A minimalistic look at widening operators

David Monniaux
CNRS / VERIMAG*

March 7, 2019

Abstract

We consider the problem of formalizing in higher-order logic the familiar notion of widening in abstract interpretation. It turns out that many axioms of widening (e.g. widening sequences are ascending) are not useful for proving correctness. After keeping only useful axioms, we give an equivalent characterization of widening as a lazily constructed well-founded tree. In type systems supporting dependent products and sums, this tree can be made to reflect the condition of correct termination of the widening sequence.

1 The usual framework

We shall first recall the usual definitions of abstract interpretation and widening operators.

1.1 Abstract interpretation

Abstract interpretation is a framework for formalizing approximation relationships arising in program semantics and static analysis [7, 8]. *Soundness* of the abstraction is expressed by the fact that the approximation takes place in a controlled direction. In order to prove that some event is unreachable, we can try computing some superset of the set of reachable states (an *over-approximation*), in the hope that this set does not intersect the event. If we wish to obtain a set of initial states that necessarily result in some event further down the program, we compute an *under-approximation* of the set of initial states that verify that property. Because most static analysis is concerned with over-approximations, we shall only consider this case in this article.

Most introductory materials on abstract interpretation describe abstraction as a *Galois connection* between a concrete space S (typically, the powerset $\mathcal{P}(\Sigma)$ of the set of states Σ of the program, or the powerset of the set of finite execution traces Σ^* of the program) and an abstract space $S^\sharp$. For instance, if the program state consists in a program counter location, taken within a finite set P of program locations, and three integer variables, $\Sigma = P \times \mathbb{Z}^3$, $S = \mathcal{P}(P \times \mathbb{Z}^3)$, the abstract state can be, for instance, $P \to (\{\bot\} \cup I^3)$, where P is the set of program locations, $a \to b$ denotes the set of functions from a

*VERIMAG is a joint laboratory of CNRS, Université Joseph Fourier and Grenoble-INP.

to b, I is the set of well-formed pairs (a, b) defining intervals ($a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{+\infty\}$ and $a \leq b$) and $\bot$ is a special element meaning "unreachable". S and $S^\sharp$ are ordered; here, S is ordered by set inclusion $\subseteq$ and $S^\sharp$ is ordered by $\sqsubseteq_P$, the pointwise application of $\sqsubseteq$ for all program locations: $\bot \sqsubseteq x^\sharp$ for all x in $S^\sharp$, and $((a_1, b_1), (a_2, b_2), (a_3, b_3)) \sqsubseteq ((a'_1, b'_1), (a'_2, b'_2), (a'_3, b'_3))$ if for all $1 \leq i \leq 3$, $a'_i \leq a_i$ and $b'_i \geq b_i$. For the sake of simplicity, we shall give examples further on where P is a singleton; the generalization to any finite P is straightforward. $P \to (\{\bot\} \cup I^3)$ is then isomorphic to $\{\bot\} \cup I^3$ and we shall thus consider, as a running example, the case where S is $\mathcal{P}(\mathbb{Z}^3)$ and $S^\sharp$ is $\{\bot\} \cup I^3$.

S and $S^\sharp$ are connected by an *abstraction map* α and a *concretization map* γ. γ maps any abstract state $x^\sharp$ to the set of concrete states that it represents;[1] here, $\gamma((a_1, b_1), (a_2, b_2), (a_3, b_3))$ is the set of triples (v_1, v_2, v_3) such that for all $1 \leq i \leq 3$, $a_i \leq v_i \leq b_i$. α maps a set x of concrete states to the "best" (least) abstract element $x^\sharp$ such that $x \subseteq \gamma(x^\sharp)$. Here, if $x \subseteq \mathbb{Z}^3$, then for all $1 \leq i \leq 3$, $a_i = \inf_{(v_1,v_2,v_3)\in x} v_i$ and $b_i = \sup_{(v_1,v_2,v_3)\in x} v_i$. $\subseteq$ and $\sqsubseteq$ must be compatible: if $x^\sharp \sqsubseteq y^\sharp$, then $\gamma(x^\sharp) \subseteq \gamma(y^\sharp)$.

