CMSC 631 – Program Analysis and Understanding
Fall 2017

Abstract Interpretation

Based on lectures by David Schmidt, Alex Aiken, Tom Ball, and Cousot & Cousot
What is an Abstraction?

• A property from some domain

- Blue (color)
- Planet (classification)
  - $6000..7000$ km (radius)
Example Abstraction

Concrete values: sets of integers

Abstract values

Concretization function $\gamma$ maps each abstract value to concrete values it represents
Abstraction is Imprecise

Concrete values: sets of integers

Abstract values

Abstraction function \( \alpha \) maps each concrete set to the best (least imprecise) abstract value
Composing $\alpha$ and $\gamma$

Concrete values: sets of integers

Abstract values

Abstraction followed by concretization is sound but imprecise
\( \alpha \) and \( \gamma \) Form a Galois Insertion

- \( \alpha \) and \( \gamma \) 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 \( A \)
Plan

• A simple example
  • Approximating the sign of an arithmetic expression

• A more realistic example
  • Approximating sets of integers by ranges in a while language

• Convergence and precision
  • Widening and narrowing
Concrete Language

• Concrete domain:
  ■ Sets of Integers: $2^\mathbb{Z}$

• Expressions: integers and multiplication
  ■ $e ::= i \mid e \cdot e \mid e + e \mid -e$

• Standard semantics of the program
  ■ $\text{Eval} : e \rightarrow \mathbb{Z}$
  ■ $\text{Eval}(i) = i$
  ■ $\text{Eval}(e_1 \cdot e_2) = \text{Eval}(e_1) \times \text{Eval}(e_2)$
  ■ …
Abstract Language

• Abstract domain: 0 and signs and “don’t know”
  - \( a ::= 0 | + | - | T \)

• Programs: abstract values and multiplication
  - \( ae ::= a | ae*ae | ae + ae | -ae \)

• Semantics of the program
  - Define \( \text{Acomp} : e \rightarrow ae \) and \( \text{Abseval} : ae \rightarrow a \)
  - Let \( \text{AEval} : e \rightarrow a \) be \( \text{Abseval} \cdot \text{Acomp} \)
    - Abstract concrete constants, then evaluate abstractly
    - But we define \( \text{AEval} \) directly next
Semantics of abstract expressions

- Define an abstract semantics that computes only the sign of the result

- $\text{AEval} : e \rightarrow \{-, 0, +, T\}$

- $\text{AEval}(i) = \begin{cases} 
  + & i > 0 \\ 
  0 & i = 0 \\ 
  - & i < 0 
\end{cases}$

- $\text{AEval}(e1 * e2) = \text{AEval}(e1) \times \text{AEval}(e2)$

- $\text{AEval}(e1 + e2) = \text{AEval}(e1) \pm \text{AEval}(e2)$

- $\text{AEval}(-e1) = -\text{AEval}(e1)$
Semantics of abstract operations

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

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

<table>
<thead>
<tr>
<th>-</th>
<th>+</th>
<th>0</th>
<th>-</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>+</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Two Ways to Lose Information

• OK: Abstraction still precise enough
  - Eval((5 * 5) + 6) = 31
  - AEval((5*5) + 6) = (+ × +) ± + = +
    - 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
  - Eval((1 + 2) + -3) = 0
  - AEval((1 + 2) + -3) = (+ ± +) ± - = + ± - = ⊤
    - We don’t know which value we computed
    - ...and we can’t even figure out its sign
Adding Integer Division

• What happens when we divide by zero?
  - If we divide any integer in a set by 0, the result is the empty set, since \( x \div 0 \) is undefined.

\[
\gamma(\bot) = \emptyset
\]

<table>
<thead>
<tr>
<th>( \div )</th>
<th>+</th>
<th>0</th>
<th>-</th>
<th>\top</th>
<th>\bot</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>\top</td>
<td>\bot</td>
</tr>
<tr>
<td>0</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
</tr>
<tr>
<td>-</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>\top</td>
<td>\bot</td>
</tr>
<tr>
<td>\top</td>
<td>\top</td>
<td>0</td>
<td>\top</td>
<td>\top</td>
<td>\bot</td>
</tr>
<tr>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
</tr>
</tbody>
</table>

What should the ? be?

*Hint:* what should be the result of 5 divided by 7?
Could gain precision by extending \( a \) with “0 or +” and “0 or −”
The Abstract Domain

- Look, Ma, a lattice!

