq_8 T \quad &\text{iff} \quad \forall_{c_T \in T}.\ c_T \in S
\end{aligned}$$

Viewing widening through this lens, it becomes clear what goes wrong: The normalization operator (which performs transitive closure) is semantically (that is, with respect to $\sqsubseteq_8$) a no-op—it is merely the identity function. But with respect to $\preceq_8$, closure *moves downwards*. So composing $\triangledown$ and closure is no longer necessarily an upper bound operation with respect to $\preceq_8$.

Formulated as an isolated widening, these sets are kept *distinct*. Let $\equiv_8$ be the equivalence relation induced by $\sqsubseteq_8$ – that is, $S \equiv_8 T \Leftrightarrow S \sqsubseteq_8 T \land T \sqsubseteq_8 S$, and let

$=_8$ be the equivalence relation similarly induced by $\preceq_8$. Then domain elements occupy the quotient class $O_{/\equiv_8}$, where O is the set of Octagon constraints. In this domain, transitive closure is indeed safe. But widening operands are members of $O_{/=_8}$, which is *simply not equipped* with a closure operator.

With isolated widening, we cannot make this mistake: transitive closure is defined on the domain of abstract states and simply *cannot be applied* to a widener (which acts on a different set).

6 Implementation

We have implemented isolated widening for Crab [23], a parametric framework for building abstract interpreters. Crab is written in C++ and provides a front-end for analyzing LLVM bitcode. We have used Crab to evaluate the use of isolated widening on a large number of C programs from SV-COMP 2019. Our main aim with the implementation has been to gauge the extent to which isolated widening requires a larger implementation effort than the alternatives.

We modified the Crab fixpoint iterator [1] in order to call *reflect*, *reify*, and $\mathbb{W}$. This required only three new lines of C++ code. Before a new cycle in the weak topological ordering (wto) [4] of the control-flow graph (CFG) is analyzed, the new code calls *reflect*. Then, the cycle is analyzed recursively (i.e., analyzing other nested cycles) until a fixpoint is reached. Before a new fixpoint iteration starts, the new code calls *reify*. Finally, the standard call to Cousot's widening was replaced with a call to $\mathbb{W}$. In addition, we extended each Crab abstract domain $\mathcal{D}$ to implement the trivial isolated widening $\langle \mathcal{D}, \preceq^*, \mathsf{id}, \mathsf{id}, \triangledown \rangle$. This required 10 lines of C++ code since all domains share the same implementation.

For proof of concept, we implemented (a) delayed widening and (b) the reduced product of an arbitrary numerical domain with intervals as I-widening. Recall that the reduction with intervals was motivated by a desire to exploit widening-with-thresholds from the interval domain, to implement threshold widening for a number of numerical abstract domains. Regarding (a), Crab already implemented delayed widening on top of the fixpoint iterator. We replaced that code with the isolated widening defined in Section 5.2. The effort was minimal and it did not add more lines of C++ code. For (b), we needed to add more code although the amount was still relatively small (around 130 lines of C++ code).

Crab provides standard numerical domains such as Interval, Zone, Octagon, and Polyhedra. Since numerical domains are typically insufficient to prove non-trivial properties, Crab allows combining numerical domains as reduced products. Moreover, Crab provides array domains which are implemented as functor domains whose parameters can be arbitrary abstract domains.

With this in mind, we first implemented the I-widening described in Section 5.1. The implementation is parametric on the particular numerical domain, and took some 50 lines of code. We also implemented, in 40 lines, an I-widening for the reduced product of two numerical domains and a numerical domain with a finite lattice domain (used to track boolean variables). Finally, we implemented an I-widening for the array smashing domain, another 40 lines of C++ code.

To test the correctness of the implementation, we compared it against the old implementation, using 2736 programs from the SV-COMP 2019 competition (the categories `ReachSafetyControlFlow`, `ReachSafetyLoops`, and `System_Device-DriversLinux64`). We noted that the new implementation of delayed widening did not affect precision of analysis, nor analysis time.

In conclusion, the implementation of isolated widening in an existing abstract interpreter has been almost effortless.

7 Related work

The concept of widening in abstract interpretation is almost as old as abstract interpretation itself [8]. Its essential role in both the theory and practice of program analysis was clarified by Cousot and Cousot [11]. While it was initially designed as a tool to ensure or speed up the discovery of fixpoints for *monotone* functions, it has utility beyond that; non-monotone analyses have been proposed that rely on widening to escape iteration sequences that may be looping rather than ascending [15].