Abstract interpretation replaces a possibly infinite number of concrete program execution, which cannot be simulated in practice, by a simpler "abstract" execution. For instance, one may replace running a program using our three integer variables over all possible initial states by a single abstract execution with interval arithmetic. The resulting final intervals are guaranteed to contain all possible outcomes of the concrete program. More formally, if one has a transition relation $\tau \subseteq \Sigma \times \Sigma$, one defines the forward concrete transfer function $f_\tau : S \to S$ as $f_\tau(x) = \{\sigma' \mid \sigma \to_\tau \sigma' \wedge \sigma \in x\}$. $f_\tau(W)$ is the set of states reachable in one forward step from W. We say that the *abstract transfer function* ${f_\tau}^\sharp(x^\sharp)$ is a correct abstraction for f_τ if for all $x^\sharp$, $f_\tau \circ \gamma(x^\sharp) \subseteq \gamma \circ {f_\tau}^\sharp(x^\sharp)$. This *soundness property* means that if we have a superset of the concrete precondition, we get a superset of the concrete postcondition.

As usual in program analysis, obtaining loop invariants is the hardest part. Given a set $x_0 \subseteq \Sigma$ of initial states, we would like to obtain a superset of the set of reachable states $x_\infty = \{\sigma' \mid \sigma \to_\tau^* \sigma' \wedge \sigma \in x_0\}$. The sets of states x_n reachable in at most n steps from x_0 is defined by induction: $x_{n+1} = \phi(x_n)$, where $\phi(x) = f_\tau(x) \cup x_0$. The sequence (x_n) is ascending, and its limit is

[1] In some presentations of abstract interpretation, abstract elements $x^\sharp$ are identified with their concretization $\gamma(x^\sharp)$. For instance, one talks directly of the interval $[a, b]$, not of the pair (a, b). This can makes explanations smoother by clearing up notations. It is however important for some purposes to distinguish the machine representation of an abstract element $x^\sharp$ from its concretization $\gamma(x^\sharp)$, if only because γ may not be injective. For instance, $x = y \wedge x \leq 1$ and $x = y \wedge y \leq 1$ define exactly the same part of the plane (as geometrical convex polyhedra) but are different in their machine representation. This is the same difference as that between the *syntax* and the *semantics* of a logic.

Certain abstract operations may be sensitive to the syntax of an abstract element; that is, they may yield different results for $x^\sharp$ and $y^\sharp$ even though $\gamma(x^\sharp)$ and $\gamma(y^\sharp)$. For instance, the original widening operator defined for the convex polyhedra [9] was sensitive to syntax, and was later improved, losing this sensitivity [12, p. 56–57][11, §2.2].

Also, while in many cases $\sqsubseteq$ is defined by $a \sqsubseteq b \iff \gamma(a) \subseteq \gamma(b)$, this relation may sometimes be too costly or impossible to compute, and some smaller relation may be used. Finally, since our goal is to write programs and proofs in a proof assistant based on intuitionistic type theory, we thought it best to clearly separate the computational, constructive content from the non-computational content (the set of reachable states of the program cannot be defined constructively, in general).

x_∞, which is the least fixed point of ϕ by Kleene's fixed point theorem; this sequence is thus often known as *Kleene iterations*. x_∞ is also known as the *strongest invariant* of the program. An *inductive invariant* is a set x such that $x_0 \subseteq x$ and $f_\tau(x) \subseteq x$, and by Tarski's theorem, the intersection of all such sets is x_∞.

Obviously, the set of all possible states (often noted $\top$) is an inductive invariant, but it is uninteresting since it cannot be used to prove any interesting property of the program. A major goal of program analysis is to obtain program invariants x that are strong enough to prove some interesting properties, yet not too costly too establish.

In some cases, interesting inductive invariants may be computed directly. Various approaches have recently been proposed for the direct computation of invariants, without Kleene iterations. Costan et al. [5] proposed a method for computing least fixed points in the lattice of real intervals by downward *policy iteration*, also known as *strategy iteration*, a technique borrowed from game theory; they later extended their framework to other domains. Gawlitza and Seidl [10] proposed a method for computing least fixed points in certain lattices by upward strategy iteration. Monniaux [15, 16] showed that least fixed point problems in some lattices expressing numerical constraints can be reduced to *quantifier elimination* problems, which in turn can be solved algorithmically. Other recent proposals include expressing the least invariant problem in the abstract lattice directly as a constrained minimization problem, then solving it with operational research tools [6]. One common factor to these approaches is that they target specific classes of abstract domains and programs; in addition to lack of genericity, they may also suffer from high complexity.

