Abstract interpretation part 2: more of the same, plus widening

Claire Le Goues

15-819O: Program Analysis

Correctness holds when:

- The abstract domain lattice has finite height.
- The flow functions are monotonic.
- The abstraction function is correct.
  - Easy enough for zero analysis, at least.
- The flow functions are locally sound.
  - Explicit link to semantics!

Collecting Semantics

- Any state σ has type Var $\rightarrow$ Z, varies from program point to program point.
- Properly define program points as a set of **labels**
  - Now, we are answering questions about properties with respect to program points (e.g., is x always positive at label i?)
- To answer these questions define *contexts*:

$$C \in \text{Contexts. } C \text{ has type Labels} \rightarrow P(\Sigma)$$

  - For each label i, C(i) = all possible states σ at label i
- This is called the *collecting semantics* of the program
  - Records (super-)set of all possible traces that can reach a program point l
  - This is basically what model checkers approximate!

Back to Abstract Interpretation

- Pick a complete lattice A (abstractions for $\mathcal{P}(\Sigma)$)
  - Along with a monotonic abstraction $\alpha : \mathcal{P}(\Sigma) \rightarrow A$
  - Alternatively, pick $\beta : \Sigma \rightarrow A$
  - This uniquely defines its Galois connection γ
- Take the relations between C_i and move them to the abstract domain:

$$a : \text{Label} \rightarrow A$$

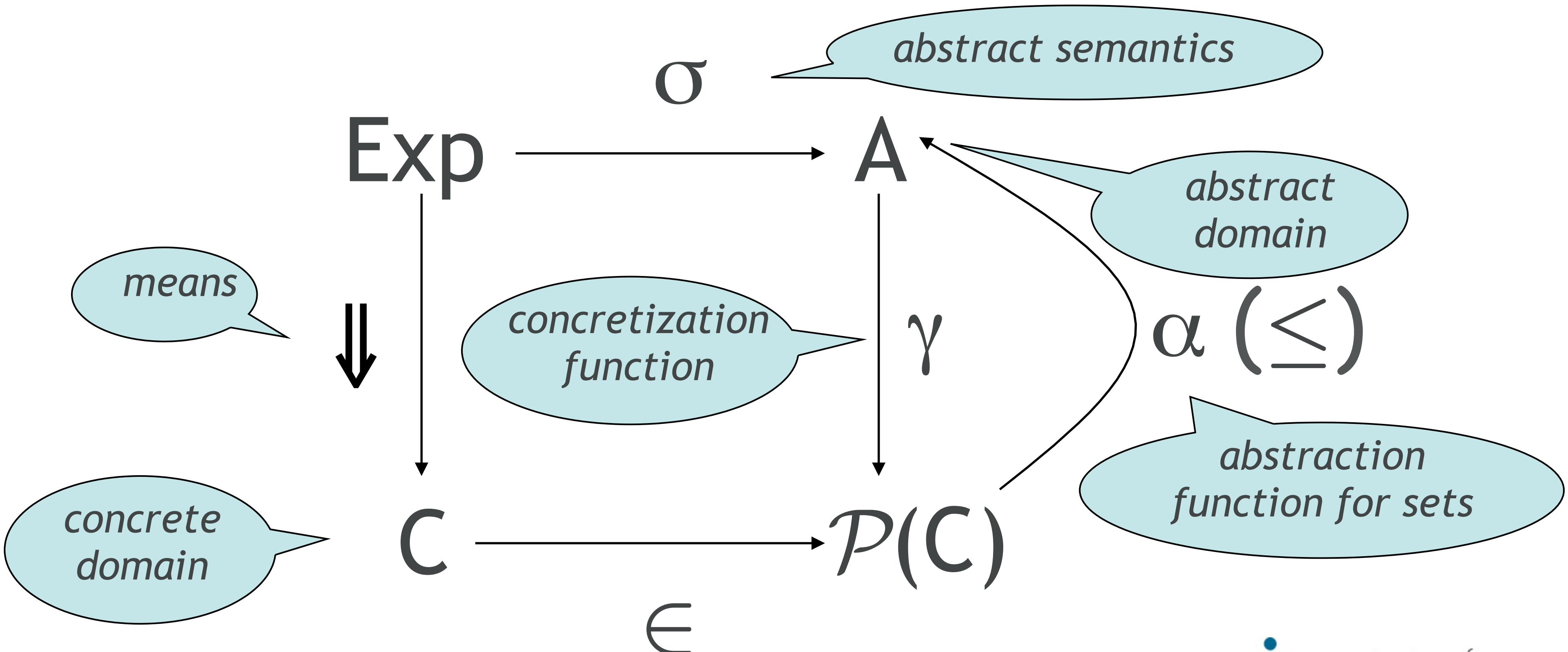
- Assignment

Concrete: $C_j = \{\sigma[x := n] \mid \sigma \in C_i \wedge e \Downarrow \sigma = n\}$