We have assumed the definition of (classical) widening given by Cousot and Cousot [11]. Many definitions found in the literature differ in subtle ways, and not all are strictly equivalent. For example, from the earliest work on widening, P. and R. Cousot pointed out that it is not necessary to restrict widening to be a single mechanism; widening could involve the use of a succession of different mechanisms. Cousot [6] thus views a widening operator as having type $\mathbb{N} \rightarrow (\mathcal{D} \times \mathcal{D}) \rightarrow \mathcal{D}$, to allow for such "dynamic", as well as delayed, widening.

Bourdoncle [4] analysed the use of chaotic iteration with widening and explored how iteration strategy affects both precision and overall efficiency of analysis. A central problem (not discussed in this paper) is how to select a good set W of widening points. Bourdoncle [4] introduced *weak topological orderings* and demonstrated their utility in the choice of W.

Much of the research on widening has been in the context of relational or weakly-relational abstract domains. Widening plays a central role in the seminal work on polyhedral analysis [13]. The implementation problems caused by the non-semantic nature of the left (widener) were previously observed in work on Zones, Octagons and convex polyhedra [2, 28, 29], although a general solution has not been suggested, to our knowledge.

Delayed widening [3] is an obvious approach to limiting the loss of precision incurred by widening. Widening is delayed for a fixed number k of iterations, so that the widening operator associated with a widening point is treated as a join for k steps. More sophisticated delay can be introduced by taking syntactic aspects of the given program into account. Halbwachs, Proy and Roumanoff [25] proposed such a widening "up to" scheme, which has since been extended and dubbed "widening with thresholds" [26]. Simon and King [30] propose a generalisation ("widening with landmarks") in the context of polyhedral analysis.

Widening/narrowing does not always deal well with complex program structure, including nested loops. Much work has focused on improved precision of analysis, at a reasonable cost of overhead. Our particular perspective on widening (and the implementation discipline it enforces) does not preclude the use of a number of these proposed widening-related techniques. This includes "lookahead widening" and similar "analysis guiding" techniques for loops that exhibit multi-phase behaviour [19, 20], widening with landmarks [30], and post-fixpoint improvements such as those suggested by Halbwachs and Henry [24].

A work that is close to ours as far as motivation is concerned, is Mihaila et al.'s recasting of "widening as abstract domain" [27]. The authors also seek a systematic, modular approach to composing abstract domains in the presence of widening, so that ad hoc design or modifications of a fixpoint engine is avoided. They show how different widening strategies can be built into abstract domains, including delayed widening, widening with thresholds, and lookahead widening. The proposed machinery, however, is very different to ours. It assumes that a program location ℓ is a widening point if and only if it is the target of a *back-edge* in a control flow graph, in which case ℓ's join is replaced by a widening operation. Our approach does not restrict the choice of widening points. Rather than force different widening techniques into a given abstract domain, our proposal is to *separate* syntactic and semantic aspects of analysis, by clearly distinguishing the roles of the (syntactic) *widener* and the (semantic) *widenee*.

8 Conclusion

We have proposed an alternative approach to accelerating fixpoint finding in abstract interpretation. The approach, which we have called isolated widening, is a generalisation of the classical widening technique, in that any classical widening can be trivially translated to an I-widening, but the converse does not hold. This generality allows isolated widening to retain any information about the history of widening at a program point, beyond simply the previous and next abstract states, allowing for more flexible widenings than the traditional approach supports, such as delayed widening, all while isolating any added information from the abstract domain itself.

Importantly, this isolation also sidesteps certain pitfalls that arise with classical widenings. For example, it clarifies the distinction between the semantics of an abstract domain, such as Octagons, from the syntactic view of the representation used during widening. Isolated widening makes this distinction explicit, preserving the semantic view in the abstract domain, and transforming the syntactic view into its own view with a distinct semantics in the widening domain. Crucially, this approach keeps the widening information in its own representation rather than immediately transforming it to the abstract domain, avoiding the accidental strengthening of abstract states that Miné and others have observed to undo the effect of widening and cause nontermination of analysis [29]. Additionally, we have shown that I-widenings are compositional, unlike classical widenings, simplifying the implementation of product domains.

We have implemented our approach in the context of the Crab abstract interpretation framework [14], finding that it required minimal effort, and did not affect performance or analysis precision. We conclude that I-widening is a practical generalisation of classical widening.

Acknowledgments

This work was partially supported by the Australian Research Council through Discovery Early Career Researcher Award DE160100568.