1.2 Abstract Kleene iterations and widening operators

The more traditional approach to finding inductive invariants by abstract interpretation is to perform *abstract Kleene iterations*. Let $x_0^\sharp$ be an abstraction of x_0. Define $\phi^\sharp(x^\sharp) = f_\tau^\sharp(x^\sharp) \sqcup x_0^\sharp$, where $\sqcup$ is a sound overapproximation of the concrete union $\cup$: $\gamma(x^\sharp) \cup \gamma(y^\sharp) \subseteq \gamma(x^\sharp \sqcup y^\sharp)$. From the soundness of $f_\tau^\sharp$ and $\sqcup$, $\phi^\sharp$ is a sound abstraction of ϕ: for all $x^\sharp$, $\phi \circ \gamma(x^\sharp) \subseteq \gamma \circ \phi^\sharp(x^\sharp)$. By induction, for all n, $x_n \subseteq \gamma(x_n^\sharp)$.

In many presentations of abstract interpretation, it is supposed that the abstract transfer function $f_\tau^\sharp$ and the abstract union $\sqcup$ are monotonic. Intuitively, this means that if the analysis has more precise information at its disposal, then its outcome is more precise. This is true for elementary transfer functions in most abstract domains, and thus of their composition into abstract transfer functions of more complex program constructions. A well-known exception is when the abstract transfer function is itself defined as the overapproximation of a least fixed-point operation using a widening operator (see below), yet there exist less well-known cases where the abstract transfer function may be non-monotonic.[2]

[2]Such is for instance the case of the symbolic constant propagation domain proposed by Miné [14, §5][13, §6.3.4]. The full symbolic propagation strategy can induce non-monotonic effects: if the analysis knows more relationships, it can perform spurious rewritings and paradoxically provide a less precise result. The same is true of the linearization step: for nonlinear terms, a choice has to be made between several valid linearizations; while all choices lead to sound results, they do not have the same precision and the choice heuristic does not necessarily

Let us nevertheless temporarily assume that $f_\tau^\sharp$ and $\sqsubseteq$ and, thus, $\phi^\sharp$, are monotonic, and that $a^\sharp, b^\sharp \sqsubseteq a^\sharp \sqcup b^\sharp$ for all $a^\sharp$ and $b^\sharp$. Then ${x_0}^\sharp \sqsubseteq {x_1}^\sharp$ and by induction, for all n, by repeatedly application of monotonic $\phi^\sharp$, $x_n^\sharp \sqsubseteq x_{n+1}^\sharp$. The sequence $x_n^\sharp$ is ascending. If this sequence is stationary, there is a N such that $x_{N+1}^\sharp = x_N^\sharp$. Then, $\gamma(x_N^\sharp) = \gamma(x_{N+1}^\sharp) = \gamma(f_\tau^\sharp(x_N^\sharp) \sqcup x_0^\sharp) \supseteq \gamma \circ f_\tau^\sharp(x_N^\sharp) \supseteq f \circ \gamma(x_N^\sharp)$, and $\gamma(x_N^\sharp) = \gamma(x_{N+1}^\sharp) = \gamma(f_\tau^\sharp(x_N^\sharp) \sqcup x_0^\sharp) \supseteq \gamma(x_0^\sharp)$, which means that $\gamma(x_N^\sharp)$ is an inductive invariant. Obviously, if the abstract domain $S^\sharp$ is finite, then any ascending sequence is stationary.[3]

More generally, the same results hold for any domain of *finite height* (there exists an integer L such that any strictly ascending sequence has at most length L), and, even more generally, for any domain satisfying the *ascending chain condition* (there does not exist any infinite strictly ascending sequence). Yet, even the very simple domain of products of intervals that we defined earlier does not satisfy the ascending chain condition!

