Static Program Analysis

Part 5 – widening and narrowing

http://cs.au.dk/~amoeller/spa/

Anders Møller & Michael I. Schwartzbach
Computer Science, Aarhus University

Interval analysis

- Compute upper and lower bounds for integers
- Possible applications:
  - array bounds checking
  - integer representation
  - …
- Lattice of intervals:

$$\mathit{Intervals} = \mathit{lift}(\{\ [\mathtt{l}, \mathtt{h}]\ \mid \mathtt{l},\mathtt{h} \in N \wedge \mathtt{l} \leq \mathtt{h}\ \})$$

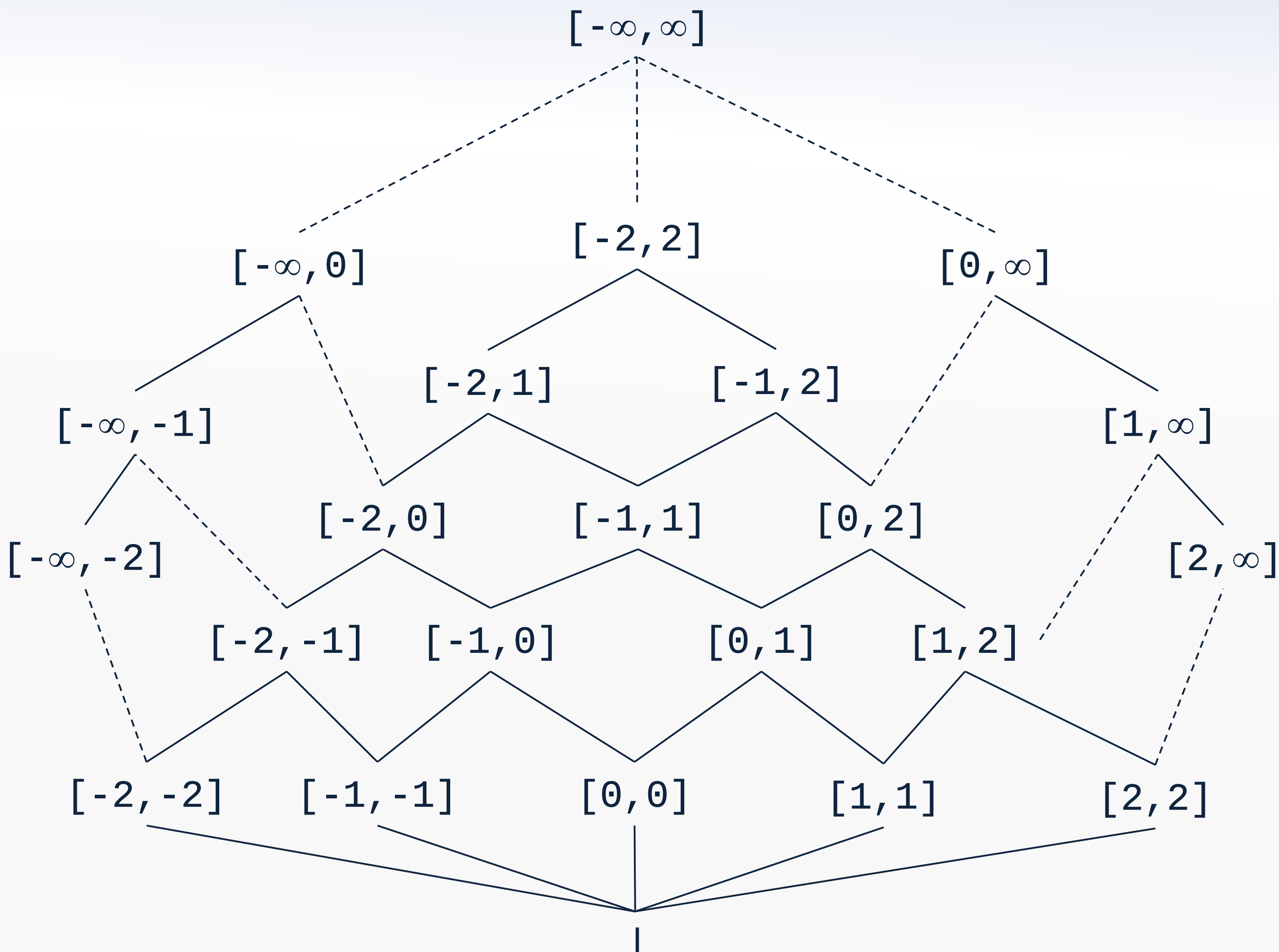
where

$$N = \{-\infty, \ldots, \mathtt{-2}, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots, \infty\}$$

and intervals are ordered by inclusion:

$$[\mathtt{l}_1, \mathtt{h}_1] \sqsubseteq [\mathtt{l}_2, \mathtt{h}_2] \text{ iff } \mathtt{l}_2 \leq \mathtt{l}_1 \wedge \mathtt{h}_1 \leq \mathtt{h}_2$$

The interval lattice



bottom element here interpreted as “not an integer”

Interval analysis lattice

- The total lattice for a program point is

  $$\mathit{Vars} \to \mathit{Intervals}$$

  that provides bounds for each (integer) variable

- If using the worklist solver that initializes the worklist with only the *entry* node, use the lattice *lift*($\mathit{Vars} \to \mathit{Intervals}$)
  - bottom value of *lift*($\mathit{Vars} \to \mathit{Intervals}$) represents "unreachable program point"
  - bottom value of $\mathit{Vars} \to \mathit{Intervals}$ represents "maybe reachable, but all variables are non-integers"

- This lattice has *infinite height*, since the chain

