COMP 150-AVS

Abstract Interpretation

Based on lectures by David Schmidt and by Alex Aiken

What is an Abstraction?

- A property from some domain

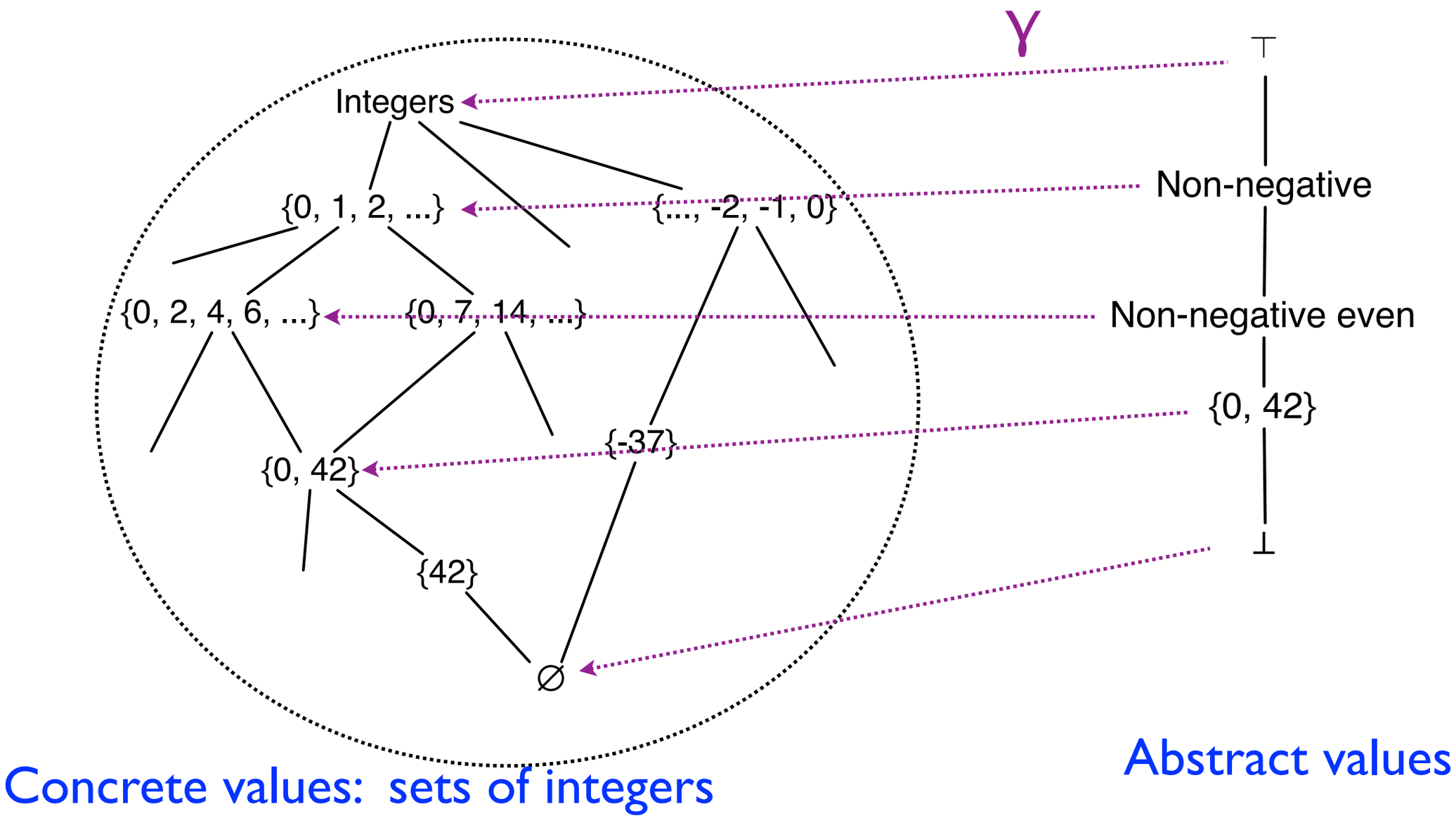
Blue (color)