Abstract: $a_j = \alpha\, \{\sigma[x := n] \mid \sigma \in \gamma(a_i) \wedge e \Downarrow \sigma = n\}$

Correctness Condition

- In general, abstract interpretation satisfies the following (amazingly common) diagram



Other Abstract Domains

- Linear relationships between variables
  - A convex polyhedron is a subset of $\mathbb{Z}^k$ whose elements satisfy a number of inequalities:

$$a_1x_1 + a_2x_2 + \ldots + a_kx_k \geq c_i$$

  - This is a complete lattice; linear programming methods compute lubs
- Linear relationships with at most two variables
  - Convex polyhedra but with ≤ 2 variables per constraint
  - Octagons ($x + y \geq c$) have efficient algorithms
- Modulus constraints (e.g. even and odd)

Abstract Chatter

- AI, Dataflow and Software Model Checking
  - The big three (aside from flow-insensitive type systems) for program analyses
- Are in fact quite related:
  - David Schmidt. *Data flow analysis is model checking of abstract interpretation*. POPL '98.
- AI is usually flow-sensitive (per-label answer)
- AI can be path-sensitive (if your abstract domain includes $\vee$, for example), which is just where model checking uses BDD's
- Metal, SLAM, ESP, … can all be viewed as AI

Abstract Interpretation Conclusions

- AI is a very powerful technique that underlies a large number of program analyses
  - Including Dataflow Analysis and Model Checking
- AI can also be applied to functional and logic programming languages
- There are a few success stories
  - Strictness analysis for lazy functional languages
  - PolySpace for linear constraints
- In most other cases however AI is still slow
- When the lattices have infinite height and widening heuristics are used the result becomes unpredictable

Termination holds when:

- The abstract domain has finite height
  - We've stuck to domains for which this is trivially true so far.
- The flow functions are monotonic
  - We proved this just by looking at the definition of the partial order over the abstract state.

Interval analysis

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

Flow function

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

$$f_I[\![x := y + z]\!](\sigma) = \sigma$$
$$\text{where } \sigma(y) = \bot \vee \sigma(z) = \bot$$

No loops.

```
1.x : = 0
2.if x = y goto 5
3.x := x + 1
4.if x = y goto 5
5.y:= 0
```

Loops?

```
1. x : = 0
2. if x = y goto 5
3. x := x + 1
4. goto 2
5. y:= 0
```

```
1. y := x
2. z := 1
3. while [ y > 1 ] do
4. ([z := z * y] ;
5. [y = y-1])
6. y := 6
```

Example of Non-Termination

- The analysis never terminates, or terminates very late if the loop bound is known statically
- It is time to approximate even more: widening
- We redefine the join (lub) operator of the lattice to ensure that from [1..1] upon union with [2..2] the result is [1..+∞) and not [1..2]
- Now the sequence of states is
  - [1..1], [1, +∞), [1, +∞), Done (no more infinite chains)

Formal Definition of Widening
(Cousot 16.399 “Abstract Interpretation”, 2005)

- A widening $\triangledown : (P \times P) \to P$ on a poset $\langle P, \sqsubseteq \rangle$ satisfies:
  - $\forall\, x, y \in P .\quad x \sqsubseteq (x \triangledown y) \quad \wedge \quad y \sqsubseteq (x \triangledown y)$
  - For all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$ the increasing chain $y^0 =^{def} x^0, \ldots, y^{n+1} =^{def} y^n \triangledown x^{n+1}, \ldots$ is not strictly increasing.
- Two different main uses:
  - Approximate missing lubs. *(Not for us.)*
  - Convergence acceleration. *(This is the real use.)*
    - A widening operator can be used to effectively compute an upper approximation of the least fixpoint of $F \in L \triangledown L$ starting from below when L is computer-representable but does not satisfy the ascending chain condition.

