

10/1/02 CMSC 631 - Abstract Interpretation 1

---

---

---

---

---

---

---

---

- D: concrete domain
- P(D): powerset of concrete domain
- A: abstract domain
- $\llbracket \cdot \rrbracket$ : Exp  $\rightarrow$  D: meaning of expressions
- $\llbracket \cdot \rrbracket$ : Exp  $\rightarrow$  A: abstract meaning of expressions
- $\llbracket \cdot \rrbracket$ : A  $\rightarrow$  P(D): concretization function
- $\llbracket \cdot \rrbracket$ : P(D)  $\rightarrow$  A: abstraction function
- $\text{id} \leq \llbracket \cdot \rrbracket \circ \llbracket \cdot \rrbracket$
- $\text{id} = \llbracket \cdot \rrbracket \circ \llbracket \cdot \rrbracket$

10/1/02 CMSC 631 - Abstract Interpretation 2

---

---

---

---

---

---

---

---

## Abstract interpretation

- Consider a tiny language with only integers and multiplication
  - $\llbracket \cdot \rrbracket$ : Exp  $\rightarrow$  D: meaning of expressions
  - $e := i \mid e * e$
  - $\llbracket i \rrbracket = i$
  - $\llbracket (e_1 * e_2) \rrbracket = \llbracket e_1 \rrbracket * \llbracket e_2 \rrbracket$

10/1/02 CMSC 631 - Abstract Interpretation 3

---

---

---

---

---

---

---

---

## An abstraction

- Compute only the sign of the result
  - $\llbracket \cdot \rrbracket$ : Exp  $\rightarrow$  A: abstract meaning of expressions

$$\llbracket i \rrbracket = \begin{cases} + & \text{if } i > 0 \\ 0 & \text{if } i = 0 \\ - & \text{if } i < 0 \end{cases}$$

$$\llbracket (e_1 * e_2) \rrbracket = \llbracket (e_1) \rrbracket \llbracket (e_2) \rrbracket$$

*	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

10/1/02

CMSC 631 - Abstract Interpretation

4

---

---

---

---

---

---

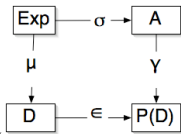
---

---

## Soundness

- Associate with each abstract value the set of concrete values it represents
  - $\llbracket \cdot \rrbracket$  A  $\rightarrow$  P(D): concretization function
  - $\llbracket + \rrbracket = \{ i \mid i > 0 \}$
  - $\llbracket 0 \rrbracket = \{ 0 \}$
  - $\llbracket - \rrbracket = \{ i \mid i < 0 \}$

- $\llbracket (e) \rrbracket \subseteq \llbracket \llbracket (e) \rrbracket \rrbracket$



10/1/02

CMSC 631 - Abstract Interpretation

5

---

---

---

---

---

---

---

---

## Abstract interpretation

- Computation in an abstract domain
  - In this case,  $\{+, 0, -\}$
- The abstract semantics is sound
  - approximates the actual semantics
- The concretization function establishes the connection between the domains

10/1/02

CMSC 631 - Abstract Interpretation

6

---

---

---

---

---

---

---

---

## Adding plus

- Can't limit ourselves to +, 0 or -

+	+	0	□
+	+	+	?
0	+	0	□
□	?	□	□

- Add □ (top) to represent any integer

10/1/02

CMSC 631 - Abstract Interpretation

7

---

---

---

---

---

---

---

---

## With Top

+	+	0	□	□
+	+	+	□	□
0	+	0	□	□
□	□	□	□	□
□	□	□	□	□

*	+	0	□	□
+	+	0	□	□
0	0	0	0	0
□	□	0	+	□
□	□	0	□	□

10/1/02

CMSC 631 - Abstract Interpretation

8

---

---

---

---

---

---

---

---

## Finding the right abstractions is a key design problem

- Want things to be as accurate as possible
  - while still allowing the analysis to be feasible

10/1/02

CMSC 631 - Abstract Interpretation

9

---

---

---

---

---

---

---

---

## What about integer division

- What is  $+ / 0$ ?
- Use  $\perp$  (bottom) to represent the empty set of answers
  - in this case, all operations are strict in bottom