Planet (classification)

$\sqsubseteq$

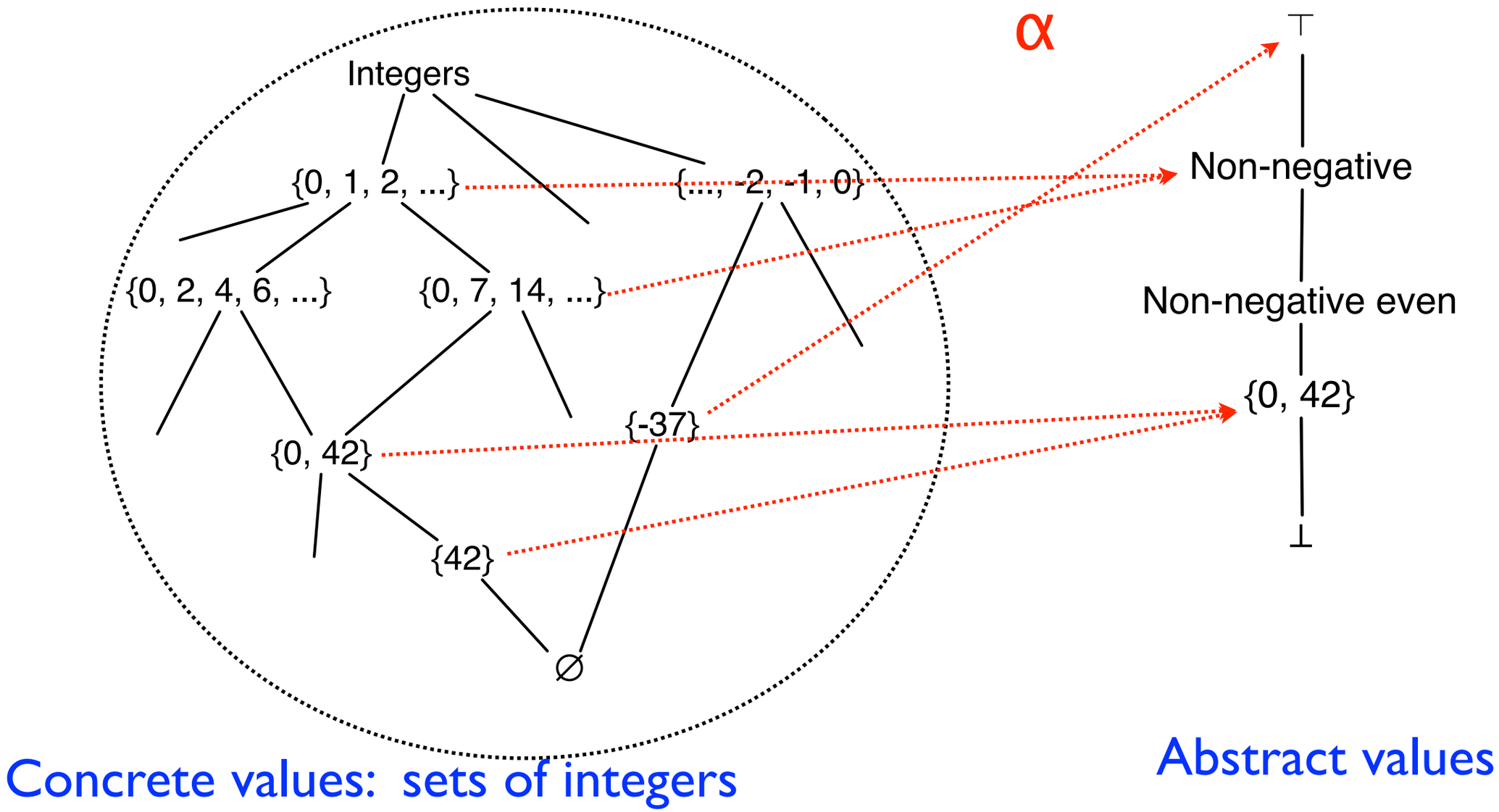
6000..7000km (radius)

