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

Meeting 20, CSCI 5535, Spring 2010

Announcements

- Homework 7 due today
- No new homework today
  - Work on your projects!
  - Project status due Tue Mar 30

Guest Lecturer on March 30

Manu Sridharan

Pointer analysis – how to reason about heap locations

The Problem: Static Analysis

- It is extremely useful to predict program behavior statically (= without running the program)
  - Why? What uses?
  - Finding bugs (crashes, security)
  - Compiler optimization

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

The Problem: Static Analysis

- It is extremely useful to predict program behavior statically (= without running the program)
  - For optimizing compilers, program analyses, software engineering tools, finding security flaws, etc.
- The semantics we studied so far give us the precise behavior of a program
- However, precise static predictions are impossible
  - The exact semantics is not computable
- We must settle for approximate, but correct, static analyses (e.g. VC vs. WP)
A Tiny Language

Consider the following language of arithmetic ("shrIMP")

\[ e ::= n \mid e_1 \cdot e_2 \]

Denotational semantics of this language

\[ \llbracket n \rrbracket = n \]
\[ \llbracket e_1 \cdot e_2 \rrbracket = \llbracket e_1 \rrbracket \cdot \llbracket e_2 \rrbracket \]

Take denot. 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

\[ \sigma : \text{Exp} \to \{-, 0, +\} \]

\[ \sigma(n) = \text{sign}(n) \]
\[ \sigma(e_1 \cdot e_2) = \sigma(e_1) \circ \sigma(e_2) \]

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?

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:

\[ \beta : \mathbb{Z} \to \{-, 0, +\} \]

This is called the abstraction function (\( \beta \))

- This three-element set is the abstract domain

Also define the concretization function (\( \gamma \)):

\[ \gamma : \{-, 0, +\} \to \mathbb{Z} \]

\[ \gamma(+) = \{ n \in \mathbb{Z} \mid n > 0 \} \]
\[ \gamma(0) = \{ 0 \} \]
\[ \gamma(-) = \{ n \in \mathbb{Z} \mid n < 0 \} \]

Another View of Soundness

Soundness can be stated succinctly

\[ \forall e \in \text{Exp}. [e] \in \gamma(\sigma(e)) \]

(\( \sigma(e) = + \))

\( \forall e \in \text{Exp}. [e] \in \gamma(\sigma(e)) \]

(\( \sigma(e) = - \))

\( \forall e \in \text{Exp}. [e] \in \gamma(\sigma(e)) \)

(\( \sigma(e) = 0 \))

\( \forall e \in \text{Exp}. [e] \in \gamma(\sigma(e)) \)
Another View of Soundness

- Soundness can be stated succinctly

\[ \forall e \in \text{Exp}. [e] \in \gamma(\sigma(e)) \]

(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:

\[
\begin{array}{ccc}
\text{Exp} & \xrightarrow{\sigma} & A \\
C & \xrightarrow{\gamma \text{ (concretization)}} & \mathcal{P}(C)
\end{array}
\]

One-Slide Summary: Abstract Interp

- This is our first example of an abstract interpretation

- We carry out computation in an abstract domain \( \exists \oplus \ominus \ominus \ominus \)

- 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

\[
e ::= n \mid e_1 \star e_2 \mid - e
\]

- We define \( \sigma(-e) = \ominus \sigma(e) \)

\[
\begin{array}{|c|c|c|}
\hline
& - & 0 + \\
\hline
\ominus & 0 & - \\
\hline
\end{array}
\]
Adding Unary Minus and Addition

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

Now we add addition:

- We define \( \sigma(e_1 + e_2) = \sigma(e_1) \oplus \sigma(e_2) \)

Adding Addition

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

Loss of Precision

- Abstract computation may lose information:
  \[ [(1 + 2) + -3] = 0 \]
  but: \( \sigma((1+2) + -3) = \gamma(1+2) \oplus \sigma(-3) \)
  \[ \gamma(1) \oplus \sigma(2) \oplus \sigma(-3) \]
  \[ = + \oplus + \oplus - = + \oplus - = \top \]

- 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 zero
    \[ 0/0 = ? \]
    \[ \gamma(0/0) = \top \]

Adding Division

- Straightforward except for division by 0
  - We say that there is no answer in that case
  - \( \gamma(x/0) : \{ n \mid n \neq n_1 \oplus 0, n_1 > 0 \} \)
- Introduce \( \bot \) to be the abstraction of the \( 0 \)
- \( \bot = "\text{nothing}"
- \( \top = "\text{something unknown}"

The Abstract Domain

• Our abstract domain forms a lattice
• A partial order is induced by \( \gamma \)
  \( a_1 \sqsubseteq a_2 \) iff \( \gamma(a_1) \subseteq \gamma(a_2) \)
  - 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 glb
  - 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

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 \( \gamma \)
  \( \gamma : A \to \mathcal{P}(C) \)
  \( \gamma(a) = \{ x \in C \mid \beta(x) \sqsubseteq a \} \)
• And the abstraction for sets \( \alpha \)
  \( \alpha : \mathcal{P}(C) \to A \)
  \( \alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \} \)

Example: With Our Sign Lattice

• Consider our sign lattice
  \( + \quad \text{if} \ n > 0 \)
  \( 0 \quad \text{if} \ n = 0 \)
  \( - \quad \text{if} \ n < 0 \)
• \( \alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \} \)
  - Example: \( \alpha \{1, 2\} = + \quad \alpha \{1, 0\} = 0 \quad \alpha \{0\} = - \)
• \( \gamma(a) = \{ n \mid \beta(n) \sqsubseteq a \} \)
  - Example: \( \gamma(+) = \{ n \mid n > 0 \} \quad \gamma(0) = \{ n \mid n < 0 \} \quad \gamma(-) = \{0\} \)

Lattice History

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

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

- \[ \alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \} \]
  - Example:
    \[ \alpha(\{1, 2\}) = \text{lub} \{ + \} = + \]
    \[ \alpha(\{1, 0\}) = \text{lub} \{ +, 0 \} = + \]

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

Galois Connections

- We can show that
  - \( \gamma \) and \( \alpha \) are monotonic (with \( \subseteq \) ordering on \( \mathcal{P}(C) \))
  - \( \alpha(\gamma(a)) = a \) for all \( a \in A \)
  - \( \gamma(\alpha(S)) \subseteq S \) for all \( S \in \mathcal{P}(C) \)
- Such a pair of functions is called a **Galois connection**
  - Between the lattices \( A \) and \( \mathcal{P}(C) \)

Correctness Condition

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

\[ \begin{align*}
\text{Denotation} & \quad \gamma(\alpha(S)) \\
\text{Abstract Domain} & \quad \mathcal{P}(C) \\
\text{Abstract Semantics} & \quad \sigma \\
\text{Exp} & \quad \mathcal{A} \\
\end{align*} \]

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
     \[ = "\alpha \text{ and } \gamma \text{ are almost inverses}" \]
  3. Abstraction of operations is correct
     \[ a_1 \text{ op }^\alpha a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2)) \]

Review of Verification Conditions

- What is the VC for
  \[ \text{for } i = e_{\text{low}} \text{ to } e_{\text{high}} \text{ do } \text{Inv } c \]