Lecture 7: Widening Operators and Collecting Semantics for Dataflow Analysis

17-355/17-655/17-819: Program Analysis

Rohan Padhye and Jonathan Aldrich

March 2, 2021

* Course materials developed with Claire Le Goues

Motivation: Interval Analysis

- Goal: track the range of integers each variable could have
  - Generalizes constant propagation to ranges
  - Use case?

Motivation: Interval Analysis

- Goal: track the range of integers each variable could have
  - Generalizes constant propagation to ranges
  - Use case: array bounds checking

$$
\begin{array}{rcl}
L & = & \mathbb{Z}_\infty \times \mathbb{Z}_\infty \qquad \text{where } \mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, \infty\} \\
[l_1, h_1] \sqsubseteq [l_2, h_2] & \textit{iff} & \\
[l_1, h_1] \sqcup [l_2, h_2] & = & \\
\top & = & \\
\bot & = & \\
\sigma_0 & = & \\
\alpha(x) & = & [x, x]
\end{array}
$$

$\sigma \in \mathbf{Var} \to L$

Note: $\leq$ and *max* are extended to handle ∞

Motivation: Interval Analysis

- Goal: track the range of integers each variable could have
  - Generalizes constant propagation to ranges
  - Use case: array bounds checking

$$
\begin{array}{rcl}
L & = & \mathbb{Z}_\infty \times \mathbb{Z}_\infty \qquad \text{where } \mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, \infty\} \\
[l_1, h_1] \sqsubseteq [l_2, h_2] & \mathit{iff} & l_2 \leqslant_\infty l_1 \wedge h_1 \leqslant_\infty h_2 \\
[l_1, h_1] \sqcup [l_2, h_2] & = & [min_\infty(l_1, l_2), max_\infty(h_1, h_2)] \\
\top & = & [-\infty, \infty] \\
\bot & = & [\infty, -\infty] \\
\sigma_0 & = & \top \\
\alpha(x) & = & [x, x]
\end{array}
$$

$$\sigma \in \mathbf{Var} \to L$$

Note: $\leq$ and max are extended to handle ∞

Interval Analysis Flow Functions

$$f_I[\![x := y + z]\!](\sigma) \quad =$$

Interval Analysis Flow Functions

$$
\begin{array}{lll}
f_I[\![x := y + z]\!](\sigma) & = \sigma[x \mapsto [l, h]] & \text{where } l = \sigma(y).low +_\infty \sigma(z).low \\
 & & \text{and } h = \sigma(y).high +_\infty \sigma(z).high \\
f_I[\![x := y + z]\!](\sigma) & = \sigma & \text{where } \sigma(y) = \bot \vee \sigma(z) = \bot
\end{array}
$$

How would we use this to check array bounds?

What is the height of the Interval Analysis lattice?

Applying Interval Analysis to a Program

$$\begin{array}{ll} 1: & x := 0 \\ 2: & \text{if } x = y \text{ goto } 5 \\ 3: & x := x + 1 \\ 4: & \text{goto } 2 \\ 5: & y := 0 \end{array}$$

stmt	worklist	x	y
0	1,2,3,4,5	$\top$	$\top$
1	2,3,4,5		
2	3,4,5		
3	4,5		
4	2,5		

Applying Interval Analysis to a Program

$$
\begin{aligned}
1:&\quad x := 0\\
2:&\quad \text{if } x = y \text{ goto } 5\\
3:&\quad x := x + 1\\
4:&\quad \text{goto } 2\\
5:&\quad y := 0
\end{aligned}
$$

stmt	worklist	x	y
0	1,2,3,4,5	$\top$	$\top$
1	2,3,4,5	[0,0]	$\top$
2	3,4,5	[0,0]	$\top$
3	4,5	[1,1]	$\top$
4	2,5	[1,1]	$\top$
2	3,5	[0,1]	$\top$
3	4,5	[1,2]	$\top$
4	2,5	[1,2]	$\top$
2	3,5	[0,2]	$\top$
3	4,5	[1,3]	$\top$
4	2,5	[1,3]	$\top$
2	3,5	[0,3]	$\top$
...			