Example Abstraction



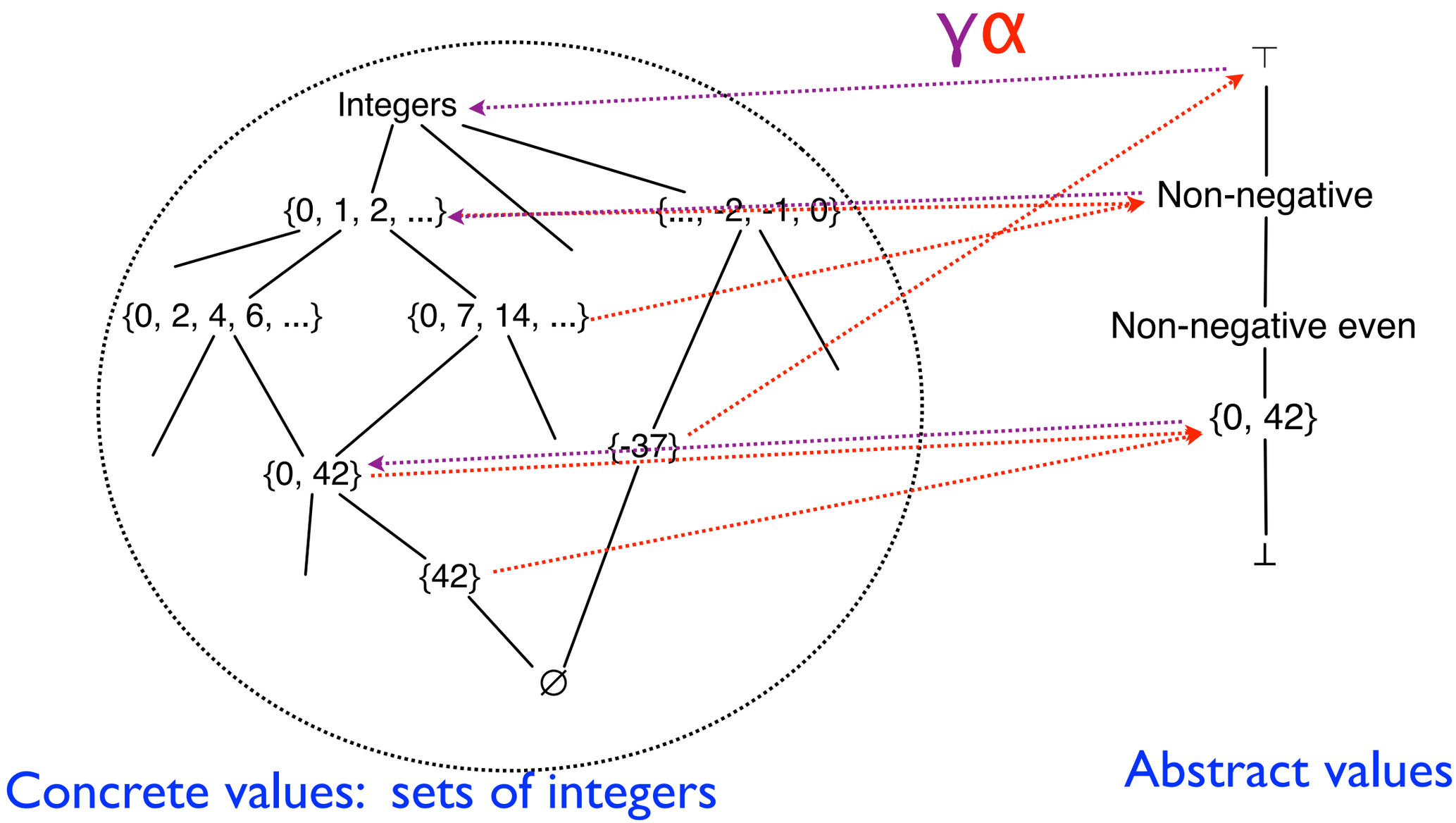
Concretization function γ maps each abstract value to concrete values it represents

