What is an Abstraction?

- A property from some domain

Example Abstraction

Concrete values: sets of integers

Concrete values: sets of integers

Concretization function \( \gamma \) maps each abstract value to concrete values it represents

Abstract function \( \alpha \) maps each concrete set to the best abstract value

\( \gamma \) and \( \alpha \) are monotonic

- Recall: \( f \) is monotonic if \( x \leq y \Rightarrow f(x) \leq f(y) \)
- Also called “order preserving”

\( S \subseteq \gamma(\alpha(S)) \) for any concrete set \( S \)

\( \alpha(\gamma(A)) = A \) for any abstract element \( A \)

Next up: Abstract interpretation in action

- We’ll develop an abstract interpretation of a simple language and prove it correct using these ideas

\( \alpha \) and \( \gamma \) Form a Galois Insertion
Concrete Language

- Concrete domain: integers \( i \)
- Expressions: integers and multiplication
  \[ e ::= i \mid e \cdot e \]

- Standard semantics of the program
  \[ \text{Eval} : e \rightarrow \text{Int} \]
  \[ \text{Eval}(i) = i \]
  \[ \text{Eval}(e_1 \cdot e_2) = \text{Eval}(e_1) \times \text{Eval}(e_2) \]

Abstract Language

- Abstract domain: 0 and signs
  \[ a ::= 0 \mid + \mid - \]
- Programs: abstract values and multiplication
  \[ ae ::= a \mid ae \cdot ae \]

- Semantics of the program
  \[ \text{Define } \text{Acomp} : ae \rightarrow a \]
  \[ \text{Let } \text{Aeval} : e \rightarrow a \text{ be } \text{Acomp} \circ \alpha \]
  - We'll define \( \text{Aeval} \) directly next

Abstraction

- Define an abstract semantics that computes only the sign of the result
  \[
  \begin{array}{c|ccc}
    \text{Eval} & + & 0 & - \\
    \hline
    + & + & 0 & - \\
    0 & 0 & 0 & 0 \\
    - & - & 0 & +
  \end{array}
  \]
  \[ \text{AEval} : e \rightarrow \{+, 0, -\} \]
  \[ \text{AEval}(i) = \begin{cases} + & i > 0 \\ 0 & i = 0 \\ - & i < 0 \end{cases} \]
  \[ \text{AEval}(e_1 \cdot e_2) = \text{AEval}(e_1) \times \text{AEval}(e_2) \]

Soundness

- We can show our abstraction correctly predicts the sign of an expression
- Proof: by structural induction on \( e \)
  \[ \text{Eval}(e) > 0 \iff \text{AEval}(e) = + \]
  \[ \text{Eval}(e) = 0 \iff \text{AEval}(e) = 0 \]
  \[ \text{Eval}(e) < 0 \iff \text{AEval}(e) = - \]

Another Approach to Soundness

- Natural concretization function
  \[
  \gamma(+) = \{ i \mid i > 0 \} \\
  \gamma(0) = \{ 0 \} \\
  \gamma(-) = \{ i \mid i < 0 \} 
  \]
  - Note: This presentation is slightly non-standard
    - Usually defined in terms of execution traces

Soundness (cont’d)

- Our abstraction is sound if
  \[ \text{Eval}(e) \in \gamma(\text{AEval}(e)) \]
- Soundness proof: later
Adding Unary Negation

- $e ::= i \mid e \cdot e \mid -e$
- $\text{Eval}(-e) = -\text{Eval}(e)$
- $\text{AEval}(-e) = -\text{AEval}(e)$
- No problems

Adding Addition

- $e ::= i \mid e \cdot e \mid -e \mid e + e$
- $\text{Eval}(e_1 + e_2) = \text{Eval}(e_1) + \text{Eval}(e_2)$
- $\text{AEval}(e_1 + e_2) = \text{AEval}(e_1) + \text{AEval}(e_2)$

Our abstract domain is not closed under addition.

Solution

- Add an abstract value to represent any integer
- Finding closed domain often key design problem

<table>
<thead>
<tr>
<th>$\gamma(\top)$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$\circ$</td>
<td>$\circ$</td>
</tr>
<tr>
<td>$\bullet$</td>
<td>$\bullet$</td>
</tr>
<tr>
<td>$\bullet$</td>
<td>$\bullet$</td>
</tr>
</tbody>
</table>

Other operations also need to handle $\top$

Adding Integer Division

- What happens when we divide by zero?
  - The result is not an integer (it’s undefined)
  - If we divide each integer in a set by 0, the result is the empty set

<table>
<thead>
<tr>
<th>$\gamma(\bot)$</th>
<th>$\bot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\circ$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bullet$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bullet$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Find the bug: the table is not correct.

Hint: what should be the result of 7 divided by 5?

Two Ways to Lose Information

- OK: Abstraction still precise enough
  - $\text{Eval}(5 \cdot 5 + 6) = 31$
  - $\text{AEval}(5 \cdot 5 + 6) = (+ \cdot +) + + = +$
  - Abstractly, we don’t know which value we computed
  - ...but we don’t care, since we only want the sign

- Not so good: “Don’t know” values
  - $\text{Eval}(1 + 2 + -3) = 0$
  - $\text{AEval}(1 + 2 + -3) = (+ + -) + + - = \bot$
  - We don’t know which value we computed
  - ...and we can’t even figure out its sign

Adding Integer Division (cont’d)

