Abstract Interpretation, or "Non-Standard Semantics", or "Picking the Right Abstraction"

Meeting 14, CSCI 5535, Spring 2009

Announcements

• Homework 3 is graded
  - Graded out of 19, out of 38 on moodle because of 0.5 point values
• Homework 6 out, due Mon Mar 9
  - Just one short exercise

Announcements: Projects

• Project suggestions posted
• Project proposal due Sun Mar 8
  - "Part of Homework 6"
• Project status update due
  - Sun Mar 29 or Fri Mar 20 or Sun Mar 22?
• Project paper due Wed May 6

The Problem: Static Analysis

• It is extremely useful to predict program behavior statically (= without running the program)
  - Why? What uses?
    • "Don't need to run" → avoid deployment
    • Find bugs before run
  • Don't need test bench
  • Prove correctness = security

The Plan

• We will introduce abstract interpretation by example
• Starting with a miniscule language we will build up to a fairly realistic application
• Along the way we will see most of the ideas and difficulties that arise in a big class of applications
A Tiny Language

- Consider the following language of arithmetic ("shrIMP")
  \[ e ::= n \mid e_1 \cdot e_2 \]
- Denotational semantics of this language
  \[ [n] = n \]
  \[ [e_1 \cdot e_2] = [e_1] \cdot [e_2] \]
- Take deno. sem. as the "ground truth"
- For this language the precise semantics is computable (but in general it's not)

An Abstraction

- Assume that we are interested not in the value of the expression, but only in its sign:
  - positive (+), negative (-), or zero (0)
- We can define an abstract semantics that computes only the sign of the result

\[ \sigma: Exp \rightarrow \{-, 0, +\} \]
\[ \sigma(n) = \text{sign}(n) \]
\[ \sigma(e_1 \cdot e_2) = \sigma(e_1) \otimes \sigma(e_2) \]

“I Saw the Sign”

- Why did we want to compute the sign of an expression?
  - One reason: no one will believe you know abstract interpretation if you haven’t seen the sign thing :-)
- What could we be computing instead?
  - "Take infinite and turn into finite"

Correctness of Sign Abstraction

- Can show that the abstraction is correct in the sense that it predicts the sign

\[ [e] > 0 \iff \sigma(e) = + \]
\[ [e] = 0 \iff \sigma(e) = 0 \]
\[ [e] < 0 \iff \sigma(e) = - \]
- Our semantics is abstract but precise
- Proof is by structural induction on the expression \( e \)
  - Each case repeats similar reasoning

Another View of Soundness

- Associate each concrete value to an abstract value:
  \[ \beta: \mathbb{Z} \rightarrow \{-, 0, +\} \]
- This is called the abstraction function (\( \beta \))
  - This three-element set is the abstract domain
- Also define the concretization function (\( \gamma \)):
  \[ \gamma: \{-, 0, +\} \rightarrow \mathcal{P}(\mathbb{Z}) \]
  \[ \gamma(+) = \{ n \in \mathbb{Z} \mid n > 0 \} \]
  \[ \gamma(0) = \{ 0 \} \]
  \[ \gamma(-) = \{ n \in \mathbb{Z} \mid n < 0 \} \]

Another View of Soundness

- Soundness can be stated succinctly
  \[ \forall e \in \text{Exp.} \exists [e] \in \gamma(\sigma(e)) \]
  (the real value of the expression is among the concrete values represented by the abstract value of the expression)
Another View of Soundness

- Soundness can be stated succinctly
  \[ \forall e \in \text{Exp}. \quad e \in \gamma(\sigma(e)) \]
  (the real value of the expression is among the concrete values represented by the abstract value of the expression)

- Let \( C \) be the concrete domain (e.g. \( \mathbb{Z} \)) and \( A \) be the abstract domain (e.g. \( \{-, 0, +\} \))

- Commutative diagram:

```
     Exp  \sigma  A
    /\     \   |
C  \sigma  \gamma(C)
```

Often, this is called the concrete domain

One-Slide Summary: Abstract Interp

- This is our first example of an abstract interpretation
- We carry out computation in an abstract domain
- The abstract semantics is a sound approximation of the standard semantics
- The concretization and abstraction functions establish the connection between the two domains

Adding Unary Minus and Addition

- We extend the language to
  \[ e ::= n | e_1 \cdot e_2 | - e \]
- We define \( \sigma(- e) = \ominus \sigma(e) \)
Adding Unary Minus and Addition

- We extend the language to
  \[ e ::= n \mid e_1 \cdot e_2 \mid -e \]
- We define \( \sigma(-e) = \ominus \sigma(e) \)

Adding Addition

- The sign values are not closed under addition
- What should be the value of \( + \oplus - \)?
- Start from the soundness condition:
  \( \gamma(+ \oplus -) \supset \{ n_1 + n_2 \mid n_1 > 0, n_2 < 0 \} = \mathbb{Z} \)
- We don't have an abstract value whose concretization includes \( \mathbb{Z} \), so we add one:
  \[ T \] ("top" = "don't know")

