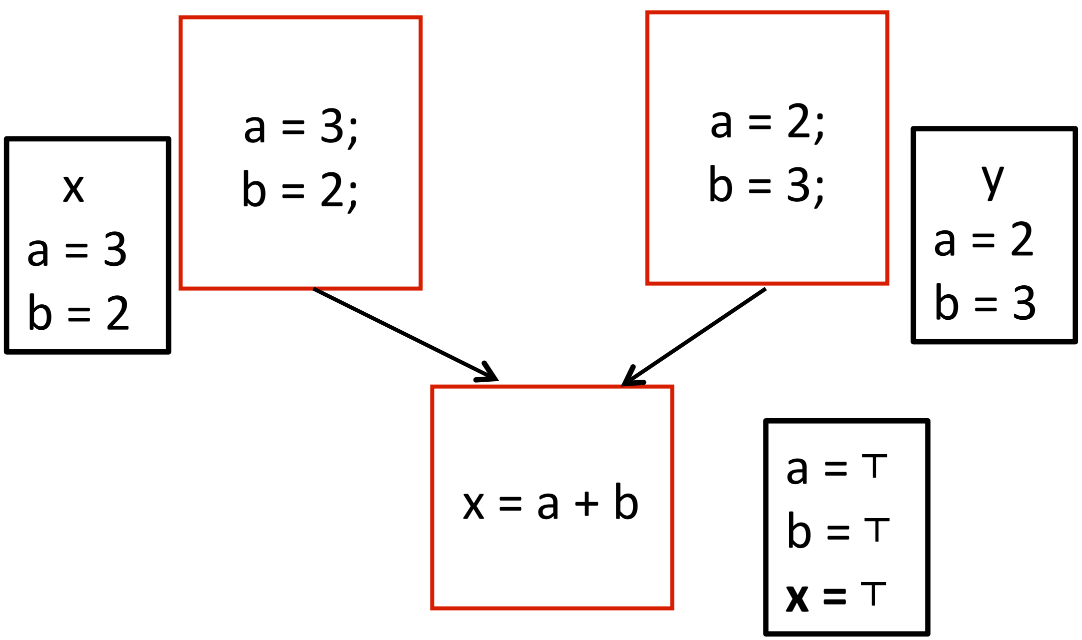
Abstract Interpretation

CSE 501

Spring 15

Distributivity of Frameworks

- (G, L, F, M) is distributive iff
  $f(x \wedge y) = f(x) \wedge f(y)$



$f(x \wedge y)$
$= f(\{3,2,?\} \wedge \{2,3,?\})$
$= f(\{\top, \top, ?\})$
$= \{\top, \top, \top\}$

$f(x) \wedge f(y)$
$= f(\{3, 2, ?\}) \wedge f(\{2, 3, ?\})$
$= \{3, 2, 5\} \wedge \{2, 3, 5\}$
$= \{\top, \top, 5\}$

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

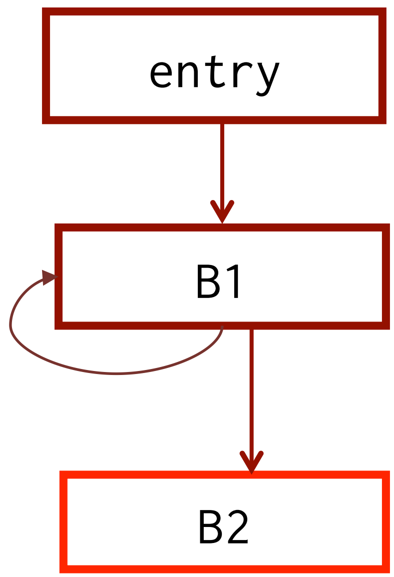
Meet Over Paths (MOP) Solution

- Another approach to solve the dataflow equations:
  - Enumerate each path p_k = [entry, n_1, $n_{2,}$ …, n_k]
  - Define $IN[p_k] = f_{nk-1}(\ldots (f_{n1} (f_{n0}(d_0))))$, where d_0 is the flow element for entry
  - Compute final solution as

