Abstract Interpretation
(Non-Standard Semantics)
a.k.a.
“Picking The Right Abstraction”
Why analyze programs statically?
The Problem

- It is extremely useful to predict program behavior \textit{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 \textit{not computable}

- We must settle for \textit{approximate}, but correct, static analyses (e.g. VC vs. WP)
One-Slide Summary

- **Abstraction interpretation** is a static analysis for soundly approximating the semantics of a program.
- While the **concrete semantics** refers to what actually happens when you run the program (e.g., “x\*x+1” may result in multiple integers), the **abstract semantics** tracks only certain information about that computation (e.g., “x\*x+1” will be some *positive* number, but we don't know which one).
- Special functions transfer between the **abstract domain** (typically a *lattice*) and the **concrete domain**.
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")?

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

- The operational semantics of this language

\[
n \Downarrow n
\]

\[
e_1 \ast e_2 \Downarrow = e_1 \Downarrow \times e_2 \Downarrow
\]

- We’ll take opsem 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

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

\[ \sigma(n) = \text{sign}(n) \]
\[ \sigma(e_1 \times e_2) = \sigma(e_1) \otimes \sigma(e_2) \]
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 example :-) 

• What could we be computing instead?
Correctness of Sign Abstraction

• We can show that the abstraction is correct in the sense that it predicts the sign

\[ e \uparrow > 0 \iff \sigma(e) = + \]
\[ e \downarrow = 0 \iff \sigma(e) = 0 \]
\[ e \uparrow < 0 \iff \sigma(e) = - \]
Correctness of Sign Abstraction

• We can show that the abstraction is correct in the sense that it predicts the sign
  \[ e \downarrow > 0 \iff \sigma(e) = + \]
  \[ e \downarrow = 0 \iff \sigma(e) = 0 \]
  \[ e \downarrow < 0 \iff \sigma(e) = - \]

• Our semantics is abstract but precise

• Proof is by structural induction on the expression \( e \)
  - Each case repeats similar reasoning
Another View of Soundness

• Link each concrete value to an abstract one:
  \[ \beta : \mathbb{Z} \rightarrow \{ -, 0, + \} \]

• This is called the **abstraction function** \((\beta)\)
  - This three-element set is the **abstract domain**

• Also define the **concretization function** \((\gamma)\):
  \[ \gamma : \{-, 0, +\} \rightarrow \mathcal{P}(\mathbb{Z}) \]
  \[
  \begin{align*}
  \gamma(+) &= \{ n \in \mathbb{Z} \mid n > 0 \} \\
  \gamma(0) &= \{ 0 \} \\
  \gamma(-) &= \{ n \in \mathbb{Z} \mid n < 0 \}
  \end{align*}
  \]
Another View of Soundness 2

• Soundness can be stated succinctly

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

(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:

\[
\begin{array}{ccc}
\text{Exp} & \xrightarrow{\sigma} & A \\
\downarrow & & \downarrow \\
C & \xrightarrow{\in} & \mathcal{P}(C)
\end{array}
\]
Another View of Soundness 3

- Consider the **generic abstraction** of an operator
  \[
  \sigma(e_1 \text{ op } e_2) = \sigma(e_1) \text{ op } \sigma(e_2)
  \]

- This is sound iff
  \[
  \forall a_1, \forall a_2. \gamma(a_1 \text{ op } a_2) \supseteq \{ n_1 \text{ op } n_2 \mid n_1 \in \gamma(a_1), n_2 \in \gamma(a_2) \}
  \]

- e.g. \[
  \gamma(a_1 \otimes a_2) \supseteq \{ n_1 \times n_2 \mid n_1 \in \gamma(a_1), n_2 \in \gamma(a_2) \}
  \]

- This reduces the proof of correctness to **one proof for each** operator
Abstract Interpretation

• 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
  \[ e ::= n \mid e_1 \ast e_2 \mid - e \]

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

• Now we add addition:
  \[ e ::= n \mid e_1 \ast e_2 \mid - e \mid e_1 + e_2 \]

• 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 “+ ⊕ -”?
- Start from the soundness condition:
  \[ \gamma(\, + \oplus - \, ) \supseteq \{ \, n_1 + n_2 \mid n_1 > 0, n_2 < 0 \} = \mathbb{Z} \]

- We don’t have an abstract value whose concretization includes \( \mathbb{Z} \), so we add one:
  \[ \top \] ("top" = "don’t know")

<table>
<thead>
<tr>
<th>⊕</th>
<th>-</th>
<th>0</th>
<th>+</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>T</td>
<td>T</td>
<td></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>
Loss of Precision

