Abstract Interpretation: concrete and abstract semantics
Concrete semantics

• We consider a very tiny language that manages arithmetic operations on integers values.

• The (concrete) semantics of the languages cab be defined by the funcion $\mu$ defined by:

$$
e = i \mid e \ast e$$

$$\mu : Exp \rightarrow Int$$

$$\mu(i) = i$$

$$\mu(e_1 \ast e_2) = \mu(e_1) \times \mu(e_2)$$
Abstract Semantics

• Consider now an abstract semantics over the domain of signs

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

\[ \sigma(i) = \begin{cases} + & \text{if } i > 0 \\ 0 & \text{if } i = 0 \\ - & \text{if } i < 0 \end{cases} \]

\[ \sigma(e_1 \times e_2) = \sigma(e_1) \times \sigma(e_2) \]
From a different perspective

- We can associate to each abstract value the set of concrete elements it represents.
- The concretization function:

\[ \gamma : \{+,-,0\} \rightarrow 2^{\mathbb{Int}} \]

\[ \gamma(+) = \{ i \mid i > 0 \} \]
\[ \gamma(0) = \{ 0 \} \]
\[ \gamma(-) = \{ i \mid i < 0 \} \]
Concretization

- The concretization function $\gamma$ maps an abstract value to a set of concrete elements.
- Let $D$ denote the concrete domain and $A$ denote the abstract domain. The correctness of the abstract semantics wrt the concrete one can be expressed by:

$$\mu(e) \in \gamma(\sigma(e))$$
Abstract Interpretation

• Abstract Interpretation is:
  – Computing the semantics of a program in an abstract domain
  – In the case of signs, the domain so far is \{+,0,-\}.

• The abstract semantics should be correct
  – it is an over approximation of the concrete semantics

• The relation between the two domains is given by a concretization function
Consider the unary operator - 

- Let us add to our language the unary operator -

$$\mu(-e) = -\mu(e)$$

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

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>+</th>
<th>0</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>+</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>
Consider the binary operation +

- Adding the addition operator forces us to modify the domain, as the previous one is not able to represent the result of adding numbers of opposite sign

\[
\begin{align*}
\mu(e_1 + e_2) &= \mu(e_1) + \mu(e_2) \\
\sigma(e_1 + e_2) &= \sigma(e_1) - \sigma(e_2)
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>0</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>+</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>?</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
So...

- We add to the domain a new element that represents all the integer numbers (both positive and negative, and zero)

\[
\gamma(T) = \text{Int}
\]

<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>
The operations should be revisited

<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></td>
<td></td>
</tr>
<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></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>0</td>
<td>+</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

10
Examples

Sometimes there is information loss due to the abstract operations

\[ \mu((1 + 2) + -3) = 0 \]
\[ \sigma((1 + 2) + -3) = (+ \quad +) + (\bar{-}+) = T \]

Sometimes there is no information loss, with respect to the abstraction

\[ \mu((5 * 5) + 6) = 31 \]
\[ \sigma((5 * 5) + 6) = (+ \quad \bar{x} +) + + = + \]
Consider the division operator /

- Problem: what is the result of dividing by zero? No number!
- So we need a new element in our domain that represents the empty set of integers (i.e. a failure state)

- But.. What’s wrong in the table below?

$$\gamma(\bot) = \emptyset$$

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

$$\bot + x = \bot$$
$$-$$
$$x \times \bot = \bot$$
$$-$$
$$- \bot = \bot$$
The resulting abstract domain

- It is a finite complete lattice
- The partial order is coherent wrt the concretization function:

\[ x \leq y \iff \gamma(x) \subseteq \gamma(y) \]
The abstraction function

- The concretization function $\gamma$ has an adjoint function, the abstraction function $\alpha$.
- Function $\alpha$ maps a set of concrete values into the best representation of this set in the abstract domain (the smaller element of the abstract domain that represents some of these elements).
- In our example:

\[
\alpha : 2^{\mathbb{Int}} \rightarrow A \\
\alpha(S) = \text{lub}\left(\{- | i < 0 \land i \in S\}, \{0 | 0 \in S\}, \{+ | i > 0 \land i \in S\}\right) \\
\sigma(i) = \alpha(\{i\})
\]
A general definition

• An **Abstract Interpretation** consists of:
  – An abstract domain \( A \) and a concrete domain \( D \)
  – \( A \) and \( D \) are complete lattices. Smaller means “more precise”
  – Two monotone adjoint function that enjoy the *Galois insertion*.
  – Abstract operations that are correct wrt the concrete ones
  – A fixpoint algorithm

• **Galois insertion:**

\[
\forall x \in 2^D. \quad x \subseteq \gamma(\alpha(x))
\]

\[
\forall a \in A. \quad x = \alpha(\gamma(x))
\]
Correctness revisited

- If case of Galois insertion, these correctness conditions are equivalent (prove it !)

\[
\mu(e) \in \gamma(\sigma(e)) \\
\sigma(e) \geq \alpha(\{\mu(e)\})
\]
Correctness