    IN[n] = U { IN[p] . p 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

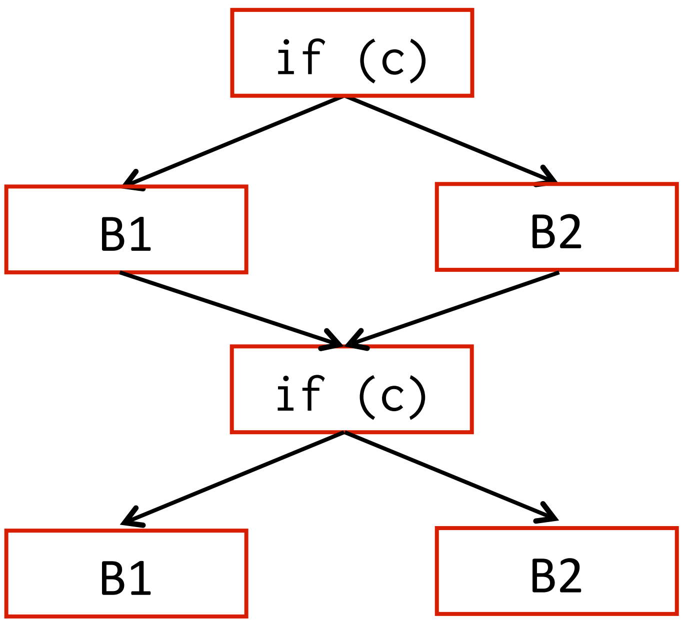
- Fact: For transfer functions that are distributive, then MFP = MOP

- Recall: $f(x \wedge y) = f(x) \wedge f(y)$
- Hence $f(x_1) \wedge f(x_2) \wedge f(x_3)$ ... $= f(\bigwedge x_i)$

- 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

Abstract Interpretation: A Unified Lattice Model for Static ...
dl.acm.org/ft_gateway.cfm?id... Association for Computing Machinery
by P Cousot - 1977 - Cited by 5755 - Related articles

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

$$e = i \mid e * e$$

$$\mu : \text{Exp} \rightarrow \text{Int}$$ ⟵ Denotation / meaning function

$$\mu(i) = i$$

$$\mu(e*e) = \mu(e) \times \mu(e)$$

- 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.

σ: Exp $\rightarrow$ {+, -, 0}

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

$\sigma(e*e) = \sigma(e) \underline{\times} \sigma(e)$

$\underline{\times}$	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

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.

$$\mu(e) > 0 \Leftrightarrow \sigma(e) = +$$
$$\mu(e) = 0 \Leftrightarrow \sigma(e) = 0$$
$$\mu(e) < 0 \Leftrightarrow \sigma(e) = -$$

Another View of Soundness

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

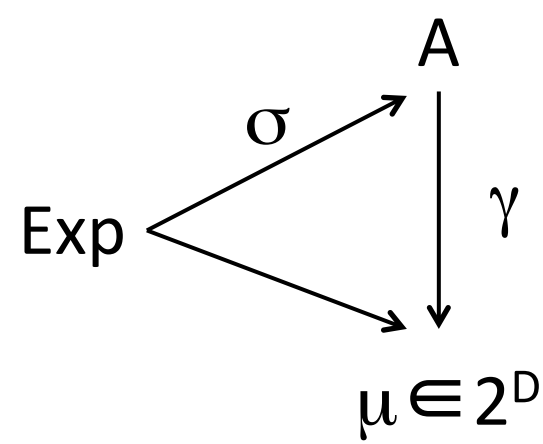
$$\gamma : \{+, 0, -\} \rightarrow 2^{Int}$$

$$\gamma(+) = \{\ i \mid i>0\ \}$$
$$\gamma(0) = \{0\}$$
$$\gamma(-) = \{\ i \mid i<0\ \}$$

