“Fixing” the specification of widenings

Enea Zaffanella

University of Parma, Italy

Venice, May 2022

Acknowledgements

People deserving a *'thank you'*

- workshop organizers
- developers and maintainers of
  - open source abstract domain libraries
  - open source static analysis tools
- Vincenzo Arceri (helpful comments)

Abstract Interpretation (AI)

AI: a mature research field

- more than 45 years of research and development
- solid theoretical results
- strong practical results (e.g., many AI-based static analysis tools)

Maturity indicators

- AI tools are firmly based on theoretical results (shortcuts taken are identifiable)
- AI tools are industrialized and commercialized
- collaborative work (libraries, frameworks): can extend/combine/compare AI tools
- artifacts, competitions, repeatability . . . : we can tell when things are going wrong

Why this talk?

Questioning one of the previous maturity indicators

- *can we **easily** tell when things are going wrong?*
- question triggered by *a specific AI tool crash* (personal experience)
- talk is about the *lesson learned* during investigation of this crash
- lesson learned is not tool specific, i.e., it may be generally useful

Focus on AI tools that are . . .

Extensible: separation of concerns

- fixed (maybe configurable) *AI engine*
- several different *abstract domains* used as plug-ins
  - experiment with new abstract domains
  - compare alternative implementations of an abstract domain

Open source: white box

- (trying to) understand what the analyzer is doing

Brief story behind the crash

Goal: test the abstract domains developed in PPLite library

- PPLite implements several polyhedra domains: Poly, U_Poly, F_Poly, ...
- PPLite also provides an Apron interface wrapper

Testing with static analyzer PAGAI (Verimag)

- round one (2018): testing Poly, U_Poly $\Longrightarrow$ OK!
- round two (2019-2020): testing F_Poly, UF_Poly $\Longrightarrow$ OK!

Testing with static analyzer IKOS (NASA JPL)

- round one (2020): testing Poly $\Longrightarrow$ OK! (or so it seemed ...)

Round two (2021): testing IKOS with PPLite's F_Poly

segmentation fault

Reacting to the unexpected crash

Software developer mindset

- bug identified after repeating test in debug mode:
  precondition failure when AI engine calls the widening operator
- *blame culture*: ~~it wasn't me (PPLite developer), it is an IKOS fault~~

AI designer mindset

- *no blame culture*: asking a few questions
  - why such a precondition?
  - why its violation got unnoticed before?
  - which is the best approach to avoid this problem?

Widening specification in [Cousot^2 POPL76, PLILP92]

Another method [CC76,CC77a] for enforcing termination of the abstract interpretation consists in using a *widening* $\nabla \in L \times L \longmapsto L$ such that:

$$\forall x, y \in L : x \sqsubseteq x \nabla y \quad (6)$$

$$\forall x, y \in L : y \sqsubseteq x \nabla y \quad (7)$$

for all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not strictly increasing . (8)

It follows, as shown by Prop. 33 in the appendix, that the *upward iteration sequence with widening*:

$$\begin{aligned} \hat{X}^0 &= \bot \\ \hat{X}^{i+1} &= \hat{X}^i && \text{if } F(\hat{X}^i) \sqsubseteq \hat{X}^i \\ &= \hat{X}^i \nabla F(\hat{X}^i) && \text{otherwise} \end{aligned} \quad (9)$$

is ultimately stationary and its limit $\hat{A}$ is a sound upper approximation of $\text{lfp}_{\bot}(F)$[6]. Observe that if L is a join-semi-lattice (the least upper bound $x \sqcup y$

Classical (safe) widening specification [Cousot^2 POPL76, PLILP92]

Separating two different points of view

- point of view of **abstract domain** designer:
  - requires: nothing
  - ensures (safe): $x \sqsubseteq x \nabla y$ and $y \sqsubseteq x \nabla y$
  - also ensures stabilization (not relevant for this talk)
- point of view of **AI engine** designer:
  - usage (**risky**): $x_{i+1} \leftarrow x_i \nabla F(x_i)$