In domains that do not satisfy the ascending condition, the abstract Kleene iterations may fail to converge in finite time. Such is the case, for instance, of the interval abstraction of the program with a single integer variable defined by the transition system τ: for all n, $n \to_\tau n+1$, and the initial state is 0. The best abstract transfer function $\phi^\sharp$ maps a pair $(0, n)$ representing an integer interval $\{0, \ldots, n\}$ to the pair $(0, n+1)$, thus the abstract Kleene iterations are $x_n^\sharp = (0, n)$ and the analysis fails to converge in finite time.

The traditional solution to the convergence problem in domains that do not satisfy the ascending chain condition is to use a *widening operator*, which is a form of convergence accelerator applied to abstract Kleene iterations [7, Def. 4.1.2.0.4][8, §4].[4] Intuitively, the widening operation examines the first abstract Kleene iterations and conjectures some possible over-approximation of the limit, which is then checked for stability; further iterations may be necessary until an inductive invariant is reached.

Here is the most common definition:

Definition 1. A widening operator $\triangledown$ on an abstract domain $(S^\sharp, \sqsubseteq)$ is a binary operator that verifies the three following properties:

1. $x^\sharp \sqsubseteq x^\sharp \triangledown y^\sharp$
2. $y^\sharp \sqsubseteq x^\sharp \triangledown y^\sharp$
3. for any sequence $v_n^\sharp$, a sequence of the form $u_{n+1}^\sharp = u_n^\sharp \triangledown v_n^\sharp$ is ultimately stationary.

We can then use $u_0^\sharp = x_0^\sharp$, $u_{n+1}^\sharp = u_n^\sharp \triangledown \phi^\sharp(u_n^\sharp)$. By the third property of the widening operator, there exists N such that $u_N^\sharp = u_N^\sharp \triangledown \phi^\sharp(u_N^\sharp)$. Thus, $\phi^\sharp(u_N^\sharp) \sqsubseteq$

choose the best one.

[3] This explains the popularity of Boolean abstractions: $S^\sharp$ is the set of sets of bit vectors of fixed length L, and these sets are often represented by *reduced ordered binary decision diagrams* (ROBDD) [4]. Reachability analysis in BDD-based model-checkers is thus a form of Kleene iteration in the BDD space. Very astute implementation techniques, involving generalized hashing of data structures, ensure that equality tests take constant time and that $\phi^\sharp$ is computed efficiently.

[4] For each infinite height domain, one or more widening operators must be designed. Consequently, most literature on abstract interpretation domains includes descriptions of widening operators. We shall list here only the earliest examples, namely the widenings on intervals and convex polyhedra [7, 9, 11, 12].

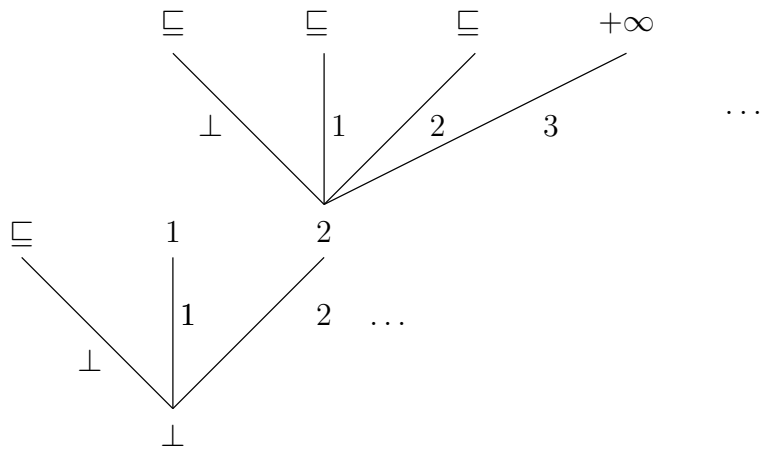

Figure 1: Interpretation of widening as a well-founded tree. Each node represents a proposal $u_n^\sharp$ from the widening system. Each edge is labelled with the answer $v_n^\sharp$ from the analysis system. The widening system either answers $\sqsubseteq$ when it determines that $v_n^\sharp \sqsubseteq u_n^\sharp$, either makes a new proposal. A path from the root of the tree is an abstract Kleene iteration sequence. The well-foundedness of the tree ensures the termination of such sequences.

