

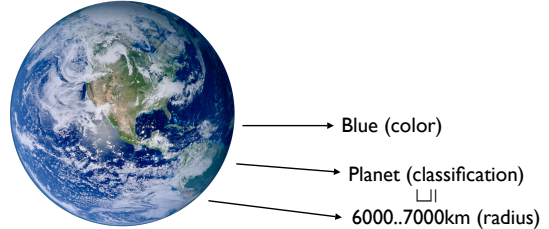
CMSC 631 – Program Analysis and Understanding Fall 2004

Abstract Interpretation

Based on lectures by David Schmidt and by Alex Aiken

What is an Abstraction?

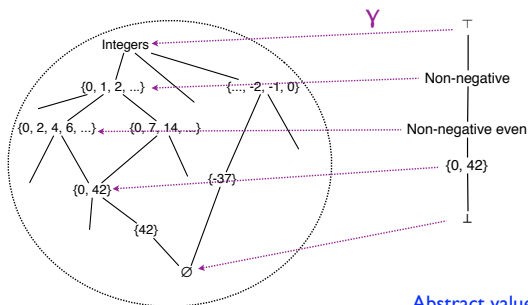
- A property from some domain



CMSC 631

2

Example Abstraction



Concrete values: sets of integers

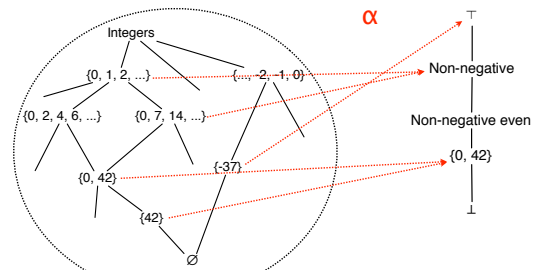
Abstract values

Concretization function γ maps each abstract value to concrete values it represents

CMSC 631

3

Abstraction is Imprecise



Concrete values: sets of integers

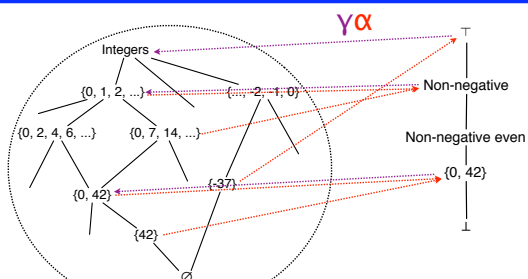
Abstract values

Abstraction function α maps each concrete set to the best abstract value

CMSC 631

4

Composing α and γ



Concrete values: sets of integers

Abstract values

Abstraction followed by concretization is sound but imprecise

CMSC 631

5

α and γ Form a Galois Insertion

- α and γ are monotonic
 - Recall: f is monotonic if $x \leq y \Rightarrow f(x) \leq f(y)$
 - Also called "order preserving"
- $S \subseteq \gamma(\alpha(S))$ for any concrete set S
- $\alpha(\gamma(A)) = A$ for any abstract element A
- Next up: Abstract interpretation in action
 - We'll develop an abstract interpretation of a simple language and prove it correct using these ideas

CMSC 631

6

Source Language

- Integers and multiplication
 - $e ::= i \mid e * e$
- Standard semantics of the program
 - $\text{Eval} : e \rightarrow \text{Int}$
 - $\text{Eval}(i) = i$
 - $\text{Eval}(e_1 * e_2) = \text{Eval}(e_1) \times \text{Eval}(e_2)$

CMSC 631

7

Abstraction

- Define an abstract semantics that computes only the sign of the result

\times	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

$$\text{AEval} : e \rightarrow \{-, 0, +\}$$

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

$$\text{AEval}(e_1 * e_2) = \text{AEval}(e_1) \times \text{AEval}(e_2)$$

CMSC 631

8

Soundness

- We can show our abstraction correctly predicts the sign of an expression
- Proof: by structural induction on e
 - $\text{Eval}(e) > 0$ iff $\text{AEval}(e) = +$
 - $\text{Eval}(e) = 0$ iff $\text{AEval}(e) = 0$
 - $\text{Eval}(e) < 0$ iff $\text{AEval}(e) = -$

CMSC 631

9

Another Approach to Soundness

- Natural concretization function

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

$$\gamma(0) = \{0\}$$

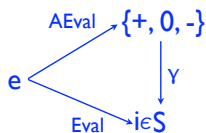
$$\gamma(-) = \{i \mid i < 0\}$$

- Note: This presentation is slightly non-standard
 - Usually defined in terms of execution traces

CMSC 631

10

Soundness (cont'd)



- Our abstraction is sound if
 - $\text{Eval}(e) \in \gamma(\text{AEval}(e))$
- Soundness proof: later