  $$[0,0] \sqsubseteq [0,1] \sqsubseteq [0,2] \sqsubseteq [0,3] \sqsubseteq [0,4] \ldots$$

  occurs in *Intervals*

Interval constraints

- For assignments:

$$[\![x = E]\!] = JOIN(\mathrm{v})[x \rightarrow eval(JOIN(\mathrm{v}), E)]$$

- For all other nodes:

$$[\![\mathrm{v}]\!] = JOIN(\mathrm{v})$$

where $JOIN(\mathrm{v}) = \bigsqcup_{\mathrm{w} \in pred(\mathrm{v})} [\![\mathrm{w}]\!]$

Evaluating intervals

- The *eval* function is an *abstract evaluation*:
  - $eval(\sigma, x) = \sigma(x)$
  - $eval(\sigma, intconst) = [intconst, intconst]$
  - $eval(\sigma, E_1 \text{ op } E_2) = \overline{\text{op}}(eval(\sigma,E_1),eval(\sigma,E_2))$
- Abstract operators:
  - $\overline{\text{op}}([l_1, h_1],[l_2, h_2]) =$

$$\Big[\min_{x\in[l_1,h_1],\, y\in[l_2,h_2]} x \text{ op } y,\ \max_{x\in[l_1,h_1],\, y\in[l_2,h_2]} x \text{ op } y\Big]$$

not trivial to implement!

Fixed-point problems

- The lattice has infinite height,
  so the fixed-point algorithm does not work ☹

- The sequence of approximants

  $f^i(\perp)$ for $i = 0, 1, \ldots$

  is not guaranteed to converge
- (Exercise: give an example of a program where this happens)

- Restricting to 32 bit integers is not a practical solution
- *Widening* gives a useful solution…

Does the least fixed point exist?

- The lattice has infinite height,
  so Kleene's fixed-point theorem does not apply ☹

- **Tarski's fixed-point theorem:**

> In a complete lattice L, every monotone function $f: L \to L$ has a unique least fixed point given by
> $$\mathrm{lfp}(f) = \sqcap\{\, x \in L \mid f(x) \sqsubseteq x \}$$

(Proof?)