$u_N^\sharp$. But then, by the same reasoning as for stationary Kleene iterations without widening, $\phi^\sharp(u_N^\sharp) \sqsubseteq u_N^\sharp$ and thus $\gamma \circ f_\tau(u_N^\sharp) \cup \gamma(x_0^\sharp) \subseteq \gamma(u_N^\sharp)$ and $\gamma(u_N^\sharp)$ is an inductive invariant.

Let us now have a second look at the hypotheses that we really used to establish that result. Though it is often assumed that the abstract domain is a complete lattice, and that the abstract transfer function is monotonic, we never used either hypotheses. In fact, the only hypotheses that we used are:

- f_τ is monotonic and the concrete domain $\mathcal{P}(S)$ is a complete lattice, thus ϕ has a least fixed point which is the least inductive invariant of the program.
- For all $a^\sharp$ and $b^\sharp$, $b^\sharp \sqsubseteq a^\sharp \triangledown b^\sharp$.
- For all sequence $v_n^\sharp$, any sequence defined by $u_{n+1}^\sharp = u_n^\sharp \triangledown v_n^\sharp$ is stationary.

2 Relaxation of conditions and interpretation in inductive types

During our work on the Astrée tool [3], and when formalizing the notion of widening in the Coq proof assistant [1],[5] we realized that the usual definitions of abstract domains and widenings are unnecessarily restrictive for practical purposes. Pichardie [17, §4.4] already proposed a relaxation of these conditions, but his definition of widenings is still fairly complex. We propose here a drastically reduced definition of widenings, which subsumes both the $\sqsubseteq$ ordering and the $\triangledown$ operator.

[5] Coq is a proof assistant in higher order logic available from `http://coq.inria.fr`.

Definition 2. A widening system is an algorithm that proposes successive abstract elements $u_0^\sharp, u_1^\sharp, \ldots, u_n^\sharp$ to the rest of the analyzer, and receives $v_n^\sharp$ from it — in practical use, $v_n^\sharp = \phi^\sharp(u_n^\sharp)$ and $\phi^\sharp$ is an abstraction of the concrete transformer ϕ of a loop or, more generally, of a monotonic system of semantic equations. It can then either terminate with some guarantee that $\gamma(v_n^\sharp) \subseteq \gamma(u_n^\sharp)$, or propose the next element $u_{n+1}^\sharp$. The system never provides infinite sequences.

It is obvious that any widening that verifies the conditions of Def. 1 also verifies these conditions. Note that Def. 2 is strictly laxer than Def. 1. For instance, we make no requirement that $\gamma(u_n^\sharp) \subseteq \gamma(u_{n+1}^\sharp)$; a widening system could first try some ascending sequence $u_0^\sharp, \ldots, u_n^\sharp$, realize that it is probably a bad idea to go this way, and restart with another sequence $u_{n+1}^\sharp, \ldots$.

Definition 2 can be easily recast as couple of mutually inductive types :

$$\begin{array}{ll} \mathit{widening} & \cong S^\sharp \times (S^\sharp \to \mathit{answer}) \\ \mathit{answer} & \equiv \mathit{termination} \mid \mathit{next} \text{ of } \mathit{widening} \end{array} \tag{1}$$

These types define a tree, as depicted in Fig. 1. A run of the widening system, that is, a sequence $u_0^\sharp, u_1^\sharp, \ldots, u_n^\sharp$, corresponds to a path in the tree. The absence of infinite widening sequences means that the tree should be well-founded. Note that, even in an eager language such as Objective Caml, this tree is never constructed in memory: its nodes are constructed on demand by application of the function $S^\sharp \to \mathit{answer}$.

In a higher-order type system with dependent sums and products such as the Calculus of inductive constructions (as in Coq), the above inductive datatype can be adorned with proof terms. A tree node *widening* is a pair $(u^\sharp, a)$ where a maps each $v^\sharp$ to an *answer*. $a(v^\sharp)$ is either "$\sqsubseteq$", carrying a proof term stating that $\gamma(v^\sharp) \subseteq \gamma(u^\sharp)$, or another *widening* tree node.

3 Implementation in Coq

We shall first show how to implement our concept of widening system in Coq, then we shall give a few concrete examples of how common abstract interpretation techniques can be implemented within this framework.