- We’ve got:
  - A set of elements \( \{ \bot, +, 0, -, \top \} \)
  - A relation \( \subseteq \) that is
    - Reflexive
    - Anti-symmetric
    - Transitive
  - And
    - The least upper bound (lub, \( \cup \)) and greatest lower bound (glb, \( \cap \)) exists for any pair of elements
    - So it’s a lattice
• Concretization function $\gamma$

\[
\begin{align*}
\gamma(\top) & = \text{all integers} \\
\gamma(+) & = \{i \mid i>0\} \\
\gamma(0) & = \{0\} \\
\gamma(-) & = \{i \mid i<0\} \\
\gamma(\bot) & = \emptyset
\end{align*}
\]

• Abstraction function maps concrete values (sets of integers) to the smallest valid abstract element

\[
\alpha(S) = \bigcup \left(\begin{array}{c}
- \exists i \in S. i<0 \\
\bot \text{ otherwise}
\end{array}\right) \sqcup \left(\begin{array}{c}
0 \exists i \in S. i=0 \\
\bot \text{ otherwise}
\end{array}\right) \sqcup \left(\begin{array}{c}
+ \exists i \in S. i>0 \\
\bot \text{ otherwise}
\end{array}\right)
\]

Abstraction and Concretization
• 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 $A$
Our abstraction is sound if

- $\text{Eval}(e) \in \gamma(\text{AEval}(e))$

Soundness proof: next
Conditions for Correctness

• We can show that if
  • $\alpha$ and $\gamma$ form a Galois insertion
  • And abstract operations $\text{op}$ are locally correct
    - $\gamma(\text{op}(a_1, \ldots, a_n)) \supseteq \text{op}(\gamma(a_1), \ldots, \gamma(a_n))$
    - Note: We’ve extended $\text{op}$ 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
By structural induction on expressions

- Base cases: an integer $i$, so $\text{Eval}(i) = i$
  - if $i < 0$ then $\gamma(\text{AEval}(i)) = \gamma(-) = \{j \mid j < 0\}$
  - Other cases similar

- Induction: for any operation
  - $\text{Eval}(e_1 \text{ op } e_2)$
  - $\gamma(\text{AEval}(e_1)) \text{ op } \gamma(\text{AEval}(e_2))$ by induction
  - $\subseteq \gamma(\text{AEval}(e_1 \text{ op } e_2))$ by local correctness of $\text{op}$
  - $= \gamma(\text{AEval}(e_1 \text{ op } e_2))$ by definition of AEval

Proof: Show $\text{Eval}(e) \in \gamma(\text{AEval}(e))$
A Simple Imperative Language

- For arithmetic language
  - Number of operations in Aeval was the same as eval
  - No loops, so convergence is trivial

- Slightly more realistic
  - \( c ::= \text{skip} \mid c; c \mid x := e \mid \text{if0 } e \text{ then } c \text{ else } c \mid \text{while0 } e \text{ do } c \)
  - \( e ::= \ldots \mid e < e \mid \ldots \text{ etc.} \)

- Standard concrete (big step) semantics

- Goal: approximate the \textit{collecting semantics} of \( c \)
Concrete Semantics

- Semantics over states and numbers
  - $\langle e, \sigma \rangle \rightarrow n$
  - $\langle c, \sigma \rangle \rightarrow \sigma$

- Standard rules
  - $\langle x, \sigma \rangle \rightarrow \sigma(x)$
  - $\langle n, \sigma \rangle \rightarrow n$
  - $\langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto n]$
  - $\langle \text{skip}, \sigma \rangle \rightarrow \sigma$
  - $\langle c0, \sigma \rangle \rightarrow \sigma0$
  - $\langle c1, \sigma0 \rangle \rightarrow \sigma1$
  - $\langle c0; c1, \sigma \rangle \rightarrow \sigma1$
Collecting Semantics

• Resembles collecting semantics
  - \( \langle e, S \rangle \rightarrow N \)
  - \( \langle c, S \rangle \rightarrow S' \)
  - Where \( S \) is a set of states, and \( N \) is a set of numbers

• Many rules are straightforward liftings

\[
\langle n, S \rangle \rightarrow \{n\} \\
\langle x, S \rangle \rightarrow \{ n \mid \sigma \in S \land n = \sigma(x) \}
\]
More (straightforward) rules

\[ \langle \text{skip}, S \rangle \rightarrow S \]

\[ \langle c_0, S \rangle \rightarrow S_0 \]
\[ \langle c_1, S_0 \rangle \rightarrow S_1 \]
\[ \langle c_0; c_1, S \rangle \rightarrow S_1 \]

\[ \langle e, S \rangle \rightarrow N \]

\[ S' = \{ \sigma' \mid n \in \mathbb{N} \land (\sigma \in S) \land \sigma' = \sigma[x \mapsto n] \} \]