Formally…

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

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

where $min_W(l_1, l_2) = l_1$ if $l_1 \leq l_2$
and $min_W(l_1, l_2) = -\infty$ otherwise

where $max_W(h_1, h_2) = h_1$ if $h_1 \geq h_2$
and $max_W(h_1, h_2) = \infty$ otherwise

institute for SOFTWARE RESEARCH

Properties: 1/2

- Must return an upper bound of operands.
  - Why?

$$\forall\, l_{previous}, l_{current} : \; l_{previous} \sqsubseteq W(l_{previous}, l_{current}) \wedge$$
$$l_{current} \sqsubseteq W(l_{previous}, l_{current})$$

Properties: 2/2

- When applied to an ascending chain, the result must be of finite height.
  - Why?

$l_0^W = l_0$ and $\forall\, i > 0 : l_i^W = W(l_{i-1}^W, l_i)$

institute for SOFTWARE RESEARCH

Loss of precision!

- Nice to apply only when necessary, such as only at loop heads (can be inferred).
- Or: use constants in program. If we have a "nearby" constant, like 10, and we see an ascending chain, we can hold off until the top of the chain reaches the constant.
  - $\bot$, [0,0], [0,1], [0,2], [0,3], ... becomes $\bot$, [0,0], [0,10], ...
  - If it keeps ascending, then we widen to infinity.

institute for SOFTWARE RESEARCH

More formally

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

Example

```
1. x := 0
2. y := 1
3. if x=10 goto 7
4. x = x + 1
5. y = y - 1
6. goto 3
7. skip
```

Formal Definition of Widening
(Cousot 16.399 "Abstract Interpretation", 2005)

- A widening $\triangledown : (P \times P) \to P$ on a poset $\langle P, \sqsubseteq \rangle$ satisfies:
  - $\forall\, x, y \in P\ .\quad x \sqsubseteq (x \triangledown y) \quad \wedge \quad y \sqsubseteq (x \triangledown y)$
  - For all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$ the increasing chain $y^0 =^{def} x^0, \ldots, y^{n+1} =^{def} y^n \triangledown x^{n+1}, \ldots$ is not strictly increasing.
- Two different main uses:
  - Approximate missing lubs. *(Not for us.)*
  - Convergence acceleration. *(This is the real use.)*
    - A widening operator can be used to effectively compute an upper approximation of the least fixpoint of $F \in L \triangledown L$ starting from below when L is computer-representable but does not satisfy the ascending chain condition.

institute for SOFTWARE RESEARCH

Formal Widening Example

$[1,1] \triangledown [1,2] = [1,+\infty)$

- Range Analysis on z:

```
L0:    z := 1 ;
L1:    while z<99 do
L2:       z := z+1
L3:    done /* z ≥ 99 */
L4:
```

Original x^i	Widened y^i
$x^{L0}_0 = \bot$	$y^{L0}_0 = \bot$
$x^{L1}_0 = [1,1]$	$y^{L1}_0 = [1,1]$
$x^{L2}_0 = [1,1]$	$y^{L2}_0 = [1,1]$
$x^{L3}_0 = [2,2]$	$y^{L3}_0 = [2,2]$
$x^{L2}_1 = [1,2]$	$y^{L2}_1 = [1,+\infty)$
$x^{L3}_1 = [2,+\infty)$	$y^{L3}_1 = [2,+\infty)$
$x^{L4}_0 = [99,+\infty)$	$y^{L4}_0 = [99,+\infty)$
stable (fewer than 99 iterations!)	

$x^{Li}_j =^{def}$ the jth iterative attempt to compute an abstract value for z at label Li

Recall lub S = [min(S)..max(S)]
lub $\{[2,+\infty),[1,+\infty)\} = \{[1,+\infty)\}$

One Slide Summary

- In **abstract interpretation**, the **abstraction** function β and **concretization** function γ form a **Galois connection**: they are almost inverses.
- To abstract the **state** σ at each program point we use a **collecting semantics** (the abstract domain holds sets of states). This shows the link between abstract interpretation and model checking.
- This will result in recursively-defined equations. We use the **fixed point** theorem to solve them. This shows the link between abstract interpretation and dataflow analysis.
- **Widening** operators help accelerate convergence.

Semantics, redux.

- Imagine we want to add a new for loop statement type to While:
- for (x = e_1, x op_r e_2, x := e_3) do S done

- Let's specify that, in both big- and small-step semantics.