CMSC 631 – Program Analysis and Understanding
Fall 2006

Abstract Interpretation

Based on lectures by David Schmidt, Alex Aiken, Tom Ball, and Cousot & Cousot

What is an Abstraction?

• A property from some domain

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
CMSC 631

Composing $\alpha$ and $\gamma$

Abstract values

Concrete values: sets of integers

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 $A$
  (Sometimes $\alpha(\gamma(A)) \subseteq A$ --- aka Galois Connection)
- $\forall x \in S, y \in A, \alpha(x) \subseteq y \iff x \subseteq \gamma(y)$
- Exercise: Prove that they are equivalent

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: $\mathbb{Z}^2$

- Expressions: integers and multiplication
  - $e := i | e + e | e * e | -e$

- Standard semantics of the program
  - Eval : $e \rightarrow \mathbb{Z}$
  - Eval(i) = i
  - Eval(e1 + e2) = Eval(e1) * Eval(e2)
  - ...

CMSC 631
Abstract Language

- Abstract domain: 0 and signs and “don’t know”
  - \( a := 0 \mid + \mid - \mid T \)
- Programs: abstract values and multiplication
  - \( ae ::= a \mid ae * ae \mid ae + ae \mid ae - ae \)

Semantics of the program

- Define \( Ac \) : \( ae \rightarrow a \)
- Let \( Ae : e \rightarrow ae Ac * \alpha \)
  - We’ll define \( Ae \) directly next

Semantics of abstract expressions

- Define an abstract semantics that computes only the sign of the result
  - \( Ae : e \rightarrow \{-, 0, +, T\} \)
  - \( Ae(i) = \)
  - \( Ae(e_1 * e_2) = Ae(e_1) \times Ae(e_2) \)
  - \( Ae(e_1 + e_2) = Ae(e_1) + Ae(e_2) \)
  - \( Ae(-e_1) = \) \(\{-, 0, +, T\} \)

Two Ways to Lose Information

- OK: Abstraction still precise enough
  - \( \text{Eval}(5 + 5) + 6 = 31 \)
  - \( \text{AEval}(5^2 + 5) + 6 = (+ \times +) + + + + \)
    - 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
  - \( \text{Eval}((1 + 2) + -3) = 0 \)
  - \( \text{AEval}((1 + 2) + -3) = (+ + +) + + + + + \)
    - 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?
  - 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

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

Find the bug: the table is not correct.
Hint: what should be the result of 7 divided by 5?

The Abstract Domain

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

Abstraction and Concretization

- 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
  \[
  \alpha(S) = \begin{cases} 
    \bot & \text{if } \exists s \in S \text{ and } s < 0 \\
    0 & \text{if } \exists s \in S \text{ and } s > 0 \\
    0 & \text{if } S \text{ is empty} \\
    \bot & \text{otherwise}
  \end{cases}
  \]

Definition

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

- Recall: \(\alpha\) and \(\gamma\) form a Galois insertion if
  - \(\alpha\) and \(\gamma\) are monotone
  - \(S \sqsubseteq \gamma(\alpha(S))\) or \(\id \leq \gamma \alpha\) for any concrete set \(S\)
  - \(A = \alpha(\gamma(A))\) or \(\id = \alpha \gamma\) for any abstract element \(A\)
Soundness, Again

- 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

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 $\gamma(\text{AEval}(i)) = \gamma(i) = \{ j \mid j < 0 \}$
    - Other cases similar
  - Induction: for any operation
    - $\text{Eval}(e_1 \text{ op } e_2)$
      - $\gamma(\text{AEval}(e_1) \text{ op } \text{AEval}(e_2))$ by definition of $\text{AEval}$
    - $\gamma(\text{AEval}(e_1) \text{ op } \text{AEval}(e_2))$ by local correctness of $\text{op}$
    - $\gamma(\text{AEval}(e_1 \text{ op } e_2))$ by definition of $\text{AEval}$

A Simple Imperative Language

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

- Slightly more realistic
  - $e ::= c | x | e := e | \text{while } e \text{ e } e$

- Standard concrete semantics

- Goal: approximate the collecting semantics of $e$
Collecting Semantics

• The union of all possible program executions
• Given a concrete semantics
  • Similar to a big-step semantics: (e,s) \Downarrow (v,s)

\[ \text{State'} := (x \mapsto v), \text{State'} \]

\[ F = \text{State} \rightarrow e \rightarrow (\text{State}, v) \]

\[ F \circ (x := v) = (s[x \mapsto v], v) \]

\[ F \circ x = (s, s(x)) \]

Example of collecting semantics

• Example program
  \[ P: \ x := 0; \text{ while } (x \leq 100) \{ x := x + 2 \} \]

• Concrete domain (of collecting semantics)
  • \( D := 2^{\mathbb{Z}} \rightarrow \mathbb{Z} \) (sets of integers)
  • Make \( D \) a lattice by using \( \subseteq \) as the partial order in the range

For \( P \); the collecting semantics for \( x \) at the entry of the loop is (ignoring the value component)

\[ \{ x \mapsto 0, x \mapsto 2, \ldots, x \mapsto 102 \} \]

Collecting Semantics (contd.)