In fewer words

- widening is meant to **replace join**

Do not miss footnote 6 in [Cousot^2 PLILP92]

[6] Numerous variants are possible. For example, we might assume $x \sqsubseteq y$ in (6) and (7), and use $\hat{X}^{i+1} = \hat{X}^i \triangledown (\hat{X}^i \sqcup F\left(\hat{X}^i\right))$ in (9), or use a different widening for each iterate (as in [Cou81]) or even have a widening which depends upon all previous iterates.

Alternative (**risky**) widening specification [Cousot[2] PILP92, footnote 6]

Separating two different points of view

- point of view of **abstract domain** designer:
  - requires (**risky**): $x \sqsubseteq y$
  - ensures (safe): $y \sqsubseteq x \nabla y$
  - also ensures stabilization
- point of view of **AI engine** designer:
  - usage (safe): $x_{i+1} \leftarrow x_i \nabla \left(x_i \sqcup F(x_i)\right)$

In fewer words

- widening is meant to be used **in addition to join**

What about implementations of abstract domains?

Focusing on (open source) numerical domains

- Apron library: boxes, octagons, polyhedra, . . .
- ELINA: octagons and polyhedra
- PPL (Parma Polyhedra Library): boxes, octagons, polyhedra, . . .
- PPLite: polyhedra
- VPL (Verified Polyhedra Library): polyhedra
- domains embedded into AI tools (Frama-C, Goblint, Jandom, IKOS, SeaHorn, . . .)

Reasoning applicable to non-numeric domains

- GoLiSA uses a domain of finite automata for string analysis

Example: interval widening

Classical implementation

- $[\ell_0, u_0] \nabla [\ell_1, u_1] \stackrel{\text{def}}{=} [(\ell_1 < \ell_0 \mathbin{?} -\infty : \ell_0), (u_0 < u_1 \mathbin{?} +\infty : u_0)]$
- adopted in, among others, Apron boxes

Alternative implementation

- requires $[\ell_0, u_0] \subseteq [\ell_1, u_1]$
- $[\ell_0, u_0] \nabla [\ell_1, u_1] \stackrel{\text{def}}{=} [(\ell_1 \neq \ell_0 \mathbin{?} -\infty : \ell_0), (u_0 \neq u_1 \mathbin{?} +\infty : u_0)]$
- adopted in **PPL boxes**

Example: polyhedra (standard) widening

Classical implementations

- none (of those I have checked)
- note that the **specifications** in [CH78,H79th] are safe

Alternative implementations

- all of them: **Apron**, **ELINA**, **PPL**, **PPLite**, **VPL**
- same holds for the more precise widenings in **PPL** [BHRZ SAS03] and **PPLite** [BZ IC20]

Example: dummy widening for Noetherian (ACC) domains

Classical implementation

- $x \nabla y \stackrel{\text{def}}{=} x \sqcup y$
- adopted in, among others, Frama-C's sign domain:

```
sign_domain.ml: let widen = join
```

Alternative implementation

- requires $x \sqsubseteq y$
- $x \nabla y \stackrel{\text{def}}{=} y$
- adopted in **Frama-C's generic lattice set**:

```
abstract_inter.ml: let widen _wh _t1 t2 = t2
```

Classifying implementations of widenings

Abstract Domain	safe impl (no precondition)	**risky** impl (requires $x \sqsubseteq y$)
boxes	IKOS, Apron*, ...	**PPL**
bounded differences	IKOS	**PPL**
octagons	IKOS, Apron*, ...	**PPL**
parallelotopes	Jandom	
polyhedra		**Apron**, **ELINA**, **PPL**, **PPLite**, **VPL**
finite automata	GoLiSA	