• Abstract computation may lose information:

\[ \lceil (1 + 2) + -3 \rceil = 0 \]

but:
\[ \sigma((1+2) + -3) = \]
\[ (\sigma(1) \oplus \sigma(2)) \oplus \sigma(-3) = \]
\[ (+ \oplus +) \oplus - = \top \]

• 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

- Straightforward except for division by 0
  - We say that there is no answer in that case
  - \( \gamma(+ \odot 0) = \{ n \mid n = n_1 / 0 \; , \; n_1 > 0 \} = \emptyset \)

- Introduce \( \perp \) to be the abstraction of the \( \emptyset \)
  - We also use the same abstraction for non-termination!
    - \( \perp \) = “nothing”
    - \( \top \) = “something unknown”

<table>
<thead>
<tr>
<th>( \emptyset )</th>
<th>-</th>
<th>0</th>
<th>+</th>
<th>( \top )</th>
<th>( \perp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
<td>0</td>
<td>-</td>
<td>( \top )</td>
<td>( \perp )</td>
</tr>
<tr>
<td>0</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>0</td>
<td>+</td>
<td>( \top )</td>
<td>( \perp )</td>
</tr>
<tr>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
</tr>
</tbody>
</table>
Game Criticism

- This term refers to a conflict between the mechanics or dynamics of a game and its story. For example, *BioShock* was viewed as promoting selflessness through story but selfishness through gameplay, a disconnect that pulled some players out of the game. The term is often viewed as “highbrow” or “pretentious”.
• This 1962 Newbery Medal-winning novel by Madeleine L'Engle includes Charles Wallace, Mrs. Who, Mrs. Whatsit, Mrs. Which and the space-bending Tesseract.
This American Turing-award winner is known for developing Speedcoding and FORTRAN (the first two high-level languages), as well creating a way to express the formal syntax of a language and using that approach to specify ALGOL. He later focused on function-level (as opposed to value-level) programming. His first major programming project calculated the positions of the Moon.
The Abstract Domain

- Our abstract domain forms a **lattice**
- A partial order is induced by $\gamma$
  
  $$a_1 \leq a_2 \iff \gamma(a_1) \subseteq \gamma(a_2)$$
  
  - 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 gub
  - Even infinite subsets!
• Every finite lattice is (trivially) complete
• Every complete lattice is a complete partial order (recall: proof techniques: induction!)
  - 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
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 $\text{glb}$

- 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 $\beta$

$$\beta : C \rightarrow A$$

(maps a concrete value to the best abstract value)
- $A$ must be a lattice

• We can derive the concretization function $\gamma$

$$\gamma : A \rightarrow P(C)$$

$$\gamma(a) = \{ x \in C \mid \beta(x) \leq a \}$$

• And the abstraction for sets $\alpha$

$$\alpha : P(C) \rightarrow A$$

$$\alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \}$$
Example

• 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: \( \alpha([1, 2]) = \text{lub} \{ + \} = + \)
  - \( \alpha([1, 0]) = \text{lub} \{ +, 0 \} = \top \)
  - \( \alpha(\{\}) = \text{lub} \emptyset = \bot \)

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

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

\[
\begin{align*}
\text{Exp} & \xrightarrow{\sigma} \text{A} \\
\downarrow & \\
\mathcal{C} & \xrightarrow{\gamma} \mathcal{P}(\mathcal{C}) \\
\end{align*}
\]

- \(\mathcal{C}\) means concrete domain
- \(\mathcal{P}(\mathcal{C})\) means abstract domain
- \(\sigma\) means abstract semantics
- \(\alpha(\leq)\) means abstraction function for sets
- \(\gamma\) means concretization function
Three Little Correctness Conditions

• Three conditions define a correct abstract interpretation
  
• $\alpha$ and $\gamma$ are monotonic

• $\alpha$ and $\gamma$ form a Galois connection
  
  = “$\alpha$ and $\gamma$ are almost inverses”

1. Abstraction of operations is correct

  $a_1 \text{ op } a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2))$
“On The Board” Questions

• What is the VC for:

$$\text{for } i = e_{\text{low}} \text{ to } e_{\text{high}} \text{ do } c \text{ done}$$

• This axiomatic rule is unsound. Why?

$$\begin{align*}
\Gamma \vdash \{A \land p\} \quad &c_{\text{then}}\quad \{B_{\text{then}}\} \\
\Gamma \vdash \{A \land \neg p\} \quad &c_{\text{else}}\quad \{B_{\text{else}}\}
\end{align*}$$

$$\begin{align*}
\Gamma \vdash \{A\} \quad &\text{if } p \text{ then } c_{\text{then}} \quad \text{else } c_{\text{else}} \quad \{B_{\text{then}} \lor B_{\text{else}}\}
\end{align*}$$
Homework

• Read Cousot & Cousot Article