• Extend concrete semantics to sets of states

\[ G := 2^{\text{State}} \rightarrow e \rightarrow 2^{\text{State} \circ v} \]

\[ G \circ e = (F \circ e | s \in S) \]

But, the collecting semantics are ideal. Cannot usually be computed precisely.

The Intervals Domain

• Abstract domain of integer ranges (for single variable \( x \))

\[ A := \{ [l, u] | l, u \in \mathbb{Z} \cup \{-\infty, \infty\}, l \leq u \} \]

\[ [l_1, u_1] \subseteq [l_2, u_2] \Rightarrow l_1 \leq l_2 \land u_1 \leq u_2 \]

\[ [l_1, u_1] \cup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)] \]

• Abstraction function \( \alpha : D \rightarrow A \)

\[ \alpha (X) = [\min(v | x \mapsto v \in X)), \max(v | x \mapsto v \in X))] \]

• Concretization function \( \gamma : A \rightarrow D \)

\[ \gamma ([l, u]) = \{ x \mapsto i | 1 \leq i \leq u \} \]
Galois Insertion?

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

Example:
- $x = \{-2, 8, -5\}$, $\alpha(x) = [5, 8]$, $\gamma(\alpha(x)) = [-5, -4, \ldots, 8]$
- $y = [-8, 8]$, $\gamma(y) = [-8, -7, \ldots, 7, 8]$, $\alpha(\gamma(y)) = [-8, 8]$

Abstract semantics

$F' ::= A \rightarrow e \rightarrow A$

$F' (x \mapsto [l, u]) (x := x + 2) = (x \mapsto [l+2, u+2])$

...
**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 : \( \triangledown \)**

- Given a lattice \( L \), a widening \( \triangledown : L \times L \rightarrow L \) must satisfy
  - \( \forall x, y \in L : x \triangledown y \)
  - \( \forall x, y \in L : y \triangledown x \)

  For all chains \( x^0 \subseteq x^1 \subseteq \ldots \)

  \( y^0 = x^0, \ldots, y^{i+1} = y^i \triangledown x^{i+1}, \ldots \)

  Is not strictly increasing
**Example Widening for Intervals**

\[ \bot \vee X = X \]
\[ X \vee \bot = X \]
\[ [l_1, u_1] \triangledown [l_2, u_2] = \]
\[ \text{[if } l_2 < l_1 \text{ then } -\infty \text{ else } l_1, \text{ if } u_2 > u_1 \text{ then } +\infty \text{ else } u_1] \]

Given a sequence of iterates for a loop
\[ x^0, x^1, \ldots, x^i, \ldots \]
Use widening instead to compute
\[ y^0 = x^0, \ldots, y^{i+1} = y^i \triangledown x^{i+1} \]

**Narrowing : \( \triangle \)**

- Recover precision lost due to widening
- Given a lattice \( L \), a narrowing \( \triangle : L \times L \rightarrow L \) must satisfy
  - \( \forall x, y \in L \ (y \sqsubseteq x) \Rightarrow (y \sqsubseteq (x \triangle y) \sqsubseteq x) \)
- Given \( x^0 \sqsubseteq x^1 \sqsubseteq \ldots \)
  - \( y^0 = x^0, \ldots, y^{i+1} = y^i \triangle x^{i+1}, \ldots \)
  - Is not strictly decreasing

**Example Narrowing for Intervals**

\[ \bot \triangle X = \bot \]
\[ X \triangle \bot = \bot \]
\[ [l_1, u_1] \triangle [l_2, u_2] = \]
\[ \text{[if } l_1 = -\infty \text{ then } l_2 \text{ else } l_1, \text{ if } u_1 = +\infty \text{ then } u_2 \text{ else } u_1] \]
Narrowing Example

\[
x := 0 \quad x^1 \mapsto \bot \quad \triangledown [0,0] = [0,0] \\
x^2 \mapsto x^1 \quad \triangledown [2,2] = [0,\infty] \\
x^1 \mapsto x^2 \quad \triangledown [2,102] = [0,102] \\
\text{while (} x \leq 100 \text{)} \\
\quad x := x + 2 \\
x^2 \mapsto [2, \infty] \\
x^1 \mapsto [2, \infty] \triangle [2,102] = [2,102]
\]

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
• But can also be used to model many other analysis
  • CFA, type inference etc.

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