Classifying (open source) AI engines

	safe AI engine join+widen (**alt. widen spec**)	**risky** AI engine **only widen** (classical widen spec)
LMU (Germany)	CPAchecker	
CEA-List (France)	Frama-C	
TUM (Germany)	Goblint	
Verimag (France)	Interproc & PAGAI	
Univ. Chieti-Pescara (Italy)	Jandom + **PPL**	**Jandom** (w/o PPL)
Technion (Israel)		**DIZY**
NASA JPL (USA)		**IKOS**
Univ. Venezia (Italy)		**LiSA** & **GoLisa**
Univ. Waterloo (Canada)		**SeaHorn**

Combining AI engine and widening classifications

AI engine \ widening	safe (no precondition)	risky (requires $x \sqsubseteq y$)
safe (join+widen)		
risky (only widen)		

Note: arbitrary choices for exposition purposes

PAGAI $\equiv$ any safe AI engine
IKOS $\equiv$ any **risky** AI engine

octagons $\equiv$ any safe widening
polyhedra $\equiv$ any **risky** widening

safe+**risky**: alternative approach [Cousot^2 PLILP92, footnote 6]

AI engine \ widening	safe (no precondition)	**risky** (requires $x \sqsubseteq y$)
safe (join+widen)		PAGAI + **polyhedra**
risky (only widen)		

Pros and Cons

- Pros: safe
- Cons: may lose some **precision** (explained later)

safe+safe: wearing belt and suspenders

widening / AI engine	safe (no precondition)	risky (requires $x \sqsubseteq y$)
safe (join+widen)	PAGAI + octagons	PAGAI + **polyhedra**
risky (only widen)		

Pros and Cons

- Pros: safe
- Cons: may lose some **efficiency** (useless joins before widening)

risky+safe: classical approach [Cousot^2 POPL76]

AI engine \ widening	safe (no precondition)	**risky** (requires $x \sqsubseteq y$)
safe (join+widen)	PAGAI + octagons	PAGAI + **polyhedra**
risky (only widen)	**IKOS** + octagons	

Pros and Cons

- Pros: safe; efficient (avoids useless joins); more precise (on some domains)
- Cons: none, as far as I can tell

Can safe widenings be more **precise** than **risky widenings**?

Short answer: YES

- non-lattice abstract domains (e.g., *parallelotopes*) may have no least upper bound
- join chooses among alternative, incomparable upper bounds
    - **poor choice can lead to precision losses**
- computing joins independently from widenings means that the choice is made without appropriate context information
- see Amato *et al.* [AS NSAD12, ARS SCP17] for widening parallelotopes
- see Gange *et al.* [GNSSS SAS13] for other non-lattice domains

Can safe widening be as **efficient** as **risky widening**?

Short answer: YES

- **risky widening** moves join computation overhead into to the AI engine
  - it is like **hiding the dust under the carpet**
- safe widening, by merging join and widening, can trigger optimizations
  - polyhedra standard widening: cheaper *constraint hull* can replace *convex polyhedral hull* (which *uselessly* computes new constraint slopes)
  - an *ad hoc* implementation can optimize further
- side note: just a *small fraction* of the static analysis time is spent in widenings

risky+**risky**: ouch!

widening / AI engine	safe (upper bound)	**risky** (requires $x \sqsubseteq y$)
safe (join+widen)	PAGAI + octagons	PAGAI + **polyhedra**
risky (only widen)	**IKOS** + octagons	**IKOS** + **polyhedra**

Pros and Cons

- Pros: irrelevant (it's **unsafe**!)
- Cons: **unsafe**! Things can go wrong!