Widening

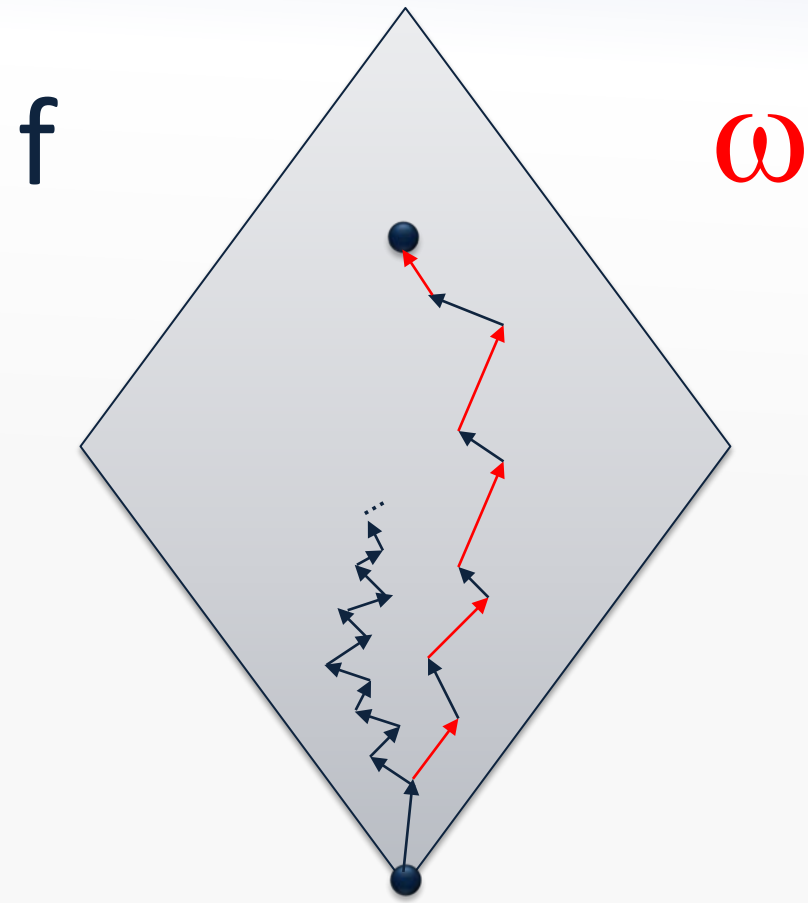
- Introduce a *widening* function $\omega: L \rightarrow L$ so that

  $$(\omega \circ f)^i(\perp) \quad \text{for } i = 0, 1, \ldots$$

  converges on a fixed point that is a safe approximation of each $f^i(\perp)$

- i.e. the function ω coarsens the information

Turbo charging the iterations

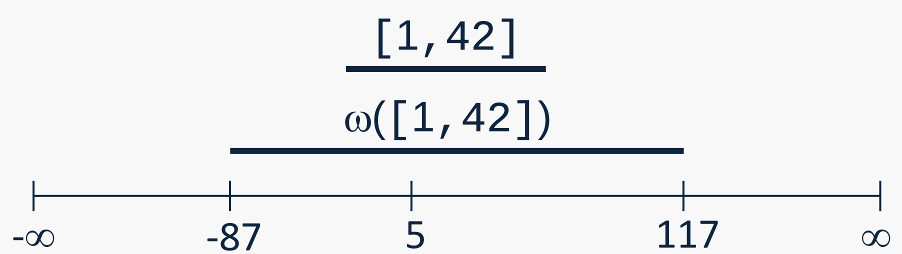


Simple widening for intervals

- The function ω: L $\rightarrow$ L is defined pointwise on
  L = (*Vars* $\rightarrow$ *Intervals*)^n
- Parameterized with a fixed finite set *B*
  - must contain $-\infty$ and ∞ (to retain the $\top$ element)
  - typically seeded with all integer constants occurring in the given program
- Idea: Find the nearest enclosing allowed interval
- On single elements from *Intervals* :

$$\omega([a, b]) = [\ max\{i \in B \mid i \le a\},\ min\{i \in B \mid b \le i\}\ ]$$

$$\omega(\perp) = \perp$$



Divergence in action

```
y = 0;
x = 7;
x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

$[x \rightarrow \bot, y \rightarrow \bot]$
$[x \rightarrow [8,8], y \rightarrow [0,1]]$
$[x \rightarrow [8,8], y \rightarrow [0,2]]$
$[x \rightarrow [8,8], y \rightarrow [0,3]]$
…

Simple widening in action