CMSC 631

11

Adding Unary Negation

- $e ::= i \mid e * e \mid -e$
- $\text{Eval}(-e) = -\text{Eval}(e)$
- $\text{AEval}(e) = \neg \text{AEval}(e)$

\neg	+	0	-
	-	0	+

- No problems

CMSC 631

12

Adding Addition

- $e ::= i \mid e * e \mid -e \mid e + e$
- $\text{Eval}(e1 + e2) = \text{Eval}(e1) + \text{Eval}(e2)$
- $\text{AEval}(e1 + e2) = \text{AEval}(e1) \pm \text{AEval}(e2)$

\pm	+	0	-
+	+	+	?
0	+	0	-
-	?	-	-

Our abstract domain is not closed under addition

Solution

- Add an abstract value to represent any integer
- Finding closed domain often key design problem

$\Upsilon(\top) = \{\text{integers}\}$

\pm	+	0	-	\top
+	+	+	\top	\top
0	+	0	-	\top
-	\top	-	-	\top
\top	\top	\top	\top	\top

- Other operations also need to handle \top

Two Ways to Lose Information

- OK: Abstraction still precise enough
 - $\text{Eval}((5 * 5) + 6) = 31$
 - $\text{AEval}((5*5) + 6) = (+ \ x \ +) \ \pm \ + \ = \ +$
 - Abstractly, we don't know which value we computed
 - ...but we don't care, since we only want the sign
- Not so good: "Don't know" values
 - $\text{Eval}((1 + 2) + -3) = 0$
 - $\text{AEval}((1 + 2) + -3) = (+ \ \pm \ +) \ \pm \ - \ = \ + \ \pm \ - \ = \ \top$
 - We also don't know which value we computed
 - ...and we can't even figure out its sign

Adding Integer Division

- What happens when we divide by zero?
 - The result is not an integer (it's undefined)
 - If we divide each integer in a set by 0, the result is the empty set

$\Upsilon(\perp) = \emptyset$

\pm	+	0	-	\top	\perp
+	+	0	-	\top	\perp
0	\perp	\perp	\perp	\perp	\perp
-	-	0	+	\top	\perp
\top	\top	0	\top	\top	\perp
\perp	\perp	\perp	\perp	\perp	\perp

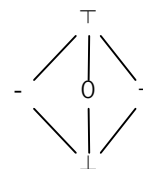
Adding Integer Division (cont'd)

- We need to extend other abstract operations to work on \perp
- Every operation involving \perp results in \perp
 - All operations are *strict* in \perp

$\perp \times a$	=	\perp
$a \times \perp$	=	\perp
$\perp \pm a$	=	\perp
$a \pm \perp$	=	\perp
$-\perp$	=	\perp

The Abstract Domain

- Look, Ma, a lattice!
- We've got:
 - A set of elements $\{\perp, +, 0, -, \top\}$
 - A relation \leq that is
 - Reflexive
 - Anti-symmetric
 - Transitive
 - And
 - The least upper bound (lub, \sqcup) and greatest lower bound (glb, \sqcap) exists for any pair of elements
 - So it's a lattice



Abstraction and Concretization

- Concretization function γ

$$\begin{aligned} \gamma(\top) &= \text{all integers} \\ \gamma(+)&= \{i \mid i > 0\} \\ \gamma(0) &= \{0\} \\ \gamma(-) &= \{i \mid i < 0\} \\ \gamma(\perp) &= \emptyset \end{aligned}$$

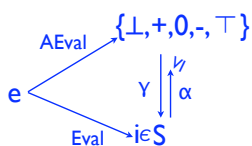
- Abstraction function maps concrete values (sets of integers) to smallest valid abstract element

$$\alpha(S) = \left\{ \begin{array}{l} - \exists i \in S. i < 0 \\ \perp \text{ otherwise} \end{array} \right\} \sqcup \left\{ \begin{array}{l} 0 \exists i \in S. i = 0 \\ \perp \text{ otherwise} \end{array} \right\} \sqcup \left\{ \begin{array}{l} + \exists i \in S. i > 0 \\ \perp \text{ otherwise} \end{array} \right\}$$

Definition

- An abstract interpretation consists of
 - A concrete domain S and an abstract domain A
 - Concretization and abstraction functions that form a Galois insertion [of A into S]
 - A (sound) abstract semantic function
- Recall: α and γ form a Galois insertion if
 - α and γ are monotone
 - $S \subseteq \gamma(\alpha(S))$ or $\text{id} \leq \gamma\alpha$ for any concrete set S
 - $A = \alpha(\gamma(A))$ or $\text{id} = \alpha\gamma$ for any abstract element A