- We show that in order to ensure the correctness of the whole analysis the following conditions are sufficient:
  1. The function $\alpha$ and $\gamma$ are monotone
  2. The function $\alpha$ and $\gamma$ form a Galois insertion
  3. The abstract operations are locally correct, i.e.

    $$\gamma(op(s_1,\ldots,s_n)) \supseteq op(\gamma(s_1),\ldots,\gamma(s_n))$$

- Notice that there is always a way to define a locally correct abstract operation. It is sufficient to consider the operations that returns the top element of the abstract domain.
Local correctness

\[ \gamma(\text{op}(s_1, \ldots, s_n)) \]

\[ \text{op}(\gamma(s_1), \ldots, \gamma(s_n)) \]
Correctness proof

• We show by structural induction on \( e \) that:

\[
\mu(e) \in \gamma(\sigma(e))
\]

• Basic step:

\[
\mu(i) = i \\
\subseteq \{i\} \\
\subseteq \gamma(\alpha(\{i\})) \\
= \gamma(\sigma(i))
\]
Correctness proof

Inductive Step

\[ \mu(e) \in \gamma(\sigma(e)) \]

\[ \mu(e_1 \text{ op } e_2) \]
\[ = \mu(e_1) \text{ op } \mu(e_2) \]
\[ \in \gamma(\sigma(e_1)) \text{ op } \gamma(\sigma(e_2)) \]
\[ \subseteq \gamma(\sigma(e_1) \overline{\text{ op }} \sigma(e_2)) \]
\[ = \gamma(\sigma(e_1 \text{ op } e_2)) \]
Adding an input

• We can extend our tiny language with the possibility to get an input value from the user
• This means that we have a variable $x$ in the expressions

$$e = i \mid e \cdot e \mid -e \mid \ldots \mid x$$
Concrete semantics

- The semantic function $\mu$ becomes

$$\mu : \text{Exp} \rightarrow \text{Int} \rightarrow \text{Int}$$

- And we may express it in terms of a family of functions, having expressions as indices and a single parameter (the input value)

$$\begin{align*}
\mu_i(j) &= i \\
\mu_x(j) &= j \\
\mu_{e_1* e_2}(j) &= \mu_{e_1}(j) \ast \mu_{e_2}(j) \\
\mu_{e_1+ e_2}(j) &= \mu_{e_1}(j) \ast \mu_{e_2}(j) \\
... &= ...
\end{align*}$$
Abstract semantics

• The same holds for the abstract semantic function $\sigma$
  
  $\sigma : \text{Exp} \rightarrow A \rightarrow A$

• Also in this case we can express $\sigma$ by a family of functions:
  
  $\sigma_i(\bar{j}) = \bar{i}$
  $\sigma_x(\bar{j}) = \bar{j}$
  $\sigma_{e_1*e_2}(\bar{j}) = \sigma_{e_1}(\bar{j}) \ast \sigma_{e_2}(\bar{j})$
  $\sigma_{e_1+e_2}(\bar{j}) = \sigma_{e_1}(\bar{j}) + \sigma_{e_2}(\bar{j})$
  ...
  $\bar{i} = \alpha(\{i\})$
Correctness

- The following conditions are equivalent

\[ \forall i. \; \mu_e(i) \in \gamma(\sigma_e(\alpha(\{i\}))) \]

\[ \mu_e \leq_D \gamma \circ \sigma_e \circ \alpha \]

\[ \alpha \circ \mu_e \leq_A \sigma_e \circ \alpha \]
Local correctness

- We can express the local correctness condition by:

\[
op(\gamma(\sigma_{e_1}(\bar{j})), \ldots, \gamma(\sigma_{e_n}(\bar{j}))) \subseteq \gamma(\op(\sigma_{e_1}(\bar{j}), \ldots, \sigma_{e_n}(\bar{j})))
\]
Conditional statement

\[ e = \ldots \mid \text{if } e = e \text{ then } e \text{ else } e \mid \ldots \]

- **Concrete semantics**
  \[
  \mu_{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4}(i) = \begin{cases} 
  \mu_{e_3}(i) & \text{if } \mu_{e_1}(i) = \mu_{e_2}(i) \\
  \mu_{e_4}(i) & \text{if } \mu_{e_1}(i) \neq \mu_{e_2}(i)
  \end{cases}
  \]

- **Abstract semantics**
  \[
  \sigma_{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4}(i) = \sigma_{e_3}(\bar{i}) \sqcup \sigma_{e_4}(\bar{j})
  \]

- Notice the role of the lub in the abstract domain
Correctness of the conditional statm.

• Assume that the condition is true (the other case is analogous)

\[
\begin{align*}
\mu_{e_3}(i) & \in \gamma(\sigma_{e_3}(\overline{i})) \\
\subseteq & \gamma(\sigma_{e_3}(\overline{i})) \sqcup \gamma(\sigma_{e_4}(\overline{i})) \\
\subseteq & \gamma(\sigma_{e_3}(i) \sqcup \sigma_{e_4}(i)) \\
= & \gamma(\sigma_{\text{if } e_1=e_2 \text{ then } e_3 \text{ else } e_4}(i))
\end{align*}
\]