The Widening Operator

- Purpose: compress infinite ascending chains to finite length
- Compares new lattice element to previous one
  - If the new one is higher, the widening operator may skip upwards in the lattice
  - We ensure there are a finite number of such "skips" possible

$$W(\perp, l_{current}) =$$

$$W([l_1, h_1], [l_2, h_2]) =$$

The Widening Operator

- Purpose: compress infinite ascending chains to finite length
- Compares new lattice element to previous one
  - If the new one is higher, the widening operator may skip upwards in the lattice
  - We ensure there are a finite number of such “skips” possible

$$
\begin{array}{ll}
W(\bot, l_{current}) & = l_{current} \\
 & \\
W([l_1, h_1], [l_2, h_2]) & = [min_W(l_1, l_2), max_W(h_1, h_2)] \\
 & \quad \text{where } min_W(l_1, l_2) = l_1 \qquad \text{if } l_1 \leqslant l_2 \\
 & \quad \text{and } min_W(l_1, l_2) = -\infty \qquad \text{otherwise} \\
 & \quad \text{where } max_W(h_1, h_2) = h_1 \qquad \text{if } h_1 \geqslant h_2 \\
 & \quad \text{and } max_W(h_1, h_2) = \infty \qquad \text{otherwise}
\end{array}
$$

Applying Interval Analysis again, with widening

$$
\begin{array}{ll}
1: & x := 0 \\
2: & \text{if } x = y \text{ goto } 5 \\
3: & x := x + 1 \\
4: & \text{goto } 2 \\
5: & y := 0
\end{array}
$$

stmt	worklist	x	y
0	1,2,3,4,5	$\top$	$\top$
1	2,3,4,5		
2	3,4,5		
3	4,5		
4	2,5		
2	3,5		
3	4,5		
4	2,5		
2	5		
5	$\emptyset$		

$$W(\bot, l_{current}) = l_{current}$$

$$W([l_1, h_1], [l_2, h_2]) = [min_W(l_1, l_2), max_W(h_1, h_2)]$$

Applying Interval Analysis again, with widening

$$\begin{array}{ll} 1: & x := 0 \\ 2: & \text{if } x = y \text{ goto } 5 \\ 3: & x := x + 1 \\ 4: & \text{goto } 2 \\ 5: & y := 0 \end{array}$$

stmt	worklist	x	y
0	1,2,3,4,5	$\top$	$\top$
1	2,3,4,5	[0,0]	$\top$
2	3,4,5	[0,0]	$\top$
3	4,5	[1,1]	$\top$
4	2,5	[1,1]	$\top$
2	3,5	[0,∞]	$\top$
3	4,5	[1,∞]	$\top$
4	2,5	[1,∞]	$\top$
2	5	[0,∞]	$\top$
5	$\emptyset$	[0,∞]	[0,0]

$$W(\bot, l_{current}) = l_{current}$$

$$W([l_1, h_1], [l_2, h_2]) = [min_W(l_1, l_2), max_W(h_1, h_2)]$$

Properties of Widening Operators

- Accepts two lattice elements: previous and current
- Returns a lattice value to be used in place of current
- Must return an upper bound of its operands (for monotonicity)
- When applied to an ascending chain, resulting chain must be of finite height
  - New chain l_i^W defined as:

$$l_0^W = l_0$$

$$\forall i > 0 : l_i^W = W(l_{i-1}^W, l_i)$$

Applying the widening operator

- Safe to apply anywhere (the analysis just becomes more conservative)
- Useful to apply at the heads of loops

Widening operators based on program constants

- Heuristic idea: if program has the constant 10, widen to 10 before widening to ∞
  - Maybe 10 is a loop bound!
  - Ascending chain becomes $\perp, [0,0], [0,10]$

$$
\begin{aligned}
W(\perp, l_{current}) &= l_{current} \\
W([l_1,h_1],[l_2,h_2]) &= [min_K(l_1,l_2), max_K(h_1,h_2)] \\
&\quad \text{where } min_K(l_1,l_2) = l_1 && \text{if } l_1 \leqslant l_2 \\
&\quad \text{and } min_K(l_1,l_2) = max(\{k \in K \mid k \leqslant l_2\}) && \text{otherwise} \\
&\quad \text{where } max_K(h_1,h_2) = h_1 && \text{if } h_1 \geqslant h_2 \\
&\quad \text{and } max_K(h_1,h_2) = min(\{k \in K \mid k \geqslant h_2\} && \text{otherwise}
\end{aligned}
$$

Exercise: Try out the constant-based widening operator

What is the computed lattice value after line 5?