\[ \langle x:=e, S \rangle \rightarrow S' \]
Conditionals

\[ T = \{ \sigma | \sigma \in S \land \langle e, \{\sigma}\rangle \rightarrow \{0\} \} \]
\[ F = \{ \sigma | \sigma \in S \land \langle e, \{\sigma}\rangle \rightarrow \{n\} \land n \neq 0 \} \]

\[ \langle c_0, T \rangle \rightarrow S_1 \quad \langle c_1, F \rangle \rightarrow S_2 \]

\[ \langle \text{if} 0 \ e \ \text{then} \ c_0 \ \text{else} \ c_1, \ S \rangle \rightarrow S_1 \cup S_2 \]
Loops

\[ T = \{ \sigma \mid \sigma \in S \land \langle e, \{\sigma}\rangle \rightarrow \{0\} \} \]
\[ \langle c, T \rangle \rightarrow S1 \quad S1 \cup S \neq S \]
\[ \langle \text{while}0 \ e \ \text{do} \ c, \ S1 \cup S \rangle \rightarrow S2 \]

\[ \langle \text{while}0 \ e \ \text{do} \ c, \ S \rangle \rightarrow S2 \]

\[ T = \{ \sigma \mid \sigma \in S \land \langle e, \{\sigma}\rangle \rightarrow \{0\} \} \]
\[ \langle c, T \rangle \rightarrow S1 \quad S1 \cup S \neq S \]
\[ F = \{ \sigma \mid \sigma \in S \land \langle e, \{\sigma}\rangle \rightarrow \{n\} \land n \neq 0\} \]
\[ \langle \text{while}0 \ e \ \text{do} \ c, \ S \rangle \rightarrow F \]

Found a fixed point
Work out an example

• Example program $c$ is

  
  while ($x < 4$) { $x := x + 2$ }

• Suppose we compute $\langle c, S \rangle \rightarrow S'$ with $S = \{\sigma\}$

  • If $\sigma$ is $[x \mapsto 0]$ then what is $S'$?

  • What is the fixed point of $S$ at the beginning of the loop?
Soundness of Collecting Semantics

• Theorem: For all \( S, c, \sigma \in S \), and \( \sigma' \)

  \[ \langle c, \sigma \rangle \rightarrow \sigma' \text{ iff } \langle c, S \rangle \rightarrow S' \text{ and } \sigma' \in S' \]

• Thus, collecting semantics directly computes the result of all possible executions of \( c \) in stores \( S \)

  • But it’s uncomputable!

• Goal: perform an abstract interpretation of the collecting semantics

  • Computable, and thus, by soundness, approximates the result of all runs
Abstract domains

- Abstract values, and stores
  - \( N ::= + | 0 | - | T | \bot \)
  - \( S: \text{Var} \to N \)
- \( N \) and \( S \) are lattices
  - Proof as an exercise
- Note that \( S \) treats each variable independently
  - Cannot characterize stores in which the values of variables are always correlated
Command execution

\[ \langle \text{skip}, S \rangle \rightarrow S \]
\[ \langle e, S \rangle \rightarrow N \]
\[ \langle x := e, S \rangle \rightarrow S[x \mapsto N] \]
\[ \langle c0, S \rangle \rightarrow S0 \]
\[ \langle c1, S0 \rangle \rightarrow S1 \]
\[ \langle c0; c1, S \rangle \rightarrow S1 \]

All states such that \( e \) is zero

\[ \langle c0, S | e = 0 \rangle \rightarrow S0 \]
\[ \langle c1, S | e \neq 0 \rangle \rightarrow S1 \]
\[ \langle \text{if} 0 \ e \ \text{then} \ c0 \ \text{else} \ c1, S \rangle \rightarrow S0 \sqcup S1 \]
Loops

\[
\begin{align*}
\langle c, S|e=0 \rangle & \rightarrow S_1 \quad S_1 \sqcup S \neq S \\
\langle \text{while}0 \text{ e do } c, S_1 \sqcup S \rangle & \rightarrow S_2
\end{align*}
\]

\[
\begin{align*}
\langle \text{while}0 \text{ e do } c, S \rangle & \rightarrow S_2
\end{align*}
\]

\[
\begin{align*}
\langle c, S|e=0 \rangle & \rightarrow S_1 \quad S_1 \sqcup S = S \\
F & = S|e\neq0 \\
\langle \text{while}0 \text{ e do } c, S \rangle & \rightarrow F
\end{align*}
\]
Soundness

- Soundness now refers to the collecting semantics, rather than the standard semantics

  If $S = \alpha(S)$ then $\langle c, S \rangle \rightarrow S_2$ implies $\langle c, S \rangle \rightarrow S_2$
  where $\alpha(S_2) \subseteq S_2$