3.1 Framework

We assume we have an abstract domain `S` with an ordering `domain_le` (representing $\sqsubseteq$). In practice, this ordering is supposed to be decidable: there exists a function `domain_le_decide` that takes x and y as inputs and decides whether $x \sqsubseteq y$.

The *answer* is the disjunctive sum `widening + {domain_le y x}`: it either provides a new `widening` object, either a proof that $y \sqsubseteq x$. By inlining this type into the definition of `widening`, we obtain:

```
Variable S : Set.
Hypothesis domain_le : S -> S -> Prop.
Hypothesis domain_le_decide :
    forall x y : S,
      { domain_le x y } + {~ (domain_le x y) }.
```

```
Inductive widening: Set :=
  widening_intro : forall x : S,
    (forall y : S, widening + {domain_le y x}) -> widening.
```

Since `widening` is a inductive type, defining well-founded trees, it is possible to define functions by induction over elements of that type. One especially interesting inductively defined function takes $f^\sharp : S^\sharp \to S^\sharp$ as a parameter and computes x such that $f^\sharp(x^\sharp) \sqsubseteq x^\sharp$:

```
Section Recursor.
Variable f : S -> S.

Fixpoint abstract_lfp_rec
  (iteration_step : widening) :
  { lfp : S | domain_le (f lfp) lfp } :=
  let (x, xNext) := iteration_step in
  match xNext (f x) with
  | inleft next_widening => abstract_lfp_rec next_widening
  | inright fx_less_than_x => exist (fun x => domain_le (f x) x)
      x fx_less_than_x
  end.

End Recursor.
```

For ease of use, we pack `S`, `domain_le`, an abstraction relation `domain_abstracts` and other related constructs into one single `domain` record. `(domain_abstracts` $x^\sharp$ x`)` means that $x \subseteq \gamma(x^\sharp)$.

3.2 Examples

In numerical abstract domains, it is common to use "widening up to" [11, §3.2] or "widening with thresholds" [2, §6.4][3, §7.1.2]: one keeps an ascending sequence $z_1^\sharp, \ldots, z_n^\sharp$ of "magical" values, and $x^\sharp \triangledown y^\sharp$ is the least element $z_k^\sharp$ greater than $x^\sharp \sqcup y^\sharp$. For instance, instead of widening a sequence of integer intervals $[0,1]$, $[0,2]$ etc. to $[0,+\infty[$, we may try some "magical" values such as $[0,255]$, $[0,32767]$ etc. Yet, if all elements in the sequence fail to define an inductive invariant, we are forced to try $[0,+\infty[$. Otherwise said, after trying the "magical" values, we revert to the usual brutal widening on the intervals.

This is easily modeled within our framework by a "widening transformer": taking a widening W as input and a finite list l of values, it outputs a widening W' that first applies the thresholds and, as a last resort, calls W. `Variable T : domain` is a parameter defining the original domain and original widening, which is used as the last resort by our transformed widening.

```
Section Widening_ramp.
Variable T : domain.

Fixpoint ramp_widening_chain_search (bound : (domain_set T))
  (ramp : (list (domain_set T))) { struct ramp } :
  (list (domain_set T)) :=
```

```
...

Fixpoint ramp_widening_chain (ramp : (list (domain_set T))) :
  (widening_chain (domain_set T) (domain_le T)) := ...
```

A trick often used in static analysis is to delay the widening [3, §7.1.3]. Instead of performing $\triangledown$ at each iteration, one performs $\sqcup$ for a finite number of steps, then one tries $\triangledown$ again. For termination purposes, it suffices that there is some "fairness property": $\triangledown$ should not be delayed infinitely. One can for instance choose to delay widening by n steps of $\sqcup$ after each widening step. This is again implemented as a "widening transformer":

```
Definition delayed_widening_each_step :
  nat -> (widening_chain (domain_set T) (domain_le T)).
```