Abstraction is Imprecise



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

Composing α and γ



Abstraction followed by concretization is sound but imprecise

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

Source Language

- Integers and multiplication
  - e ::= i | e * e

- Standard semantics of the program
  - Eval : e → Int
  - Eval(i) = i
  - Eval(e1*e2) = Eval(e1) × Eval(e2)

Abstraction

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

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

AEval : e → {-, 0, +}

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

AEval(e1*e2) = AEval(e1) $\underline{\times}$ AEval(e2)

Soundness

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

Another Approach to Soundness

- Natural concretization function

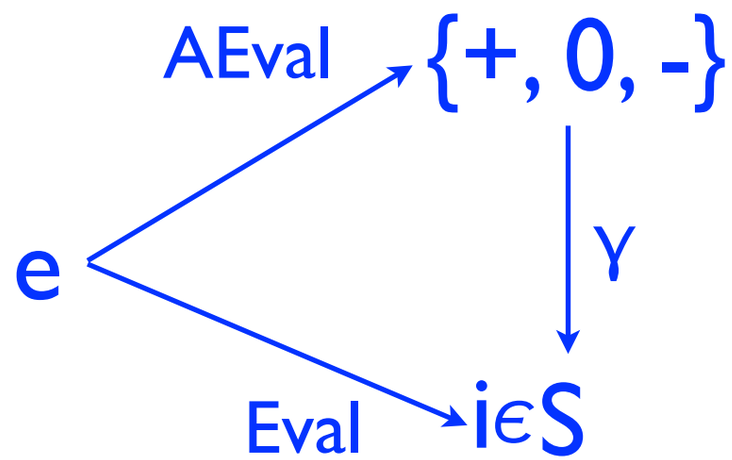
$$\gamma(+) = \{i \mid i>0\}$$

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

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

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

Soundness (cont'd)



- Our abstraction is sound if
  - Eval(e) ∈ γ(AEval(e))
- Soundness proof: later

Adding Unary Negation

- e ::= i | e * e | -e

- Eval(-e) = -Eval(e)
- AEval(e) = -AEval(e)

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

- No problems

Adding Addition

- e ::= i | e * e | -e | e + e

- Eval(e1+e2) = Eval(e1) + Eval(e2)
- AEval(e1+e2) = AEval(e1) $\underline{+}$ AEval(e2)

$\underline{+}$	+	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

$\gamma(\top)$ = {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
  - Eval((5 * 5) + 6) = 31
  - AEval((5*5) + 6) = (+ $\underline{\times}$ +) $\underline{+}$ + = +
    - 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
  - Eval((1 + 2) + -3) = 0
  - AEval((1 + 2) + -3) = (+ $\underline{+}$ +) $\underline{+}$ - = + $\underline{+}$ - = $\top$
    - We also don't know which value we computed

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

$\gamma(\bot) = \varnothing$

$\div$	+	0	-	$\top$	$\bot$
+	+	0	-	$\top$	$\bot$
0	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
-	-	0	+	$\top$	$\bot$
$\top$	$\top$	0	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

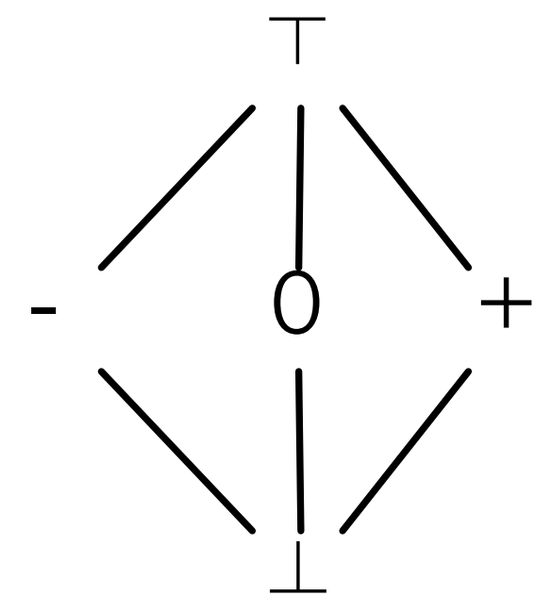
Adding Integer Division (cont'd)