References

1. G. Amato and F. Scozzari. Localizing widening and narrowing. In F. Logozzo and M. Fähndrich, editors, *Static Analysis*, volume 7935 of *LNCS*, pages 25–42. Springer, 2013.
2. R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. *Science of Computer Programming*, 58:28–56, 2005.
3. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *ACM Conf. Programming Language Design and Implementation*, pages 196–207, 2003.
4. F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In D. Bjørner, M. Broy, and I. V. Pottosin, editors, *Formal Methods in Programming and Their Applications*, volume 735 of *LNCS*, pages 128–141. Springer, 1993.
5. A. Cortesi, G. Costantini, and P. Ferrara. A survey on product operators in abstract interpretation. In A. Banerjee, O. Danvy, K.-G. Doh, and J. Hatcliff, editors, *Semantics, Abstract Interpretation, and Reasoning about Programs*, volume 129 of *Electronic Proceedings in Theoretical Computer Science*, pages 325–336, 2013.
6. P. Cousot. Semantic foundations of program analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, pages 303–346. Prentice-Hall, 1981.
7. P. Cousot. Forward non-relational infinitary static analysis, 2005. Slide set 18 from MIT Course 16.399, Abstract Interpretation, `http://www.mit.edu/afs/athena.mit.edu/course/16/16.399/www/`.
8. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
9. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282. ACM Press, 1979.
10. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computations*, 2(4):511–547, 1992.
11. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, volume 631 of *LNCS*, pages 269–295. Springer, 1992.
12. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combinations of abstractions in the ASTRÉE static analyzer. In M. Okada and I. Satoh, editors, *Advances in Computer Science (ASIAN 2006)*, volume 4435 of *LNCS*, pages 272–300. Springer, 2007.

13. P. Cousot and N. Halbwachs. Automatic discovery of linear constraints among variables of a program. In *Proc. Fifth ACM Symp. Principles of Programming Languages*, pages 84–97. ACM Press, 1978.
14. Crab: CoRnucopia of ABstractions: A language-agnostic library for abstract interpretation. `https://github.com/seahorn/crab`.
15. G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Abstract interpretation over non-lattice abstract domains. In F. Logozzo and M. Fähndrich, editors, *Static Analysis*, volume 7935 of *LNCS*, pages 6–24. Springer, 2013.
16. G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Interval analysis and machine arithmetic: Why signedness ignorance is bliss. *ACM Transactions on Programming Languages and Systems*, 37(1):1:1–1:35, 2014.
17. G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. An abstract domain of uninterpreted functions. In B. Jobstmann and K. R. M. Leino, editors, *Verification, Model Checking and Abstract Interpretation: Proceedings of the 17th International Conference*, volume 9583 of *LNCS*, pages 85–103. Springer, 2016.
18. G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Dissecting widening: Separating termination from information, 2019. ArXiv document `xxx`.
19. D. Gopan and T. Reps. Lookahead widening. In T. Ball and R. B. Jones, editors, *Computer Aided Verification*, volume 4144 of *LNCS*, pages 452–466. Springer, 2006.
20. D. Gopan and T. Reps. Guided static analysis. In H. Riis Nielson and G. Filé, editors, *Static Analysis*, volume 4634 of *LNCS*, pages 349–365. Springer, 2007.
21. E. Goubault, T. L. Gall, and S. Putot. An accurate join for zonotopes, preserving affine input/output relations. *Electr. Notes Theor. Comput. Sci.*, 287:65–76, 2012.
22. P. Granger. Improving the results of static analyses of programs by local decreasing iterations. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *LNCS*, pages 68–79. Springer, 1992.
23. A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas. The SeaHorn verification framework. In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification, Part 1*, volume 9206 of *LNCS*, pages 343–361, 2015.
24. N. Halbwachs and J. Henry. When the decreasing sequence fails. In A. Miné, editor, *Static Analysis*, volume 7460 of *LNCS*, pages 198–213. Springer, 2012.
25. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11:157–185, 1997.
26. L. Lakhdar-Chaouch, B. Jeannet, and A. Girault. Widening with thresholds for programs with complex control graphs. In T. Bultan and P.-A. Hsiung, editors, *Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 492–502. Springer, 2011.
27. B. Mihaila, A. Sepp, and A. Simon. Widening as abstract domain. In G. Brat, N. Rungta, and A. Venet, editors, *NASA Formal Methods*, volume 7871 of *LNCS*, pages 170–184. Springer, 2013.
28. A. Miné. A new numerical abstract domain based on difference-bound matrices. In O. Danvy and A. Filinski, editors, *Programs as Data Objects*, volume 2053 of *LNCS*, pages 155–172. Springer, 2001.
29. A. Miné. The Octagon abstract domain. In *Proc. Workshop in Analysis, Slicing and Transformation*, pages 310–319, 2001.
30. A. Simon and A. King. Widening polyhedra with landmarks. In N. Kobayashi, editor, *Programming Languages and Systems*, volume 4279 of *LNCS*, pages 166–182. Springer, 2006.