$\perp$	+	0	$\perp$	$\perp$	$\perp$
+	+	+	$\perp$	$\perp$	$\perp$
0	+	0	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

$\perp$	/	+	0	$\perp$	$\perp$	$\perp$
+	+	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	0	$\perp$	0	0	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	+	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

10/1/02 CMSC 631 - Abstract Interpretation 10

---

---

---

---

---

---

---

---

---

---

## The abstract domain is a lattice

- A partial order, with a top and a bottom
- For all  $x$  and  $y$  in  $A$ 
  - $x \leq y$  implies  $\perp(x) \perp(y)$

10/1/02 CMSC 631 - Abstract Interpretation 11

---

---

---

---

---

---

---

---

---

---

## The abstraction function

- $\perp$  maps a set of concrete values into an abstract value
  - $\perp: P(D) \rightarrow A$ : abstraction function
- The abstraction function  $\perp$  divides the concrete domain into subsets (not disjoint)
- $\perp$  maps a subset of the domain into the smallest containing abstract value

10/1/02 CMSC 631 - Abstract Interpretation 12

---

---

---

---

---

---

---

---

---

---

## Galois insertion

- $\alpha$   $x$  in  $P(D)$ ,  $x \sqsubseteq \alpha(\alpha(x))$ 
  - equivalently,  $\text{id} \sqsubseteq \alpha \circ \alpha$
- $\beta$   $a$  in  $A$ ,  $a = \beta(\alpha(a))$ 
  - equivalently,  $\text{id} = \beta \circ \alpha$

10/1/02

CMSC 631 - Abstract Interpretation

13

---

---

---

---

---

---

---

---

## Conditions for correctness

- $\alpha$  and  $\beta$  form a Galois insertion
  - $\text{id} \sqsubseteq \beta \circ \alpha$
  - $\text{id} = \beta \circ \alpha$
- $\alpha$  and  $\beta$  are monotonic
  - $x \sqsubseteq y$  implies  $\alpha(x) \leq \alpha(y)$
- Abstract operations are locally correct
  - $\text{op}(\alpha(a_1), \alpha(a_2), \dots, \alpha(a_n)) \sqsubseteq \alpha(\text{op}(a_1, a_2, \dots, a_n))$

10/1/02

CMSC 631 - Abstract Interpretation

14

---

---

---

---

---

---

---

---

## Correctness proof

- Proof by induction on  $e$ :  $\alpha(e) \sqsubseteq \beta(\alpha(e))$ 
  - $\alpha(e_1 \text{ op } e_2)$
  - $= \alpha(e_1) \text{ op } \alpha(e_2)$
  - $\sqsubseteq \alpha(\alpha(e_1)) \text{ op } \alpha(\alpha(e_2))$
  - $\sqsubseteq \alpha(\alpha(e_1) \text{ op } \alpha(e_2))$
  - $= \alpha(\alpha(e_1 \text{ op } e_2))$

10/1/02

CMSC 631 - Abstract Interpretation

15

---

---

---

---

---

---

---

---

## Correctness using abstraction function

- $\llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket \equiv \llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$
- **Proof**  $\llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket \sqsubseteq \llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$ 
  - $\llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket$
  - $\llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket$     **monotonicity**
  - $\llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$     **id =  $\alpha \circ \alpha$**

10/1/02

CMSC 631 - Abstract Interpretation

16

---

---

---

---

---

---

---

---

## Correctness using abstraction function

- $\llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket \equiv \llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$
- **Proof**  $\llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket \sqsubseteq \llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$ 
  - $\llbracket \{\llbracket e \rrbracket\} \rrbracket \sqsubseteq \llbracket e \rrbracket$
  - $\llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket \sqsubseteq \llbracket \{\llbracket e \rrbracket\} \rrbracket$     **monotonicity**
  - $\llbracket e \rrbracket \sqsubseteq \llbracket \{\llbracket \alpha(e) \rrbracket\} \rrbracket$     **id =  $\alpha \circ \alpha$**
  - $\llbracket e \rrbracket \sqsubseteq \llbracket \alpha(e) \rrbracket$

