Abstract Interpretation

CSE 501
Spring 15
Distributivity of Frameworks

- \((G, L, F, M)\) is distributive iff
  \[ f(x \land y) = f(x) \land f(y) \]

  \[
  \begin{align*}
  f(x \land y) &= f(\{3,2,?\} \land \{2,3,?\}) \\
  &= f(\{\top, \top, ?\}) \\
  &= \{\top, \top, \top\}
  \\
  f(x) \land f(y) &= f(\{3,2,?\}) \land f(\{2,3,?\}) \\
  &= \{3,2,5\} \land \{2,3,5\} \\
  &= \{\top, \top, 5\}
  \\
  \end{align*}
  \]

Ordering of evaluation matters!
Maximal Fixed Point (MFP) Solution

• Fact: the iterative solution to dataflow equations is the most precise

• Intuition:
  – Start with the top element at each program point
  – Refine during each iteration to satisfy all dataflow equations
  – Final result will be closest to the top

• Hence for any solution FP of dataflow equations: FP ≤ MFP
Meet Over Paths (MOP) Solution

• Another approach to solve the dataflow equations:
  – Enumerate each path \( p_k = [\text{entry}, n_1, n_2, ..., n_k] \)
  – Define \( \text{IN}[p_k] = f_{n_{k-1}}(... (f_{n_1} (f_{n_0}(d_0)))) \), where \( d_0 \) is the flow element for entry

  – Compute final solution as
    \[ \text{IN}[n] = \bigcup \{ \text{IN}[p] \ . \ p \text{ is a path from entry to } p \} \]
MFP and MOP

- Fact: MFP ≤ MOP
- Why not compute MOP in practice?

How many paths can reach B2?
MFP and MOP

• Fact: For transfer functions that are distributive, then MFP = MOP

• Recall: \( f(x \land y) = f(x) \land f(y) \)

• Hence \( f(x_1) \land f(x_2) \land f(x_3) \ldots = f(\land x_i) \)

• We can compute MOP using iterative algorithm!
Can we do even better?

• Fact: MFP, MOP are conservative

• Some paths are not possible
• IDEAL = solution that takes into account of feasible paths
• FP ≤ MFP ≤ MOP ≤ IDEAL

• Great!
  - but this is undecidable 😞
Summary

• Dataflow framework = (G, L, F, M)

• Possible solutions: FP, MFP, MOP, IDEAL
  – FP ≤ MFP ≤ MOP ≤ IDEAL

• In practice, compilers compute MFP using the iterative algorithm
ABSTRACT INTERPRETATION: A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France
Where it all started...

• Inspirations from
  – Dataflow analysis
  – Denotational semantics

• Enthusiastically embraced by the community
  – At least the functional community . . .
  – At least the first half of the paper . . .
AI by Example

with slides from Prof. Alex Aiken
A Tiny Language

- Language with only integers and multiplication

\[ e = i \mid e \times e \]

\[ \mu : \text{Exp} \rightarrow \text{Int} \]  
\[ \mu(i) = i \]
\[ \mu(e \times e) = \mu(e) \times \mu(e) \]

- Goal: define a semantics to compute the sign of all expressions without actually carrying out the computation
An Abstraction

• Define an abstract semantics that computes only the sign of the result.

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

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

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

• We can show that this abstraction is correct in that it correctly predicts the sign of an expression.

• Proof is by structural induction on \( e \).

\[
\begin{align*}
\mu(e) > 0 & \iff \sigma(e) = + \\
\mu(e) = 0 & \iff \sigma(e) = 0 \\
\mu(e) < 0 & \iff \sigma(e) = -
\end{align*}
\]
Another View of Soundness

• The soundness proof is clunky
• Instead, directly associate each abstract value with the set of concrete values it represents.

