Abstract Interpretation

Based on lectures by David Schmidt and by Alex Aiken

5CMSC 631, Fall 2003
Composing α and γ

Integers
{0, 1, 2, ...} {..., -2, -1, 0}
{0, 2, 4, 6, ...}
{0, even
{0, 42}

Abstraction followed by concretization is sound but imprecise

Example Abstraction

Concrete values: sets of integers
Abstract values

Concretization function γ maps each abstract value to concrete values it represents

Abstraction is Imprecise

Concrete values: sets of integers
Abstract values

Abstraction function α maps each concrete set to the best abstract value

α and γ Form a Galois Insertion

• α and γ are monotonic
  • Recall: f is monotonic if x≤y ⇒ f(x)≤f(y)
  • Also called "order preserving"
• S ∈ γ(α(S)) for any concrete set S
• α(γ(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
Source Language

- Integers and multiplication
  - $e ::= i \mid e \ast e$

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

Abstraction

- Define an abstract semantics that computes only the sign of the result
  - $\text{AEval} : e \rightarrow \{-, 0, +\}$
  - $\text{AEval}(i) = \begin{cases} + & i > 0 \\ 0 & i = 0 \\ - & i < 0 \end{cases}$
  - $\text{AEval}(e_1 \ast e_2) = \text{AEval}(e_1) \times \text{AEval}(e_2)$

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 $\text{AEval}(e) = -$

Soundness (cont’d)

- Our abstraction is sound if
  - $\text{Eval}(e) \in \gamma(\text{AEval}(e))$
- Soundness proof: later

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

Adding Unary Negation

- $e ::= i \mid e \ast e \mid -e$
- $\text{Eval}(-e) = -\text{Eval}(e)$
- $\text{AEval}(e) = -\text{AEval}(e)$
- $\gamma(+) = \{i \mid i > 0\}$
- $\gamma(0) = \{0\}$
- $\gamma(-) = \{i \mid i < 0\}$
- No problems
Adding Addition

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

Adding Addition

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Our abstract domain is not closed under addition

Solution

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

| \[ \gamma(\bot) = \{\text{integers}\} \] |
|---|---|
| \[ + \mid + \mid 0 \mid - \mid \top \] |
| \[ + \mid + \mid 0 \mid - \mid \top \] |
| \[ 0 \mid + \mid 0 \mid - \mid \top \] |
| \[ - \mid - \mid + \mid - \mid \top \] |
| \[ \bot \mid \bot \mid \bot \mid \bot \mid \bot \] |

Two Ways to Lose Information

- OK: Abstraction still precise enough
  - \[ \text{Eval}(5 \cdot 5 + 6) = 31 \]
  - \[ \text{AEval}(5\cdot5 + 6) = (+ \cdot +) + + = + \]
  - 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
  - \[ \text{Eval}(1 + 2 \cdot -3) = 0 \]
  - \[ \text{AEval}(1 + 2 \cdot -3) = (+ \cdot +) \cdot - = + \cdot - = \top \]
  - We also don’t know which value we computed
  - ...and we can’t even figure out its sign

The Abstract Domain

- Look, ma, a lattice!
- We’ve got:
  - A set of elements \( \{\bot, +, 0, -, \top\} \)
  - A relation \( \leq \) that is
    - Reflexive
    - Anti-symmetric
    - Transitive
  - And
    - The least upper bound \( \text{(lub, } \bot) \) and greatest lower bound \( \text{(glb, } \top) \) exists for any pair of elements
    - So it’s a lattice

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 \( \bot \)

Adding Integer Division (cont’d)

| \[ \bot \times a = \bot \] |
| \[ a \times \bot = \bot \] |
| \[ \bot + a = \bot \] |
| \[ a + \bot = \bot \] |
| \[ -\bot = \bot \] |
Abstraction and Concretization

- Concretization function $\gamma$
  \[
  \begin{align*}
  \gamma(\tau) &= \text{all integers} \\
  \gamma(+) &= \{ i | i > 0 \} \\
  \gamma(0) &= \{ 0 \} \\
  \gamma(-) &= \{ i | i < 0 \} \\
  \gamma(\bot) &= \emptyset
  \end{align*}
  \]
- Abstraction function maps concrete values (sets of integers) to smallest valid abstract element
  \[
  \alpha(S) = \begin{cases} \\
  \{ i | i < 0 \} U \{ j | j < 0 \} U \{ + \text{otherwise} \} & \text{if } S . i < 0 \\
  \{ i | i > 0 \} U \{ j | j > 0 \} U \{ - \text{otherwise} \} & \text{if } S . i > 0 \\
  \{ 0 \} U \{ j | j = 0 \} U \{ 0 \text{otherwise} \} & \text{if } S . i = 0 \\
  \end{cases}
  \]

Soundness, Again

\[
\begin{array}{c}
\text{e} \\
\text{Eval} \rightarrow \gamma \rightarrow \alpha \rightarrow \text{AEval}
\end{array}
\]

- Our abstraction is sound if
  - $\text{Eval(e)} \in \gamma(\text{AEval(e)})$
  - Soundness proof: next

Proof: Show $\text{Eval(e)} \in \gamma(\text{AEval(e)})$

- By structural induction on expressions
  - Base cases: an integer i, so $\text{Eval(i)} = i$
    - if $i < 0$ then $\gamma(\text{Eval(i)}) = \gamma(-) = \{ j | j < 0 \}$
    - Other cases similar
  - Induction: for any operation
    \[
    \begin{align*}
    \text{Eval}(e_1 \text{ op } e_2) &= \text{Eval}(e_1) \text{ op } \text{Eval}(e_2) & \text{by definition of Eval} \\
    \in \gamma(\text{AEval}(e_1)) \text{ op } \gamma(\text{AEval}(e_2)) & \text{by induction} \\
    \subseteq \gamma(\text{AEval}(e_1) \text{ op } \text{AEval}(e_2)) & \text{by local correctness of op} \\
    = \gamma(\text{AEval}(e_1 \text{ op } e_2)) & \text{by definition of AEval}
    \end{align*}
    \]

Conditions for Correctness

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

Another Proof of Correctness

- We can define correctness in terms of abstraction rather than concretization
  \[
  \text{Eval(e)} \in \gamma(\text{AEval(e)}) \iff \alpha(\text{Eval(e)}) \leq \text{AEval(e)}
  \]
- Equivalence proof:
  - ($\Rightarrow$) Assume $\text{Eval(e)} \in \gamma(\text{AEval(e)})$
    - $\text{Eval(e)} \subseteq \gamma(\text{AEval(e)})$
    - Then $\alpha(\text{Eval(e)}) \leq \alpha(\gamma(\text{AEval(e)}))$ by monotonicity
  - And $\alpha(\text{Eval(e)}) \leq \text{AEval(e)}$ since $\text{id} = \alpha \cdot \gamma$
Correctness Proof (cont’d)

- Showing
  - Eval(e) ∈ γ(AEval(e)) iff α({Eval(e)})≤AEval(e)
  - (⇒) Assume α({Eval(e)})≤AEval(e)
  - Then γ(α({Eval(e)}))≤γ(AEval(e)) by monotonicity
  - Then {Eval(e)}≤γ(AEval(e)) since id ≤ γα
  - I.e., Eval(e) ∈ γ(AEval(e))

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