```
y = 0;
x = 7;
x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

$[x \to \bot, y \to \bot]$
$[x \to [7,\infty], y \to [0,1]]$
$[x \to [7,\infty], y \to [0,7]]$
$[x \to [7,\infty], y \to [0,\infty]]$

$B = \{-\infty, 0, 1, 7, \infty\}$

Correctness of simple widening

- This form of widening works when:
  - ω is an *extensive* and *monotone* function, and
  - the sub-lattice $\omega(L)$ has *finite height*

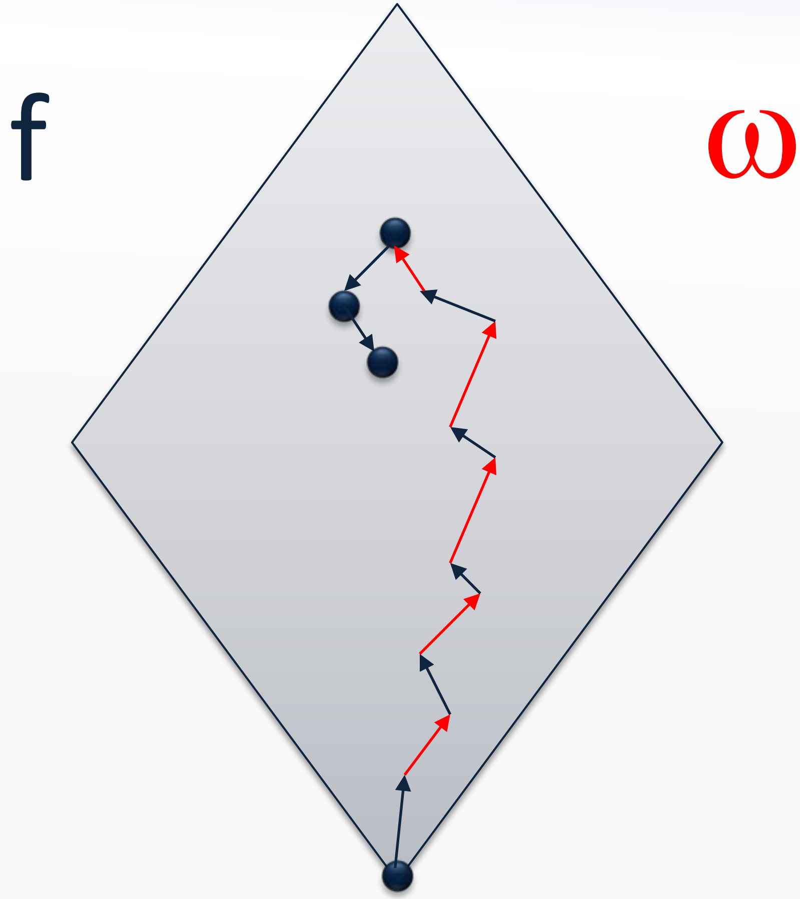
- $\omega \circ f$ is monotone and $\omega(L)$ has finite height, so $(\omega \circ f)^i(\bot)$ for $i = 0, 1, \ldots$ converges
- Let $f_\omega = (\omega \circ f)^k(\bot)$ where $(\omega \circ f)^k(\bot) = (\omega \circ f)^{k+1}(\bot)$
- $lfp(f) \sqsubseteq f_\omega$ follows from Tarski's fixed-point theorem, i.e., f_ω is a safe approximation of $lfp(f)$

Narrowing

- Widening generally shoots over the target
- *Narrowing* may improve the result by applying f