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

\[ \gamma(+) = \{ i \mid i>0 \} \]
\[ \gamma(0) = \{0\} \]
\[ \gamma(-) = \{ i \mid i<0 \} \]
Another View of Soundness

• The concretization function $\gamma$
  – Mapping from abstract values to (sets of) concrete values

• Let
  – $D$ be the concrete domain
  – $A$ be the abstract domain

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

• This is an abstract interpretation
  – Computation in an abstract domain
  – In this case \{+,0,\}.

• The abstract semantics is sound
  – approximates the standard semantics.

• The concretization function establishes the connection between the two domains.
Adding -

• Extend our language with unary -

\[ \mu(-e) = -\mu(e) \]
\[ \sigma(-e) = -\sigma(e) \]
Adding +

- Adding addition is not so easy.
- The abstract values are not closed under addition.

\[
\mu(e_1 + e_2) = \mu(e_1) + \mu(e_2)
\]

\[
\sigma(e_1 + e_2) = \sigma(e_1) \pm \sigma(e_2)
\]
Solution

• We need another abstract value to represent a result that can be any integer
• Finding a domain closed under all the abstract operations is often a key design problem
• Recall: defining lattice for dataflow analysis

\[ \gamma(T) = \text{all integers} \]

<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>
Extending Other Operations

- We also need to extend the other abstract operations to work with $T$.

<table>
<thead>
<tr>
<th>$\times$</th>
<th>$+$</th>
<th>$0$</th>
<th>$-$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<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>

\[
- \begin{array}{c|cccc}
  & + & 0 & - & T \\
\hline
  + & + & 0 & - & T \\
  0 & 0 & 0 & 0 & 0 \\
  - & - & 0 & + & T \\
  T & T & 0 & T & T \\
\end{array}
\]

\[
- \begin{array}{c|cccc}
  & + & 0 & - & T \\
\hline
  - & + & 0 & - & T \\
  - & 0 & + & T \\
\end{array}
\]
Examples

• Abstract computation doesn’t lose information:
  \[ \mu((5 \times 5) + 6) = 31 \]
  \[ \sigma((5 \times 5) + 6) = (+ \times +) \pm + = + \]

• Sometimes it does:
  \[ \mu((1 + 2) + -3) = 0 \]
  \[ \sigma((1 + 2) + -3) = (+ \pm +) \pm (-+) = T \]
Adding / (Integer Division)

• Adding / is straightforward except for the case of division by 0.

• If we divide each integer in a set by 0, what set of integers results?
  – The empty set.

\[ \gamma(\bot) = \emptyset \]
Adding / (Integer Division)

• As before we need to extend the other abstract operations.

• In this case, every entry involving bottom is bottom
  – all operations are strict in bottom

\[
\begin{align*}
\bot + x &= \bot \\
x \times \bot &= \bot \\
-\bot &= \bot
\end{align*}
\]
The Abstract Domain

• Our abstract domain forms a complete lattice.
  – A partial order $x \leq y \iff \gamma(x) \subseteq \gamma(y)$

• Every finite subset has a least upper bound (lub, $\sqcup$) and greatest lower bound (glb, $\sqcap$).

• We write $A$ for an abstract domain
  – a set of values $+$ an ordering

\[
\begin{array}{c}
T \\
+ \\
0 \\
- \\
\perp
\end{array}
\]
The Abstraction Function

- The abstraction function maps concrete values to abstract values.
  - The dual of concretization.
  - The smallest value of $A$ that is the abstraction of a set of concrete values.

$$\alpha: 2^{\text{Int}} \rightarrow A$$
$$\alpha(S) = \text{lub}( \{- | i < 0 \land i \in S\}, \{0 | 0 \in S\}, \{+ | i > 0 \land i \in S\})$$
An Aside: Galois Connection

- $(L, \alpha, \gamma, M)$ is a Galois connection between complete lattices $(L, \leq)$ and $(M, \leq)$ iff:
  - $\alpha: L \to M$ and $\gamma: M \to L$ are monotone functions

Furthermore:
- $\text{id} \leq \gamma \circ \alpha$
- $\text{id} \leq \alpha \circ \gamma$

- The function $\alpha \circ \gamma$ is called a Galois insertion
A General Definition

• An abstract interpretation consists of
  – An abstract domain $A$ and concrete domain $D$
  – Concretization and abstraction functions forming a Galois insertion.
  – A (sound) abstract semantic function.