Things are going wrong, despite . . . (1 of 4)

Widening precondition $x \sqsubseteq y$ often recalled in the literature

- footnote 6 in [Cousot^2 PLILP92] (ditto)
- example in [BHRZ SAS03, SCP05] shows safety issue on polyhedra
- quoting Halbwachs from [HH SAS12] (also recalled in [BH FMSD18]):

  widening operators are generally designed under the assumption that their first operand is smaller than the second one
- [Cousot VMCAI15] assumes F extensive in the AI engine
  - footnote 5 suggests using $\phi \stackrel{\text{def}}{=} x \sqcup F(x)$ (i.e., join+widen) to force extensivity
  - footnote 6 describes safe widening alternative

Things are going wrong, despite . . . (2 of 4)

Software libraries are often well documented (emphasis added)

- PPL documentation: Note that in the computation of the H79-widening $P \nabla Q$ of two polyhedra P, Q, **it is required as a precondition that $P \subseteq Q$**
- VPL documentation: Note that [widen p1 p2] relies on **[p1] being included in [p2]**.
- Apron documentation (e.g., C language bindings):

```
ap_abstract1_t ap_abstract1_widening(ap_manager_t* man,
                                     ap_abstract1_t* a1,
                                     ap_abstract1_t* a2)
```

Widening of a1 with a2. **a1 is supposed to be included in a2**.
This applies to all Apron domains (even boxes and octagons!)

Things are going wrong, despite . . . (3 of 4)

Software libraries sometimes assert preconditions

- in debug mode, both Apron and PPL report assertion failure (ELINA and PPLite do not check this precondition)
- AI tools **do not usually run in debug mode** (inefficient)

A side note on verified computations

- this safety issue would have been *a perfect fit for VPL (Verified Polyhedra Library)*
- alas, VPL certificate checks *cannot detect it*

  Note that [widen p1 p2] relies on [p1] being included in [p2].
  *The result includes the two operands, **although no certificate is created**.*

- see [FMP SAS13] and Fouilhé PhD thesis

Things are going wrong, despite . . . (4 of 4)

Was I the first (and only one?) testing an unsafe combination? NO!

Unsafe combinations already used in artifacts/repeatability evaluations
(*unless authors added undocumented code fixes*)

- *Abstract Semantic Differencing for Numerical Programs* by Partush and Yahav [PY SAS13]
  DIZY + **Apron polyhedra**
- *Fast Polyhedra Abstract Domain* by Singh *et al.* [SPV POPL17]:
  SeaHorn + **{ ELINA | PPL | Apron } polyhedra**
- *Fast Numerical Program Analysis with Reinforcement Learning* by Singh *et al.* [SPV CAV18]:
  SeaHorn + **ELINA polyhedra**
- *Learning Fast and Precise Numerical Analysis* by He *et al.* [HSPV PLDI20]:
  SeaHorn + **ELINA polyhedra**

Do things really go wrong?

Are those artifacts computing unsafe results?

- short answer: don't know (it is *undefined behavior!*)
- conjectures on **SeaHorn** + **ELINA polyhedra**:
  - no assertion checking $\Longrightarrow$ fewer crashes
  - delayed widening $\Longrightarrow$ problem mitigation
  - *widening not monotonic* $\Longrightarrow$ more (unsafe) precision in an iterate might result in less (safe) precision in the next iterations
- personal wild guess: these analyses could be **safe by chance**
- recall our question: *can we easily tell when things are going wrong?* NO!

Summary: all combinations are possible

widening / AI engine	safe (no precondition)	**risky** (requires $x \sqsubseteq y$)
safe (join+widen)	PAGAI + octagons	PAGAI + **polyhedra**
risky (only widen)	**IKOS** + octagons	**IKOS** + **polyhedra**

Lesson learned $\Longrightarrow$ humble recommendation

Stick to (i.e., "fix") the classical widening specification [Cousot[2] POPL76].