- We need to extend other abstract operations to work on $\bot$
- Every operation involving $\bot$ results in $\bot$
  - All operations are strict in $\bot$

| $\bot \leq a$ | $\bot$ |
| $a \leq \bot$ | $\bot$ |
| $\bot \leq \bot$ | $\bot$ |
| $a \leq \bot$ | $\bot$ |
| $\bot \leq \bot$ | $\bot$ |
The Abstract Domain
• Look, Ma, a lattice!
• We’ve got:
  • A set of elements \{ \bot, +, 0, -, \top \}
  • A relation \leq
    - Reflexive
    - Anti-symmetric
    - Transitive
  • And
    - The least upper bound (\bot, \top) and greatest lower bound (\bot, \top) exists
    - So it’s a lattice

Definition
• An abstract interpretation consists of
  • A concrete domain \( S \) and an abstract domain \( A \)
  • Concretization and abstraction functions that form a
    Galois insertion [of \( A \) into \( S \)]
  • \( A \) (sound) abstract semantic function

• Recall: \( \alpha \) and \( \gamma \) form a Galois insertion if
  • \( \alpha \) and \( \gamma \) are monotone
  • \( S \subseteq \gamma(\alpha(S)) \) or \( \text{id} \leq \gamma \alpha \) for any concrete set \( S \)
  • \( A = \alpha(\gamma(A)) \) or \( \text{id} = \alpha \gamma \) for any abstract element \( A \)

Conditions for Correctness
• We can show that if
  • \( \alpha \) and \( \gamma \) form a Galois insertion
  • And abstract operations \( \mathop{op} \) are locally correct
    - \( \gamma(\mathop{op}(a_1, \ldots, a_n)) \supseteq \mathop{op}(\gamma(a_1), \ldots, \gamma(a_n)) \)
    - Note: We’ve extended \( \mathop{op} \) pointwise to sets
      - i.e., if \( S \) and \( T \) are sets, \( S + T = \{ s+t \mid s \in S, t \in T \} \)
  • Then the abstract interpretation is sound

Abstraction and Concretization
• Concretization function \( \gamma \)
  - \( \gamma(\top) = \text{all integers} \)
  - \( \gamma(+) = \{ i \mid i > 0 \} \)
  - \( \gamma(0) = \{ 0 \} \)
  - \( \gamma(-) = \{ i \mid i < 0 \} \)
  - \( \gamma(\bot) = \emptyset \)

• Abstraction function maps concrete values (sets of integers) to the smallest valid abstract element
  - \( \alpha(S) = \{ i \mid i < 0 \} \) if \( S \subseteq \{ i \mid i < 0 \} \)
  - \( \alpha(S) = \emptyset \) otherwise
  - \( \alpha(S) = \{ i \mid i > 0 \} \) if \( S \subseteq \{ i \mid i > 0 \} \)
  - \( \alpha(S) = \emptyset \) otherwise
  - \( \alpha(S) = \{ 0 \} \) if \( S \subseteq \{ 0 \} \)
  - \( \alpha(S) = \emptyset \) otherwise

Soundness, Again
• Our abstraction is sound if
  • \( \text{Eval}(e) \subseteq \gamma(\text{AEval}(e)) \)
  • Soundness proof: next

Proof: Show \( \text{Eval}(e) \subseteq \gamma(\text{AEval}(e)) \)
• By structural induction on expressions
  • Base cases: an integer \( i \), so \( \text{Eval}(i) = i \)
    - if \( i < 0 \) then \( \gamma(\text{Eval}(i)) = \gamma(-) = \{ j \mid j < 0 \} \)
    - Other cases similar
  • Induction: for any operation
    - \( \text{Eval}(e1 \mathop{op} e2) = \text{Eval}(e1) \mathop{op} \text{Eval}(e2) \)
    - by definition of \( \text{Eval} \)
    - if \( \gamma(\text{Eval}(e1)) \mathop{op} \gamma(\text{Eval}(e2)) \)
    - by induction
    - \( \subseteq \gamma(\text{AEval}(e1) \mathop{op} \text{AEval}(e2)) \)
    - by local correctness of \( \mathop{op} \)
    - \( \subseteq \gamma(\text{AEval}(e1 \mathop{op} e2)) \)
    - by definition of \( \text{AEval} \)
Another Proof of Correctness

• We can define correctness in terms of abstraction rather than concretization
  • Eval(e) ∈ γ(AEval(e)) if α((Eval(e))) ≤ AEval(e)
• Equivalence proof:
  • (⇒) Assume Eval(e) ∈ γ(AEval(e))
  • i.e., (Eval(e)) ⊆ γ(AEval(e))
  • Then α((Eval(e))) ≤ α(γ(AEval(e))) by monotonicity
  • And α((Eval(e))) ≤ AEval(e) since id = αγ

An Alternate Abstract Domain

• That domain wasn’t the only choice, of course

The right domain depends on the problem we’re trying to solve

Relationship to Data Flow Analysis

• Abstract interpretation was invented partially to find a firm semantic foundation for data flow analysis
  • Precise relationship between concrete domain (program executions) and abstract domain (data flow facts)
  • Generic correctness proof
• Caveat: Data flow typically uses meet, abstract interpretation typically uses join

Acceleration: Widening

• Given monotone transfer functions
  • Finite height lattice ⇒ termination
• What if
  • Height is finite but large?
  • Height is infinite
• “Solution”: Widening
  • Every so often, replace A by A">A
  • This is safe (conservative, sound)
  • But apply when? where?

Limitations

• Focus is on correctness
  • Not much insight into efficient algorithms
• Theory is completely general
  • What are good choices for modeling data structures and the heap? Higher-order functions? Objects?
• Forwards vs. backwards distinction
  • Permeates literature on abstract interpretation
  • But theory doesn’t require it
**Conclusions**

- Cousot and Cousot paper(s) seminal work(s)
- The theory of abstract interpretation is often confused with using it to construct tool (e.g., data flow analysis)

**Slogan:**
- Finite lattices + monotonic functions = program analysis