In our example:

$$\forall x \in 2^D . x \subseteq \gamma(\alpha(x))$$

$$\forall a \in A . a = \alpha(\gamma(x))$$

or

$$id \leq \gamma \circ \alpha$$

$$id = \alpha \circ \gamma$$
Galois Insertions

- The abstract domain can be thought of as dividing the concrete domain into non-disjoint subsets.

- The abstraction function maps a subset of the domain to the smallest containing abstract value.
Pictorially

- In correct abstract interpretations, we expect the following diagram to commute.

\[
\begin{array}{c}
\text{Exp} \\
\sigma \downarrow \quad \downarrow \gamma
\end{array}
\quad \text{A} \quad \text{A}
\]

\[
\begin{array}{c}
\mu \in 2^D \\
\alpha
\end{array}
\]
General Conditions for Correctness

- Three conditions guarantee correctness in general:
  - $\alpha$ and $\gamma$ form a Galois insertion
    - $\text{id} \leq \gamma \circ \alpha$, $\text{id} = \alpha \circ \gamma$
  - $\alpha$ and $\gamma$ are monotonic
    - $x \leq y \Rightarrow \alpha(x) \leq \alpha(y)$
  - Abstract operations $\text{op}$ are locally correct:
    - $\gamma(\text{op}(s_1, ..., s_n)) \subseteq \text{op}(\gamma(s_1), ..., \gamma(s_n))$
Generic Correctness Proof

• Proof by induction on the structure of e:
  \[ \mu(e) \in \gamma(\sigma(e)) \]

\[ \mu(e_1 \text{ op } e_2) \]
\[ = \mu(e_1) \text{ op } \mu(e_2) \quad \text{[definition of } \mu] \]
\[ \in \gamma(\sigma(e_1)) \text{ op } \gamma(\sigma(e_2)) \quad \text{[induction]} \]
\[ \subseteq \gamma(\sigma(e_1) \text{ op } \sigma(e_2)) \quad \text{[local correctness]} \]
\[ = \gamma(\sigma(e_1 \text{ op } e_2)) \quad \text{[definition of } \sigma] \]
Another Notion of Correctness

- We can define correctness using abstraction instead of concretization.

\[ \mu(e) \in \gamma(\sigma(e)) \iff \alpha(\{\mu(e)\}) \leq \sigma(e) \]

Proof for \( \Rightarrow \) direction:

\[
\begin{align*}
\mu(e) &\in \gamma(\sigma(e)) \\
\alpha(\{\mu(e)\}) &\leq \alpha(\gamma(\sigma(e))) \quad \text{[monotonicity]} \\
\alpha(\{\mu(e)\}) &\leq \sigma(e) \quad \text{[\( \alpha \circ \gamma = \text{id} \)]}
\end{align*}
\]
Another Notion of Correctness

\[ \mu(e) \in \gamma(\sigma(e)) \iff \alpha(\{\mu(e)\}) \leq \sigma(e) \]

Proof for \(\iff\) direction:
\[
\begin{align*}
\alpha(\{\mu(e)\}) & \leq \sigma(e) \\
\gamma(\alpha(\{\mu(e)\})) & \leq \gamma(\sigma(e)) \quad [\text{monotonicity}] \\
\mu(e) & \in \gamma(\sigma(e)) \quad [\text{id} \leq \gamma \circ \alpha]
\end{align*}
\]
Extending Our Language

• Add input to the language
  – Modeled as a single free variable $x$ in expressions

\[ e = i \mid e \ast e \mid -e \mid e + e \mid \ldots \mid x \]
Semantics

• The meaning function now has type
  \[ \mu : \text{Exp} \rightarrow \text{Int} \rightarrow \text{Int} \]
• We write the function with the expression as a subscript.
  \[ \mu_i(j) = i \]
  \[ \mu_x(j) = j \]
  \[ \mu_{e_1 \ast e_2}(j) = \mu_{e_1}(j) \ast \mu_{e_2}(j) \]
  \[ \mu_{e_1 + e_2}(j) = \mu_{e_1}(j) + \mu_{e_2}(j) \]
  ...