- We have $f(f_\omega) \sqsubseteq f_\omega$ so applying f again may improve the result!
- And we also have $lfp(f) \sqsubseteq f(f_\omega)$ so it remains safe!
- This can be iterated arbitrarily many times
  - may diverge, but safe to stop anytime

Backing up



Narrowing in action

```
y = 0;
x = 7;
x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

$[x \to \perp, y \to \perp]$
$[x \to [7,\infty], y \to [0,1]]$
$[x \to [7,\infty], y \to [0,7]]$
$[x \to [7,\infty], y \to [0,\infty]]$
…
$[x \to [8,8], y \to [0,\infty]]$

$B = \{-\infty, 0, 1, 7, \infty\}$

Correctness of (repeated) narrowing

Claim: $\mathrm{lfp}(f) \sqsubseteq \ldots \sqsubseteq f^i(f_\omega) \sqsubseteq \ldots \sqsubseteq f(f_\omega) \sqsubseteq f_\omega$

- $f(f_\omega) \sqsubseteq \omega(f(f_\omega)) = (\omega \circ f)(f_\omega) = f_\omega$ since ω is extensive
  - by monotonicity of f and induction we also have, for all i:

    $f^{i+1}(f_\omega) \sqsubseteq f^i(f_\omega) \sqsubseteq f_\omega$
  - i.e. $f^{i+1}(f_\omega)$ is at least as precise as $f^i(f_\omega)$
- $f(f_\omega) \sqsubseteq f_\omega$ so $f(f(f_\omega)) \sqsubseteq f(f_\omega)$ by monotonicity of f, hence $\mathrm{lfp}(f) \sqsubseteq f(f_\omega)$ by Tarski's fixed-point theorem
  - by induction we also have, for all i:

    $\mathrm{lfp}(f) \sqsubseteq f^i(f_\omega)$
  - i.e. $f^i(f_\omega)$ is a safe approximation of lfp(f)

Some observations

- The simple notion of widening is a bit naive…
- Widening happens at *every* interval and at *every* node

- There's no need to widen intervals that are not "unstable"
- There's no need to widen if there are no "cycles" in the dataflow

More powerful widening

- A *widening* is a function $\nabla: L \times L \rightarrow L$ that is extensive in both arguments and satisfies the following property:
  for all increasing chains $z_0 \sqsubseteq z_1 \sqsubseteq \ldots$,
  the sequence $y_0 = z_0, \ldots, y_{i+1} = y_i \nabla z_{i+1}, \ldots$ converges
  (i.e. stabilizes after a finite number of steps)

- Now replace the basic fixed point solver by computing $x_0 = \bot$ and $x_{i+1} = x_i \nabla f(x_i)$ until convergence

- Theorem: $x_{k+1} = x_k$ and $lfp(f) \sqsubseteq x_k$ for some k

(Proof: similar to the correctness proof for simple widening)

More powerful widening for interval analysis

Extrapolates unstable bounds to *B*:

$\bot \nabla y = y$

$x \nabla \bot = x$

$[a_1, b_1] \nabla [a_2, b_2] =$

$[\text{if } a_1 \leq a_2 \text{ then } a_1 \text{ else } \mathit{max}\{i \in B \mid i \leq a_2\},$

$\text{if } b_2 \leq b_1 \text{ then } b_1 \text{ else } \mathit{min}\{i \in B \mid b_2 \leq i\}]$

The ∇ operator on L is then defined pointwise down to individual intervals

For the small example program, we get the same result as with simple widening plus narrowing (but now without using narrowing)

Yet another improvement

- Divergence (e.g. in the interval analysis without widening) can only appear in presence of recursive dataflow constraint
- Sufficient to "break the cycles", perform widening only at, for example, loop heads in the CFG

Choosing the set *B*

- Defining the widening function based on constants occurring in the given program may not work well

```
f(x) { // "McCarthy's 91 function"
  var r;
  if (x > 100) {
    r = x - 10;
  } else {
    r = f(f(x + 11));
  }
  return r;
}
```

https://en.wikipedia.org/wiki/McCarthy_91_function

- (This example requires interprocedural and control-sensitive analysis)