Soundness, Again



- Our abstraction is sound if
 - $\text{Eval}(e) \in \gamma(\text{AEval}(e))$
- Soundness proof: next

Conditions for Correctness

- We can show that if
 - α and γ form a Galois insertion
 - And abstract operations op are locally correct
 - $\gamma(\text{op}(a_1, \dots, a_n)) \supseteq \text{op}(\gamma(a_1), \dots, \gamma(a_n))$
 - Note: We've extended op pointwise to sets
 - I.e., if S and T are sets, $S+T = \{s+t \mid s \in S, t \in T\}$
- Then the abstract interpretation is sound

Proof: Show $\text{Eval}(e) \in \gamma(\text{AEval}(e))$

- By structural induction on expressions
 - Base cases: an integer i , so $\text{Eval}(i) = i$
 - if $i < 0$ then $\gamma(\text{AEval}(i)) = \gamma(-) = \{j \mid j < 0\}$
 - Other cases similar
 - Induction: for any operation
 - $\text{Eval}(e_1 \text{ op } e_2)$
 - $= \text{Eval}(e_1) \text{ op } \text{Eval}(e_2)$ by definition of Eval
 - $\in \gamma(\text{AEval}(e_1)) \text{ op } \gamma(\text{AEval}(e_2))$ by induction
 - $\subseteq \gamma(\text{AEval}(e_1) \text{ op } \text{AEval}(e_2))$ by local correctness of op
 - $= \gamma(\text{AEval}(e_1 \text{ op } e_2))$ by definition of AEval

Another Proof of Correctness

- We can define correctness in terms of abstraction rather than concretization
 - $\text{Eval}(e) \in \gamma(\text{AEval}(e))$ iff $\alpha(\{\text{Eval}(e)\}) \leq \text{AEval}(e)$
- Equivalence proof:
 - (\Rightarrow) Assume $\text{Eval}(e) \in \gamma(\text{AEval}(e))$
 - I.e., $\{\text{Eval}(e)\} \subseteq \gamma(\text{AEval}(e))$
 - Then $\alpha(\{\text{Eval}(e)\}) \leq \alpha(\gamma(\text{AEval}(e)))$ by monotonicity
 - And $\alpha(\{\text{Eval}(e)\}) \leq \text{AEval}(e)$ since $\text{id} = \alpha\gamma$

Correctness Proof (cont'd)

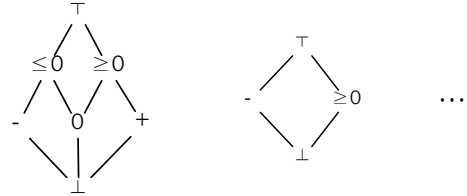
- Showing
 - $\text{Eval}(e) \in \gamma(\text{AEval}(e))$ iff $\alpha(\{\text{Eval}(e)\}) \leq \text{AEval}(e)$
 - (\Leftarrow) Assume $\alpha(\{\text{Eval}(e)\}) \leq \text{AEval}(e)$
 - Then $\gamma(\alpha(\{\text{Eval}(e)\})) \subseteq \gamma(\text{AEval}(e))$ by monotonicity
 - Then $\{\text{Eval}(e)\} \subseteq \gamma(\text{AEval}(e))$ since $\text{id} \leq \gamma\alpha$
 - I.e., $\text{Eval}(e) \in \gamma(\text{AEval}(e))$

CMSC 631

25

An Alternate Abstract Domain

- That domain wasn't the only choice, of course



- The right domain depends on the problem we're trying to solve

CMSC 631

26

Relationship to Data Flow Analysis

- Abstract interpretation was invented partially to find a firm semantic foundation for data flow analysis
 - Precise relationship between concrete domain (program executions) and abstract domain (data flow facts)
 - Generic correctness proof
- Caveat: Data flow typically uses meet, abstract interpretation typically uses join

CMSC 631

27

Acceleration: Widening

- Given monotone transfer functions
 - Finite height lattice \Rightarrow termination
- What if
 - Height is finite but large?
 - Height is infinite
- "Solution": Widening
 - Every so often, replace A by A^*A
 - This is safe (conservative, sound)
 - But apply when? where?

CMSC 631

28

Limitations

- Focus is on correctness
 - Not much insight into efficient algorithms
- Theory is completely general
 - What are good choices for modeling data structures and the heap? Higher-order functions? Objects?
- Forwards vs. backwards distinction
 - Permeates literature on abstract interpretation
 - But theory doesn't require it

CMSC 631

29

Conclusions

- Cousot and Cousot paper(s) seminal work(s)
- The *theory* of abstract interpretation is often confused with using it to construct tool (e.g., data flow analysis)
- Slogan:
 - Finite lattices + monotonic functions = program analysis

CMSC 631

30