Abstract Semantics

- Abstract semantic function:
  \[ \sigma : \text{Exp} \rightarrow A \rightarrow A \]
- Also write this semantics in the same form.

\[
\begin{align*}
\sigma_i(j) &= i \\
\sigma_x(j) &= j \\
\sigma_{e_1* e_2}(j) &= \sigma_{e_1}(j) * \sigma_{e_2}(j) \\
\sigma_{e_1+ e_2}(j) &= \sigma_{e_1}(j) + \sigma_{e_2}(j) \\
\ldots \ldots \\
\hat{i} &= \alpha(\{i\})
\end{align*}
\]
Correctness

• The correctness condition needs to be generalized.
• This is the first real use of the abstraction function.
• The following are all 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 also need a modified local correctness condition.

\[ \text{op}(\gamma(\sigma_{e1}(j)), \ldots, \gamma(\sigma_{en}(j))) \subseteq \gamma(\text{op}(\sigma_{e1}(j), \ldots, \sigma_{en}(j))) \]
Proof of Correctness

• Theorem: $\mu_e(j) \in \gamma(\sigma_e(j))$

• Proof (by induction on the structure of $e$):

Base case: $\mu_i(j) = i \in \gamma(i) = \gamma(\sigma_i(j))$

$\mu_x(j) = j \in \gamma(j) = \gamma(\sigma_x(j))$

Induction on $\mu_{op(e_1,\ldots,e_n)}(j)$:

$= \text{op}(\mu_{e_1}(j), \ldots, \mu_{e_n}(j))$ [ definition of $\mu$ ]

$\in \text{op}(\gamma(\sigma_{e_1}(j)), \ldots, \gamma(\sigma_{e_n}(j))$ [ induction ]

$\subseteq \gamma(\text{op}(\sigma_{e_1}(j), \ldots, \sigma_{e_n}(j)))$ [ local correctness ]

$= \gamma(\sigma_{op(e_1,\ldots,e_n)}(j))$ [ definition of $\sigma$ ]
If-Then-Else

- $e = \ldots | \text{if } e = e \text{ then } e \text{ else } e | \ldots$

- $\mu_{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4}(i) = \mu_{e_3}(i), \text{ if } \mu_{e_1}(i) = \mu_{e_2}(i)$
  $= \mu_{e_4}(i), \text{ otherwise}$

- $\sigma_{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4}(i) = \sigma_{e_3}(i) \sqcup \sigma_{e_4}(i)$

- Recall that the abstract domain forms a complete lattice
Correctness of If-Then-Else

• Need to show that: \( \mu_e(j) \in \gamma(\sigma_e(j)) \)
  — Where e is an if-then-else

• Assume the true branch is taken.

• (The argument for the false branch is symmetric.)

\[
\begin{align*}
\mu_{e3}(i) \\
\subseteq \gamma(\sigma_{e3}(i)) & \quad \text{[by induction]} \\
\subseteq \gamma(\sigma_{e3}(i)) \sqcup \gamma(\sigma_{e4}(i)) \\
\subseteq \gamma(\sigma_{e3}(i) \sqcup \sigma_{e4}(i)) & \quad \text{[by monotonicity of } \gamma \text{]} 
\end{align*}
\]
Designing an Abstract Interpretation

• Define abstract domain
  – Needs to be a lattice

• Define the abstraction and concretization functions
  – $\sigma : D \rightarrow A$
  – $\alpha : 2^D \rightarrow A$
    • $\alpha(S) = \text{lub}(\sigma(s))$, for all $s \in S$
  – $\gamma : A \rightarrow 2^D$

• For every expression, define how to operate in the abstract domain