We can similarly build a product domain $S_1^\sharp \times S_2^\sharp$. The widening on couples $(a_1, a_2)\triangledown(b_1, b_2) = (a_1\triangledown_1 b_1, a_2\triangledown_2 b_2)$ is implemented by a "widening transformer" taking one widening W_1 on $S_1^\sharp$ and a widening W_2 on $S_2^\sharp$ as inputs, and producing a widening on $S_1^\sharp \times S_2^\sharp$ by syntactic induction on W_1 and W_2: if $a_1 \sqsubseteq_1 b_1 \wedge a_2 \sqsubseteq_2 b_2$, then $(a_1, a_2) \sqsubseteq (b_1, b_2)$ for the product ordering and one terminates; if $a_1 \sqsubseteq_1 b_1$ but $a_2 \not\sqsubseteq_2 b_2$ then one stays on a_1 but moves one step into W_2 (and *mutatis mutandis* reversing the coordinates); if $a_1 \sqsubseteq_1 b_1$ and $a_2 \not\sqsubseteq_2 b_2$, then one moves into both W_1 and W_2. This implements the usual widening on products. This construct can be generalized to any finite products of domains.

4 Conclusion

By seeing the combination of the computational ordering $\sqsubseteq$ and the widening operator $\triangledown$ as a single inductive construct, one obtains an elegant characterization extending the usual notion of widening in abstract interpretation, suitable for implementation in higher order logic.

References

[1] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004. ISBN 3-540-20854-2.

[2] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In Torben Æ. Mogensen, David A. Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation*, number 2566 in LNCS, pages 85–108. Springer, 2002. ISBN 3-540-00326-6. doi: 10.1007/3-540-36377-7_5.

[3] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static

analyzer for large safety-critical software. In *Programming Language Design and Implementation (PLDI)*, pages 196–207. ACM, 2003. ISBN 1-58113-662-5. doi: 10.1145/781131.781153.

[4] Edmund M. Clarke, Jr, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999. ISBN 0-262-03270-8.

[5] Alexandru Costan, Stephane Gaubert, Éric Goubault, Matthieu Martel, and Sylvie Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification (CAV)*, number 4590 in LNCS, pages 462–475. Springer, 2005. ISBN 3-540-27231-3. doi: 10.1007/11513988_46.

[6] Patrick Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, pages 1–24. Springer, January 17–19 2005. ISBN 3-540-24297-X. doi: 10.1007/b105073. URL `http://www.di.ens.fr/~cousot/COUSOTpapers/VMCAI05.shtml`.

[7] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. State doctorate thesis, Université scientifique et médicale de Grenoble & Institut national polytechnique de Grenoble, 1978. URL `http://tel.archives-ouvertes.fr/tel-00288657/en/`. In French.

[8] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. of Logic and Computation*, pages 511–547, August 1992. ISSN 0955-792X. doi: 10.1093/logcom/2.4.511.

[9] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Principles of Programming Languages (POPL)*, pages 84–96. ACM, 1978. doi: 10.1145/512760.512770.

[10] Thomas Gawlitza and Helmut Seidl. Precise fixpoint computation through strategy iteration. In Rocco de Nicola, editor, *Programming Languages and Systems (ESOP)*, volume 4421 of *LNCS*, pages 300–315. Springer, 2007. ISBN 978-3-540-71316-6. doi: 10.1007/978-3-540-71316-6_21.

[11] Nicolas Halbwachs. Delay analysis in synchronous programs. In *Computer Aided Verification (CAV)*, pages 333–346. Springer, 1993. ISBN 3-540-56922-7. doi: 10.1007/3-540-56922-7_28.

[12] Nicolas Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme*. PhD thesis, Université scientifique et médicale de Grenoble & Institut national polytechnique de Grenoble, 1979. URL `http://tel.archives-ouvertes.fr/tel-00288805/en/`. In French.

[13] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École polytechnique, Palaiseau, France, December 2004. In English.

[14] Antoine Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *LNCS*, pages 348–363. Springer, January 2006. ISBN 3-540-31139-4. doi: 10.1007/11609773.

[15] David Monniaux. Automatic modular abstractions for linear constraints. In *POPL (Principles of programming languages)*. ACM, 2009. ISBN 978-1-60558-379-2. doi: 10.1145/1480881.1480899.

[16] David Monniaux. Optimal abstraction on real-valued programs. In Gilberto Filé and Hanne Riis Nielson, editors, *Static analysis (SAS '07)*, volume 4634 of *LNCS*, pages 104–120. Springer, 2007.

[17] David Pichardie. *Interprétation abstraite en logique intuitionniste : extraction d'analyseurs Java certifiés*. PhD thesis, Université Rennes 1, 2005. In French.