10/1/02

CMSC 631 - Abstract Interpretation

17

---

---

---

---

---

---

---

---

## Language with input

- So far, just constant expressions
- Add free variable  $x$
- $e ::= i \mid e+e \mid e*e \mid \dots \mid x$
- $\llbracket \cdot \rrbracket$ :  $\text{Exp} \rightarrow \text{Int} \rightarrow \text{Int}$ 
  - $\llbracket_i \rrbracket(j) = i$
  - $\llbracket_x \rrbracket(j) = j$
  - $\llbracket_{e_1 * e_2} \rrbracket(j) = \llbracket_{e_1} \rrbracket(j) * \llbracket_{e_2} \rrbracket(j)$

10/1/02

CMSC 631 - Abstract Interpretation

18

---

---

---

---

---

---

---

---

## Abstract semantics

- $\llbracket \cdot \rrbracket: \text{Exp} \rightarrow A \rightarrow A$ 
  - $\llbracket_i(j) = \llbracket(\{i\})$
  - $\llbracket_x(j) = j$
  - $\llbracket_{e_1 * e_2}(j) = \llbracket_{e_1}(j) * \llbracket_{e_2}(j)$

10/1/02

CMSC 631 - Abstract Interpretation

19

---

---

---

---

---

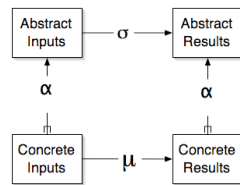
---

---

---

## Correctness

- $\llbracket_i, \llbracket_e(i) \sqsubseteq \llbracket_{\llbracket_e(\llbracket(\{i\}))}$
- $\llbracket_e \leq \llbracket_e \circ \llbracket_e \circ \llbracket$
- $\llbracket_e \circ \llbracket_e \leq \llbracket_e \circ \llbracket$



10/1/02

CMSC 631 - Abstract Interpretation

20

---

---

---

---

---

---

---

---

## If Then Else

- $e := \dots \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \mid \dots$
- $\llbracket_{\text{if } e_1 \text{ then } e_2 \text{ else } e_3}(i)$ 
  - =  $\llbracket_{e_3}(i)$  if  $\llbracket_{e_1}(i) = 0$
  - =  $\llbracket_{e_2}(i)$  otherwise
- $\llbracket_{\text{if } e_1 \text{ then } e_2 \text{ else } e_3}(i)$ 
  - = least upper bound( $\llbracket_{e_2}(i), \llbracket_{e_3}(i)$ )

10/1/02

CMSC 631 - Abstract Interpretation

21

---

---

---

---

---

---

---

---

## Recursion

- Add recursive function definitions
  - of a single function, for simplicity
- $\text{program} := \text{def } f(x) = e$
- $e := \dots \mid f(e)$
- $\square: \text{Exp} \rightarrow \text{Int} \rightarrow \text{Int}_{\square}$
- $\square': \text{Exp} \rightarrow (\text{Int} \rightarrow \text{Int}_{\square}) \rightarrow \text{Int} \rightarrow \text{Int}_{\square}$

10/1/02

CMSC 631 - Abstract Interpretation

22

---

---

---

---

---

---

---

---

## $\square'$

- $\square'_{f(e)}(g)(j) = g(\square'_e(g)(j))$
- $\square'_x(g)(j) = j$
- $\square'_{e_1+e_2}(g)(j) = \square'_{e_1}(g)(j) + \square'_{e_2}(g)(j)$

10/1/02

CMSC 631 - Abstract Interpretation

23

---

---

---

---

---

---

---

---

## Meaning of recursive functions

- $\square: \text{Exp} \rightarrow \text{Int} \rightarrow \text{Int}_{\square}$
- $\square': \text{Exp} \rightarrow (\text{Int} \rightarrow \text{Int}_{\square}) \rightarrow \text{Int} \rightarrow \text{Int}_{\square}$
- Consider a program  $\text{def } f = e$
- Define an ascending chain  $f_0, f_1, \dots$  in  $\text{Int} \rightarrow \text{Int}_{\square}$ 
  - $f_0 = \square_x \square$
  - $f_{i+1} = \square'_e(f_i)$
- Define  $\square_{\text{def } f = e} = \square_i f_i$

10/1/02

CMSC 631 - Abstract Interpretation

24

---

---

---

---

---

---

---

---