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

$$
\begin{aligned}
\perp \underline{\times} a &= \perp \\
a \underline{\times} \perp &= \perp \\
\perp \underline{+} a &= \perp \\
a \underline{+} \perp &= \perp \\
\underline{-}\perp &= \perp
\end{aligned}
$$

The Abstract Domain

- Look, Ma, a lattice!
- We've got:
  - A set of elements $\{\bot, +, 0, -, \top\}$
  - 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

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

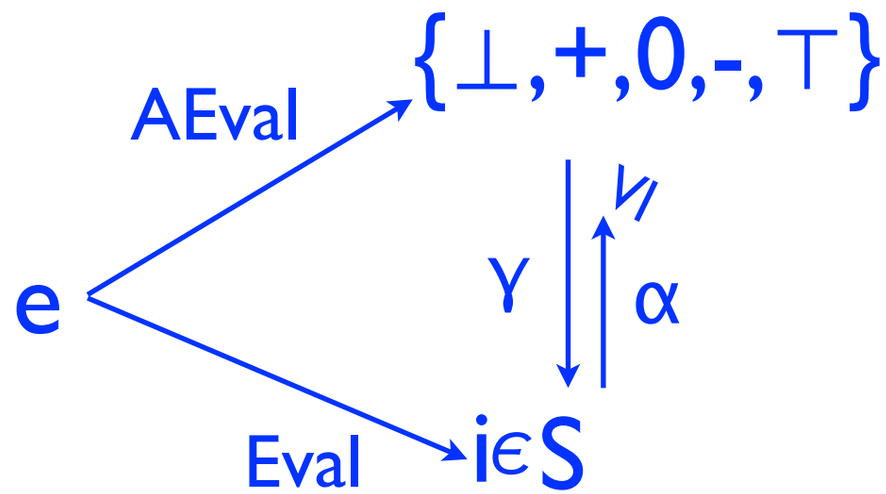
- Abstraction function maps concrete values (sets of integers) to smallest valid abstract element
  - $\alpha(S) = \left\{ \begin{array}{ll} - & \exists i \in S \,.\, i<0 \\ \bot & \text{otherwis} \end{array} \right\} \sqcup \left\{ \begin{array}{ll} 0 & \exists i \in S \,.\, i=0 \\ \bot & \text{otherwise} \end{array} \right\} \sqcup \left\{ \begin{array}{ll} + & \exists i \in S \,.\, i>0 \\ \bot & \text{otherwise} \end{array} \right\}$

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 $id \leq \gamma\alpha$ for any concrete set S
  - $A = \alpha(\gamma(A))$ or $id = \alpha\gamma$ for any abstract element A

Soundness, Again



- Our abstraction is sound if
  - Eval(e) ∈ γ(AEval(e))
- Soundness proof: next

Conditions for Correctness

- We can show that if
  - α and γ form a Galois insertion
  - And abstract operations $\underline{op}$ are locally correct
    - $\gamma(\underline{op}(a1, ..., an)) \supseteq op(\gamma(a1), ..., \gamma(an))$
    - 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 Eval(e) $\in$ γ(AEval(e))

- By structural induction on expressions
  - Base cases: an integer i, so Eval(i) = i
    - if i < 0 then γ(AEval(i)) = γ(-) = {j | j < 0}
    - Other cases similar
  - Induction: for any operation

Eval(e1 op e2)	
= Eval(e1) op Eval(e2)	by definition of Eval
$\in$ γ(AEval(e1)) op γ(AEval(e2))	by induction
$\subseteq$ γ(AEval(e1) <u>op</u> AEval(e2))	by local correctness of <u>op</u>
= γ(AEval(e1 op e2))	by definition of AEval

Another Proof of Correctness

- We can define correctness in terms of abstraction rather than concretization
  - $Eval(e) \in \gamma(AEval(e))$ iff $\alpha(\{Eval(e)\}) \leq AEval(e)$