$$
\begin{array}{ll}
1: & x := 0 \\
2: & y := 1 \\
3: & \text{if } x = 10 \text{ goto } 7 \\
4: & x := x + 1 \\
5: & y := y - 1 \\
6: & \text{goto } 3 \\
7: & \text{goto } 7
\end{array}
$$

$$
\begin{array}{lll}
W(\bot, l_{current}) & = l_{current} & \\
W([l_1, h_1], [l_2, h_2]) & = [min_K(l_1, l_2), max_K(h_1, h_2)] & \\
 & \text{where } min_K(l_1, l_2) = l_1 & \text{if } l_1 \leqslant l_2 \\
 & \text{and } min_K(l_1, l_2) = max(\{k \in K | k \leqslant l_2\}) & \text{otherwise} \\
 & \text{where } max_K(h_1, h_2) = h_1 & \text{if } h_1 \geqslant h_2 \\
 & \text{and } max_K(h_1, h_2) = min(\{k \in K | k \geqslant h_2\} & \text{otherwise}
\end{array}
$$

The program has constants
K = { -1, 0, 1, 10 }

Try out the constant-based widening operator

$$
\begin{array}{ll}
1: & x := 0 \\
2: & y := 1 \\
3: & \text{if } x = 10 \text{ goto } 7 \\
4: & x := x + 1 \\
5: & y := y - 1 \\
6: & \text{goto } 3 \\
7: & \text{goto } 7
\end{array}
$$

The program has constants
-1, 0, 1, and 10

stmt	worklist	x	y
0	1,2,3,4,5,6,7	$\top$	$\top$
1	2,3,4,5,6,7		
2	3,4,5,6,7		
3	4,5,6,7		
4	5,6,7		
5	6,7		
6	3,7		
3	4,7		
4	5,7		
5	6,7		
6	3,7		
3	4,7		
4	5,7		
5	6,7		
6	3,7		
3	7		
7	$\emptyset$		

Try out the constant-based widening operator

$1: \quad x := 0$
$2: \quad y := 1$
$3: \quad \text{if } x = 10 \text{ goto } 7$
$4: \quad x := x + 1$
$5: \quad y := y - 1$
$6: \quad \text{goto } 3$
$7: \quad \text{goto } 7$

The program has constants
-1, 0, 1, and 10

stmt	worklist	x	y
0	1,2,3,4,5,6,7	$\top$	$\top$
1	2,3,4,5,6,7	[0,0]	$\top$
2	3,4,5,6,7	[0,0]	$[1,1]$
3	4,5,6,7	$[0,0]_F, \bot_T$	$[1,1]$
4	5,6,7	[1,1]	$[1,1]$
5	6,7	[1,1]	$[0,0]$
6	3,7	[1,1]	$[0,0]$
3	4,7	$[0,1]_F, \bot_T$	$[0,1]$
4	5,7	[1,2]	$[0,1]$
5	6,7	[1,2]	$[-1,0]$
6	3,7	[1,2]	$[-1,0]$
3	4,7	$[0,9]_F, [10,10]_T$	$[-\infty,1]$
4	5,7	[1,10]	$[-\infty,1]$
5	6,7	[1,10]	$[-\infty,0]$
6	3,7	[1,10]	$[-\infty,0]$
3	7	$[0,9]_F, [10,10]_T$	$[-\infty,1]$
7	$\varnothing$	[10,10]	$[-\infty,1]$

Collecting Semantics

- Motivation: how would we prove reaching definitions correct?
- Usual approach: compare dataflow lattice elements with actual execution values
  - But “actual execution” doesn’t track which definitions reach a program point!
- Solution: augment semantics with relevant information

Collecting semantics for Reaching Definitions

- Extended version of environment $E_{RD} \in Var \to \mathbb{Z} \times \mathbb{N}$
  - $\mathbb{N}$ represents the line number where the variable was last defined

$$\frac{P(n) = x := m}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto m, \underline{n}], n+1 \rangle} \textit{ step-const}$$

$$\frac{P(n) = x := y}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto E(y), \underline{n}], n+1 \rangle} \textit{ step-copy}$$

$$\frac{P(n) = x := y \; op \; z \quad E(y) \; \mathbf{op} \; E(z) = m}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto m, \underline{n}], n+1 \rangle} \textit{ step-arith}$$

Abstraction Function for Reaching Definitions

$$\alpha_{RD}(E_{RD}, n) = \{x_m \mid \exists x \in domain(E_{RD}) \text{ such that } E_{RD}(x) = i, m\}$$

Collecting Semantics for Live Variables

- Tricky: programs execute forwards but live variables requires backwards reasoning!
- Solution: consider the set of traces generated by a program
- Analyze the traces to determine the set of live variables
- Correctness criterion: the analysis computes a superset of the variables that are actually live in any trace