❶ make your widening safe (for correctness)

❷ make your **AI engine risky** (for efficiency and precision)

widening / AI engine	safe (no precondition)	**risky** (requires $x \sqsubseteq y$)
safe (join+widen)		
risky (only widen)	**PAGAI** + octagons **IKOS** + polyhedra	

Conclusions

- having different widening specifications can cause confusion
- confusion can lead to crashes or more subtle, hidden issues
- **let's "fix" the specification of widenings:**

  risky AI engine + safe widening [Cousot^2 POPL76]

- it has several good properties:
  - it is the default one taught in AI courses and tutorials
  - it avoids all safety issues
  - it can be more precise for non-lattice domains
  - it can be as efficient as (or even more efficient than) alternative specifications

Additional slides

Safety bug example (note: using **PAGAI** + **PPLite Poly**)

Note: PAGAI is a safe AI engine (join + widen);
PAGAI obtained by avoiding the join computation before widening.

```
for (i = 0; i < 3; ++i)
  a[i] = i;
```

iter	x	y	$x \nabla y$	safe?
0	$\bot$	$\{i = 0\}$	$\{i = 0\}$	✓
1	$\{i = 0\}$	$\{i = 1\}$	$\{i = 1\}$	✗
2	$\{i = 1\}$	$\{i = 2\}$	$\{i = 2\}$	✗
3	$\{i = 2\}$	$\bot$	$\bot$	✗

Termination bug example (note: using **PAGAI** + **PPLite Poly**)

Note: PAGAI is a safe AI engine (join + widen);
PAGAI obtained by avoiding the join computation before widening.

```
for (i = 0; i < n; ++i)
  a[i] = i;
```

iter	x	y	$x \nabla y$	safe?
0	$\bot$	$\{i = 0, n \geq 1\}$	$\{i = 0, n \geq 1\}$	✓
1	$\{i = 0, n \geq 1\}$	$\{i = 1, n \geq 2\}$	$\{i = 1\}$	✗
2	$\{i = 1\}$	$\{i = 2, n \geq 3\}$	$\{i = 2\}$	✗
3	$\{i = 2\}$	$\{i = 3, n \geq 4\}$	$\{i = 3\}$	✗
...	...	...	...	✗

From risky to safe widenings

Trivial lifting of the risky implementation

$$x \nabla_{safe} y \stackrel{\text{def}}{=} x \nabla_{risky} (x \sqcup y)$$

***Ad hoc* safe widening implementation on $\mathbb{CP}_n$**

Let $\mathcal{P}_1 \equiv \mathrm{con}(\mathcal{C}_1)$, $\mathcal{P}_2 \equiv \mathrm{gen}(\mathcal{G}_2)$, where $\mathcal{C}_1 = \mathcal{C}_1^{eq} \cup \mathcal{C}_1^{ineq}$ is in minimal form.

$$\mathcal{P}_1 \nabla_{safe} \mathcal{P}_2 \stackrel{\text{def}}{=} \begin{cases} \mathcal{P}_1 \uplus \mathcal{P}_2, & \text{if } (\mathcal{P}_1 = \emptyset) \text{ or } (\mathcal{P}_2 = \emptyset) \text{ or } (g \in \mathcal{G}_2 \text{ violates } c \in \mathcal{C}_1^{eq}) \\ \mathrm{con}(\mathcal{C}_\nabla), & \text{otherwise} \end{cases}$$

where $\mathcal{C}_\nabla \stackrel{\text{def}}{=} \mathcal{C}_1 \setminus \{ c \in \mathcal{C}_1^{ineq} \mid g \in \mathcal{G}_2 \text{ violates } c \}$.

Efficiency comparison for *ad hoc* safe widening on $\mathbb{CP}_n$

Synthetic efficiency comparison (warning: no statistical value)

- 140 randomly generated (5 space dim) closed polyhedra;
  each polyhedron obtained adding 5 random rays to a random bounded box
- 70 calls of widening, in two different modes:
  - PLILP92/6: safe AI engine + **risky** widening
  - POPL76: **risky** AI engine + *ad hoc* safe widening

operations	PLILP92/6	POPL76	ratio
scalar products	808078	131482	0.16
linear combinations	42527	4456	0.10
bitset operations	5822865	106031	0.01
cumulative time	176 ms	38 ms	0.22