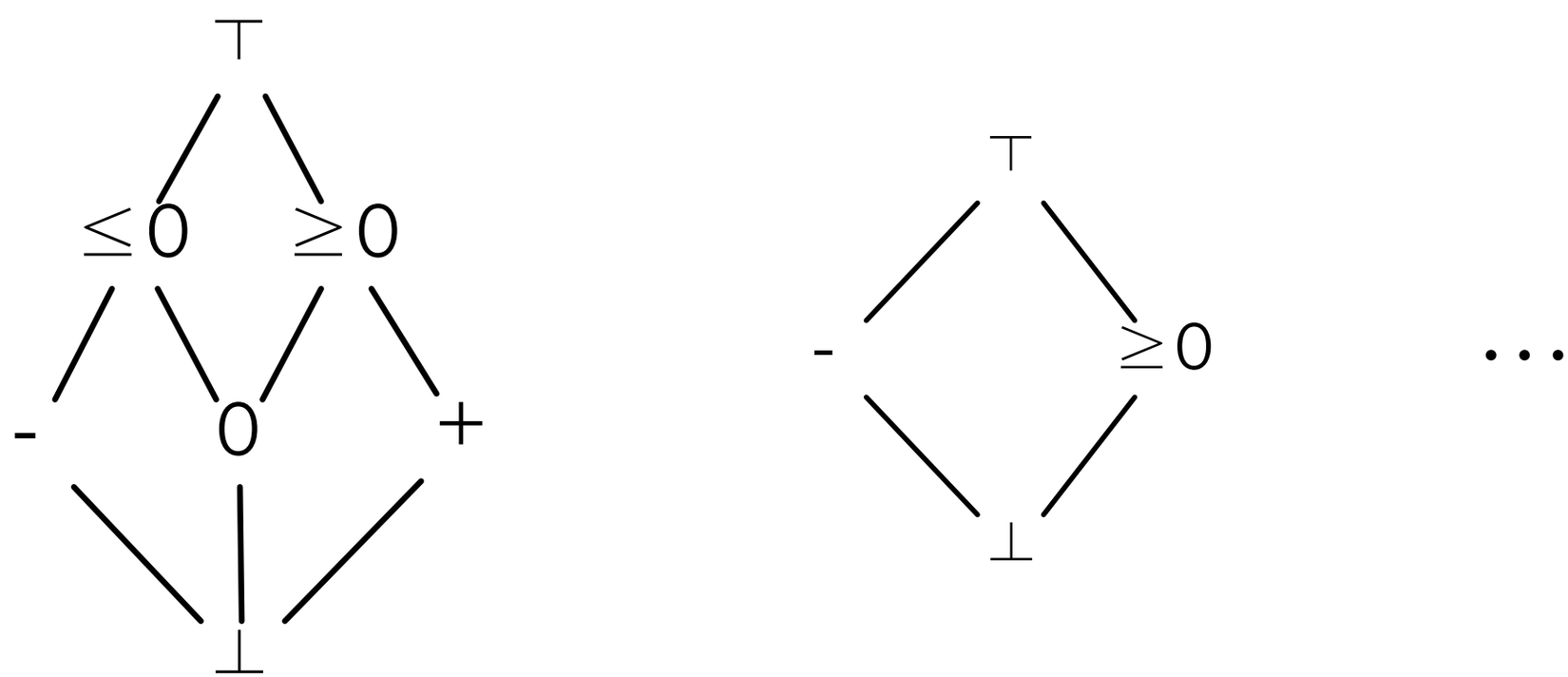
- Equivalence proof:
  - ($\Rightarrow$) Assume $Eval(e) \in \gamma(AEval(e))$
  - I.e., $\{Eval(e)\} \subseteq \gamma(AEval(e))$
  - Then $\alpha(\{Eval(e)\}) \leq \alpha(\gamma(AEval(e)))$ by monotonicity
  - And $\alpha(\{Eval(e)\}) \leq AEval(e)$ since $id = \alpha\gamma$

Correctness Proof (cont'd)

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

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

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

Acceleration: Widening

- Given monotone transfer functions
  - Finite height lattice ⇒ 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)

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

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