Loss of Precision

- Abstract computation may lose information:
  \[ ((1 + 2) + -3) = 0 \]
- But:
  \[ \sigma((1+2) + -3) = \sigma(1 + \sigma(2)) + \sigma(-3) = (+ \oplus +) \oplus - = T \]
  • We lost some precision
  • But this will simplify the computation of the abstract answer in cases when the precise answer is not computable

Adding Division

- Issues?
  - Divide by 0

Adding Division

- Straightforward except for division by 0
  - We say that there is no answer in that case
  - \( \gamma(0 \div 0) = \{ n \mid n = n_1 / 0, n_1 > 0 \} = \emptyset \)
- Introduce \( \perp \) to be the abstraction of the 0
  - We also use the same abstraction for non-termination
  \[ \perp \text{ = "nothing"} \]
  \[ T \text{ = "something unknown"} \]
The Abstract Domain

- Our abstract domain forms a lattice
- A partial order is induced by \( \gamma \)
  - We say that \( a_1 \) is more precise than \( a_2 \)
- Every finite subset has a least-upper bound (lub) and a greatest-lower bound (glb)

Lattice Facts

- A lattice is complete when every subset has a lub and a gub
  - Even infinite subsets!
- Every finite lattice is (trivially) complete
- Every complete lattice is a complete partial order (recall: denotational semantics!)
  - Since a chain is a subset
- Not every CPO is a complete lattice
  - Might not even be a lattice at all

Lattice History

- Early work in denotational semantics used lattices (instead of what?)
  - But only chains need to have lubs
  - And there was no need for \( \top \) and glb
- In abstract interpretation we’ll use \( \top \) to denote “I don’t know”.
  - Corresponds to all values in the concrete domain

From One, Many

- We can start with the abstraction function \( \beta \)
  - \( \beta : C \to A \)
  - (maps a concrete value to the best abstract value)
  - A must be a lattice
- We can derive the concretization function \( \gamma \)
  - \( \gamma : A \to \mathcal{P}(C) \)
  - \( \gamma(a) = \{ x \in C \mid \beta(x) \subseteq a \} \)
- And the abstraction for sets \( \alpha \)
  - \( \alpha : \mathcal{P}(C) \to A \)
  - \( \alpha(S) = \operatorname{lub} \{ \beta(x) \mid x \in S \} \)

Example: With Our Sign Lattice

- Consider our sign lattice
  - \( + \) if \( n > 0 \)
  - \( 0 \) if \( n = 0 \)
  - \( - \) if \( n < 0 \)
- \( \alpha(S) = \operatorname{lub} \{ \beta(x) \mid x \in S \} \)
  - Example: \( \alpha \{ (1, 2) \} = \frac{1}{2} \)
  - \( \alpha(\emptyset) = \emptyset \)
- \( \gamma(a) = \{ n \mid \beta(n) \subseteq a \} \)
  - Example: \( \gamma(+) = \{ n \mid (n) \subseteq + \} \)
  - \( \gamma(0) = \mathbb{Z} \)
  - \( \gamma(-) = \mathbb{Z} \)
Example: With Our Sign Lattice

• Consider our sign lattice
  \[ \beta(n) = \begin{cases} + & \text{if } n > 0 \\ 0 & \text{if } n = 0 \\ - & \text{if } n < 0 \end{cases} \]

• \( \alpha(S) = \text{lub } \{ \beta(n) \mid n \in S \} \)
  - Example:
    \( \alpha \{ 1, 2 \} = \text{lub } \{ + \} = + \)
    \( \alpha \{ 1, 0 \} = \text{lub } \{ +, 0 \} = \top \)
    \( \alpha(\emptyset) = \text{lub } \emptyset = \bot \)

• \( \gamma(a) = \{ n \mid \beta(n) \sqsubseteq a \} \)
  - Example:
    \( \gamma(+) = \{ n \mid \beta(n) \sqsubseteq + \} = \{ n \mid n > 0 \} \)
    \( \gamma(\top) = \{ n \mid \beta(n) \sqsubseteq \top \} = \mathbb{Z} \)
    \( \gamma(\bot) = \{ n \mid \beta(n) \sqsubseteq \bot \} = \emptyset \)

Galois Connections

• We can show that
  - \( \gamma \) and \( \alpha \) are monotonic (with \( \subseteq \) ordering on \( P(C) \))
  - \( \alpha(a) = a \) for all \( a \in A \)
  - \( \gamma(\alpha(S)) \subseteq S \) for all \( S \in P(C) \)

• Such a pair of functions is called a Galois connection
  - Between the lattices \( A \) and \( P(C) \)

Correctness Condition

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

Three Little Correctness Conditions

• Three conditions define a correct abstract interpretation
  1. \( \alpha \) and \( \gamma \) are monotonic
  2. \( \alpha \) and \( \gamma \) form a Galois connection
     \( \text{"} \alpha \text{ and } \gamma \text{ are almost inverses"} \)
  3. Abstraction of operations is correct
     \[ a_1 \text{ op } a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2)) \]

For Next Time

• Read Cousot’s "Abstract Interpretation Based Formal Methods and Future Challenges"
• Read Thompson’s Turing Award Lecture
• Project Proposal

Review of Verification Conditions
Additional Exercises

• What is the VC for

\[
\text{for } i = e_{\text{low}} \text{ to } e_{\text{high}} \text{ do Inv } c
\]