Another View of Soundness

- The concretization function γ
  - Mapping from abstract values to (sets of) concrete values
- Let
  - **D** be the concrete domain
  - **A** be the abstract domain

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



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 -

$$\mu(-e) = -\mu(e)$$
$$\sigma(-e) = \underline{-}\,\sigma(e)$$

$\underline{-}$	+	0	-
	-	0	+

Adding +

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

$$\mu(e_1 + e_2) = \mu(e_1) + \mu(e_2)$$
$$\sigma(e_1 + e_2) = \sigma(e_1) \underline{+} \sigma(e_2)$$

$\underline{+}$	+	0	-
+	+	+	?
0	+	0	-
-	?	-	-

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

γ (T) = all integers

$\underline{+}$	+	0	-	T
+	+	+	T	T
0	+	0	-	T
-	T	-	-	T
T	T	T	T	T

Extending Other Operations

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

x	+	0	-	T
+	+	0	-	T
0	0	0	0	0
-	-	0	+	T
T	T	0	T	T

-	+	0	-	T
	-	0	+	T

Examples

- Abstract computation doesn't lose information:

  $\mu((5 * 5) + 6) = 31$

  $\sigma((5 * 5) + 6) = (+ \ \underline{\times} \ +) \ \underline{+} \ + = +$

- Sometimes it does:

  $\mu((1 + 2) + -3) = 0$

  $\sigma((1 + 2) + -3) = (+ \ \underline{+} \ +) \ \underline{+} \ (\underline{-}+) = \top$

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.

$\gamma(\bot)=\varnothing$

/	+	0	-	⊤	⊥
+	+	0	-	⊤	⊥
0	0	0	0	0	⊥
-	-	0	+	⊤	⊥
⊤	⊤	0	⊤	⊤	⊥
⊥	⊥	⊥	⊥	⊥	⊥

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

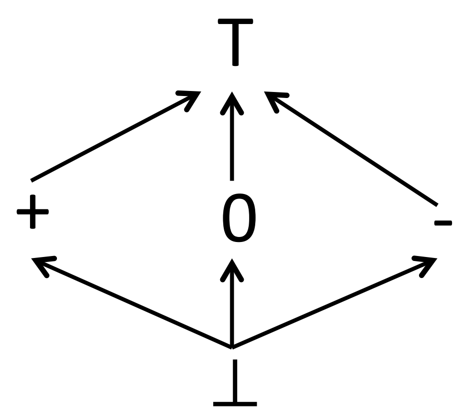
$$\bot \mathbin{\underline{+}} x = \bot$$

$$x \mathbin{\underline{\times}} \bot = \bot$$

$$\underline{-}\bot = \bot$$

The Abstract Domain

- Our abstract domain forms a **complete lattice**.
  - A partial order $x \leq y \Leftrightarrow \gamma(x) \subseteq \gamma(y)$
- 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



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.

$\alpha: 2^{Int} \longrightarrow A$

$\alpha(S) = lub(\{-|i<0 \wedge i\in S\}, \{0|0\in S\}, \{+|i>0 \wedge i\in S\})$

An Aside: Galois Connection

- (L, α, γ, M) is a Galois connection between complete lattices (L, ≤) and (M, ≤) iff:
  - α: L $\rightarrow$ M and γ: M $\rightarrow$ L are monotone functions

  Furthermore:
  - id ≤ γ o α
  - id ≤ α o γ

  - The function α o γ 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:

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

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

or

$id \leq \gamma \circ \alpha$

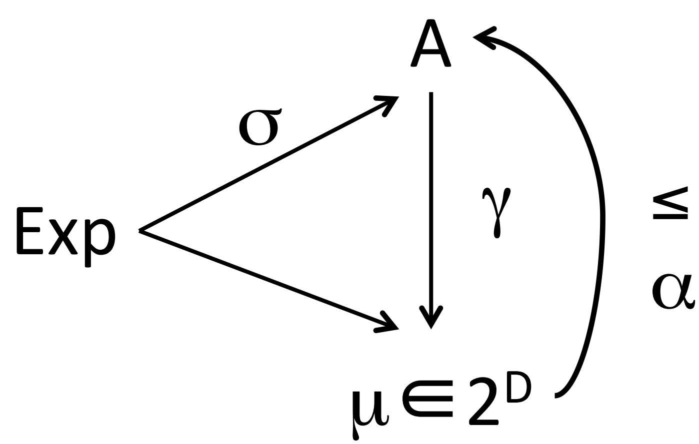
$id = \alpha \circ \gamma$

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.



General Conditions for Correctness

- Three conditions guarantee correctness in general:
  - α and γ form a Galois insertion
    - id $\leq \gamma$ o α, id $= \alpha$ o γ
  - α and γ are monotonic
    - $x \leq y \Rightarrow \alpha(x) \leq \alpha(y)$
  - Abstract operations <u>op</u> are locally correct:
    - $\gamma(\underline{op}(s_1,\ldots,s_n)) \subseteq op(\gamma(s_1),\ldots,\gamma(s_n))$

Generic Correctness Proof

- Proof by induction on the structure of e:
  $\mu(e) \in \gamma(\sigma(e))$

$\mu(e_1 \text{ op } e_2)$
$= \mu(e_1) \;\text{ op }\; \mu(e_2)$ [definition of μ]
$\in \gamma(\sigma(e_1)) \;\text{ op }\; \gamma(\sigma(e_2))$ [induction]
$\subseteq \gamma(\sigma(e_1) \;\underline{\text{op}}\; \sigma(e_2))$ [local correctness]
$= \gamma(\sigma(e_1 \;\text{ op }\; e_2))$ [definition of σ]

Another Notion of Correctness

- We can define correctness using abstraction instead of concretization.

$$\mu(e) \in \gamma(\sigma(e)) \Leftrightarrow \alpha(\{\mu(e)\}) \leq \sigma(e)$$

Proof for $\Rightarrow$ direction:

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

$\alpha(\{\mu(e)\}) \leq \alpha(\gamma(\sigma(e)))$ [monotonicity]

$\alpha(\{\mu(e)\}) \leq \sigma(e)$ [$\alpha \circ \gamma = id$]

Another Notion of Correctness

$$\mu(e) \in \gamma(\sigma(e)) \Leftrightarrow \alpha(\{\mu(e)\}) \leq \sigma(e)$$

Proof for $\Leftarrow$ direction:

$\alpha(\{\mu(e)\}) \leq \sigma(e)$

$\gamma(\alpha(\{\mu(e)\})) \leq \gamma(\sigma(e))$ [monotonicity]

$\mu(e) \in \gamma(\sigma(e))$ [id $\leq \gamma \circ \alpha$]

Extending Our Language

- Add input to the language
  - Modeled as a single free variable x in expressions

e = i | e * e | -e | e + e | … | **x**

Semantics

- The meaning function now has type

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

- We write the function with the expression as a subscript.

  $\mu_i(j) = i$

  $\mu_x(j) = j$

  $\mu_{e1*e2}(j) = \mu_{e1}(j) * \mu_{e2}(j)$

  $\mu_{e1+e2}(j) = \mu_{e1}(j) + \mu_{e2}(j)$

  …

Abstract Semantics

- Abstract semantic function:

  $\sigma : \text{Exp} \rightarrow A \rightarrow A$
- Also write this semantics in the same form.