  - Alternatively, that $S_2 \subseteq \gamma(S_2)$
The Intervals Domain

- Abstract domain of integer ranges (for single variable)
  - $A ::= \{[l,u] \mid l \in \mathbb{Z} \cup -\infty \land u \in \mathbb{Z} \cup +\infty \land l \leq u\}$
  - $[l_1, u_1] \sqsubseteq [l_2, u_2] \iff l_2 \leq l_1 \land u_1 \leq u_2$
  - $[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$

- Abstraction function $\alpha : S \rightarrow A$
  \[
  \alpha(S) = [\min(\{v \mid v \in S\}), \max(\{v \mid v \in S\})]
  \]

- Concretization function $\gamma : A \rightarrow S$
  \[
  \gamma([l,u]) = \{n \mid l \leq n \leq u\}
  \]
Galois Insertion?

• Recall:
  - $x \subseteq \gamma(\alpha(x))$
  - $z = \alpha(\gamma(z))$

• Examples:
  - $x = \{-2, 8, -5\}$
    - $\alpha(x) = [-5, 8]$ and $\gamma(\alpha(x)) = \{-5, -4, \ldots, 8\}$
  - $z = [-8, 8]$
    - $\gamma(z) = \{-8, -7, \ldots, 7, 8\}$ and $\alpha(\gamma(z)) = [-8, 8]$
Abstract Interpretation

```
x := 0
while (x <= 100)
  x := x + 2
```

\[
x \mapsto \bot
\]

\[
x \mapsto [0,0] \sqcup [2,2]
\]

\[
x \mapsto [0,2]
\]

\[
x \mapsto [2,2] \sqcup [2,4]
\]
Abstract Interpretation

\[ x \leftarrow \bot \]

\[ x := 0 \]
\[ x \leftarrow [0,100] \sqcup [102,102] \]

\[ \text{while } (x \leq 100) \]
\[ x := x + 2 \]
\[ x \leftarrow [2,100] \sqcup [102,102] \]

\[ x \leftarrow [0,100] \]

\[ x \leftarrow [0,100] \sqcup [102,102] \]
Precision

• Abstract interpretation for loop entry
  - \((x \mapsto [0, 102] \in A)\)
  - \(\gamma([0, 102]) = \{0, 1, 2, \ldots, 102\}\)

• But collecting semantics gives
  - \(\{0, 2, 4, \ldots, 102\}\)
Convergence

• How do we know that we will reach a fixed point?
  ■ We could pick A to be a finite lattice
  ■ Or, A could be an infinite lattice with no infinite ascending chain

• But our choice of A satisfies neither of these conditions

• What about speed of convergence?
  ■ Example took 50 iterations to converge
  ■ Can we do better?
Widening and Narrowing

• Widening guarantees convergence even for infinite lattices
  ■ But loses precision
  ■ Also usually improves rate of convergence even for finite lattices

• Narrowing recovers precision lost by widening
Widening: $\nabla$

- Given a lattice $L$, a widening $\nabla: L \times L \rightarrow L$ requires
  - $\forall x, y \in L. x \sqsubseteq x \nabla y$
  - $\forall x, y \in L. y \sqsubseteq x \nabla y$

For all chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$

is not strictly increasing

- Similar to role of lub $\sqcup$
Example Widening for Intervals

• $\bot \nabla X = X$

• $X \nabla \bot = X$

• $[l_1, u_1] \nabla [l_2, u_2] =$
  
  $[\text{if } l_2 < l_1 \text{ then } -\infty \text{ else } l_1, \text{ if } u_2 > u_1 \text{ then } +\infty \text{ else } u_1]$  

Given a sequence of iterates for a loop

$x^0, x^1, \ldots, x^i, \ldots$

Use widening instead to compute

$y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}$
Widening Example

\[
x := 0
\]

while (x <= 100)

\[
x := x + 2
\]

\[
x^1 \mapsto \bot \quad \triangledown [0,0] = [0,0]
\]

\[
x^2 \mapsto x^1 \quad \triangledown [2,2] = [0, +\infty]
\]

\[
x^2 \mapsto x^1 \quad \triangledown [0, +\infty] \cap [-\infty, 100] = [0, 100]
\]

\[
x^1 \mapsto [2,2]
\]

\[
x^2 \mapsto [2,2] \quad \triangledown [2,102] = [2, +\infty]
\]
Conclusions

- Galois connections with finite lattices or Widening/Narrowing?
  - Typically some combination of the two

- Theory is completely general
  - What are good choices for modeling data structures and the heap? Higher-order functions? Objects?

- Picking the right abstract domains; finding the right widening/narrowing can be tricky
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)
- But there are successful tools:
  - ASTREE has proved the absence of runtime errors in the primary control software of the Airbus A340
  - PolySpace C and Ada verifiers