$\sigma_i(\underline{j}) = \underline{i}$

$\sigma_x(\underline{j}) = \underline{j}$

$\sigma_{e1*e2}(\underline{j}) = \sigma_{e1}(\underline{j}) \; \underline{*} \; \sigma_{e2}(\underline{j})$

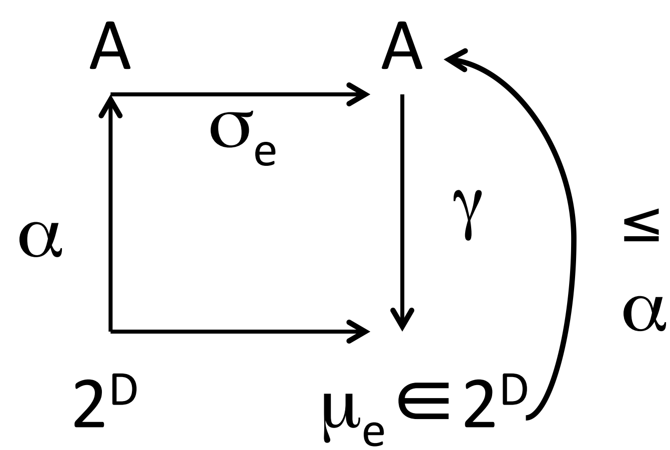
$\sigma_{e1+e2}(\underline{j}) = \sigma_{e1}(\underline{j}) \; \underline{+} \; \sigma_{e2}(\underline{j})$

… …

$\underline{i} = \alpha(\{i\})$

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.

$$op(\gamma(\sigma_{e1}(\underline{j})), \ldots , \gamma(\sigma_{en}(\underline{j}))) \quad \subseteq \quad \gamma(\underline{op}(\sigma_{e1}(\underline{j}), \ldots, \sigma_{en}(j)))$$

Proof of Correctness

- Theorem: $\mu_e(j) \in \gamma(\sigma_e(\underline{j}))$
- Proof (by induction on the structure of e):

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

$\mu_x(j) = j \in \gamma(\underline{j}) = \gamma\ (\sigma_x(\underline{j}))$

Induction on $\mu_{op(e1,\ldots,en)}(j)$:

$= op(\mu_{e1}(j), \ldots, \mu_{en}(j))$ [definition of μ]

$\in\ op(\gamma(\sigma_{e1}(\underline{j})), \ldots, \gamma(\sigma_{en}(\underline{j}))$ [induction]

$\subseteq\ \gamma\ (\underline{op}(\sigma_{e1}(\underline{j}), \ldots, \sigma_{en}(\underline{j})))$ [local correctness]

$=\ \gamma\ (\sigma_{op(e1,\ldots,en)}(\underline{j}))$ [definition of σ]

If-Then-Else

- e =... | if e = e then e else e |...

- $\mu_{\text{if e1=e2 then e3 else e4}}(i) = \mu_{e3}(i)$, if $\mu_{e1}(i) = \mu_{e2}(i)$

  $= \mu_{e4}(i)$, otherwise

- $\sigma_{\text{if e1=e2 then e3 else e4}}(\underline{i}) = \sigma_{e3}(\underline{i}) \sqcup \sigma_{e4}(\underline{i})$

- Recall that the abstract domain forms a complete lattice

Correctness of If-Then-Else

- Need to show that: $\mu_e(j) \in \gamma(\sigma_e(\underline{j}))$
  - Where e is an if-then-else
- Assume the true branch is taken.
- (The argument for the false branch is symmetric.)

$\mu_{e3}(i)$

$\in \gamma\, (\sigma_{e3}(\underline{i}))$ [by induction]

$\subseteq \gamma(\sigma_{e3}(\underline{i})) \sqcup \gamma(\sigma_{e4}(\underline{i}))$

$\subseteq \gamma(\sigma_{e3}(\underline{i}) \sqcup \sigma_{e4}(\underline{i}))$ [by monotonicity of γ]

Designing an Abstract Interpretation

- Define abstract domain
  - Needs to be a lattice
- Define the abstraction and concretization functions
  - σ : **D** $\rightarrow$ **A**
  - α: $2^D \rightarrow$ **A**
    - $\alpha(S) = lub(\sigma(s))$, for all $s \in S$
  - γ: **A** $\rightarrow 2^D$
- For every expression, define how to operate in the abstract domain