CMSC 631

Program Analysis and Understanding

Spring 2013

Abstract interpretation

What is an Abstraction?

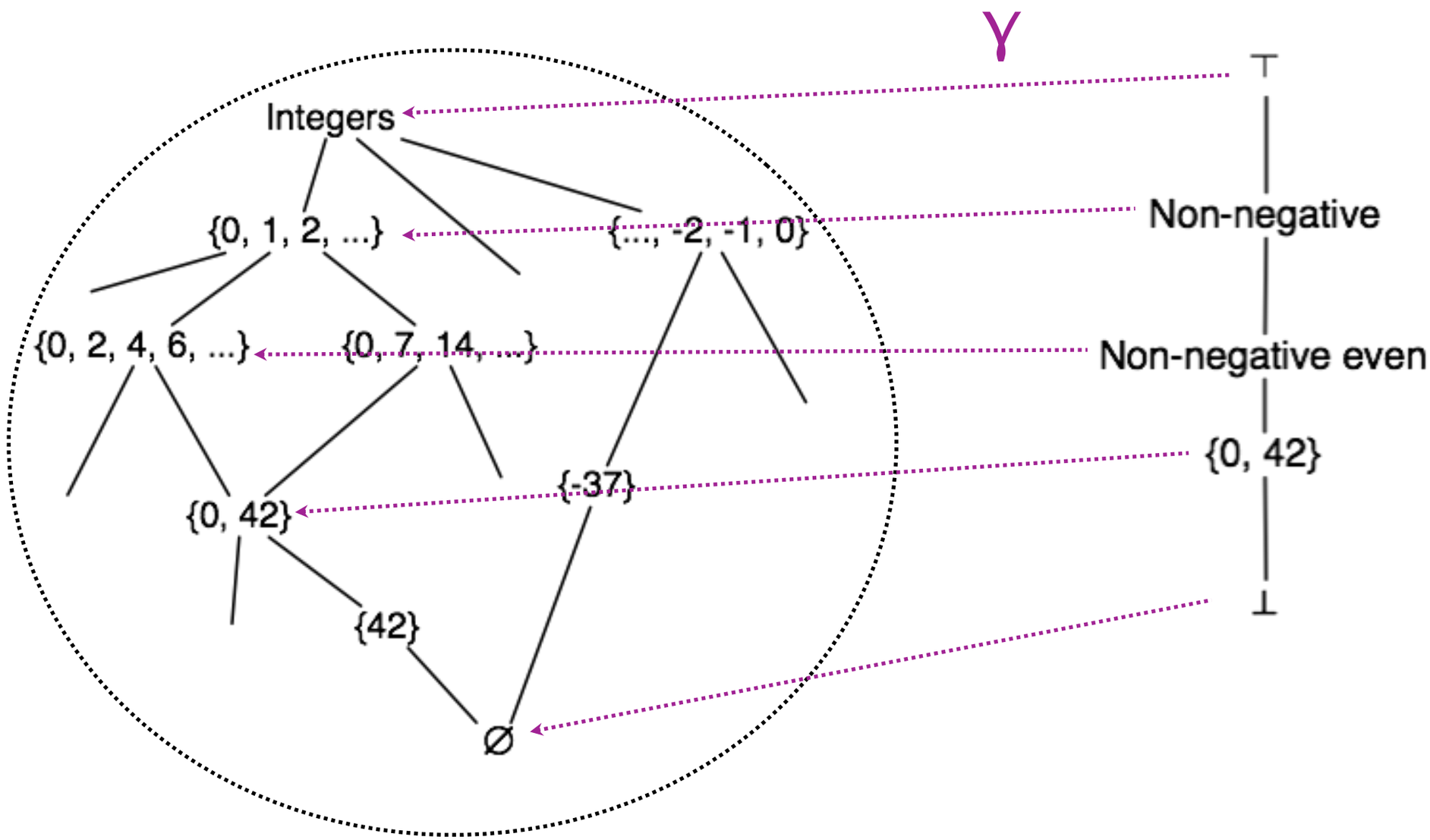
- A property from some domain

Blue (color)

Planet (classification)

6000..7000km (radius)

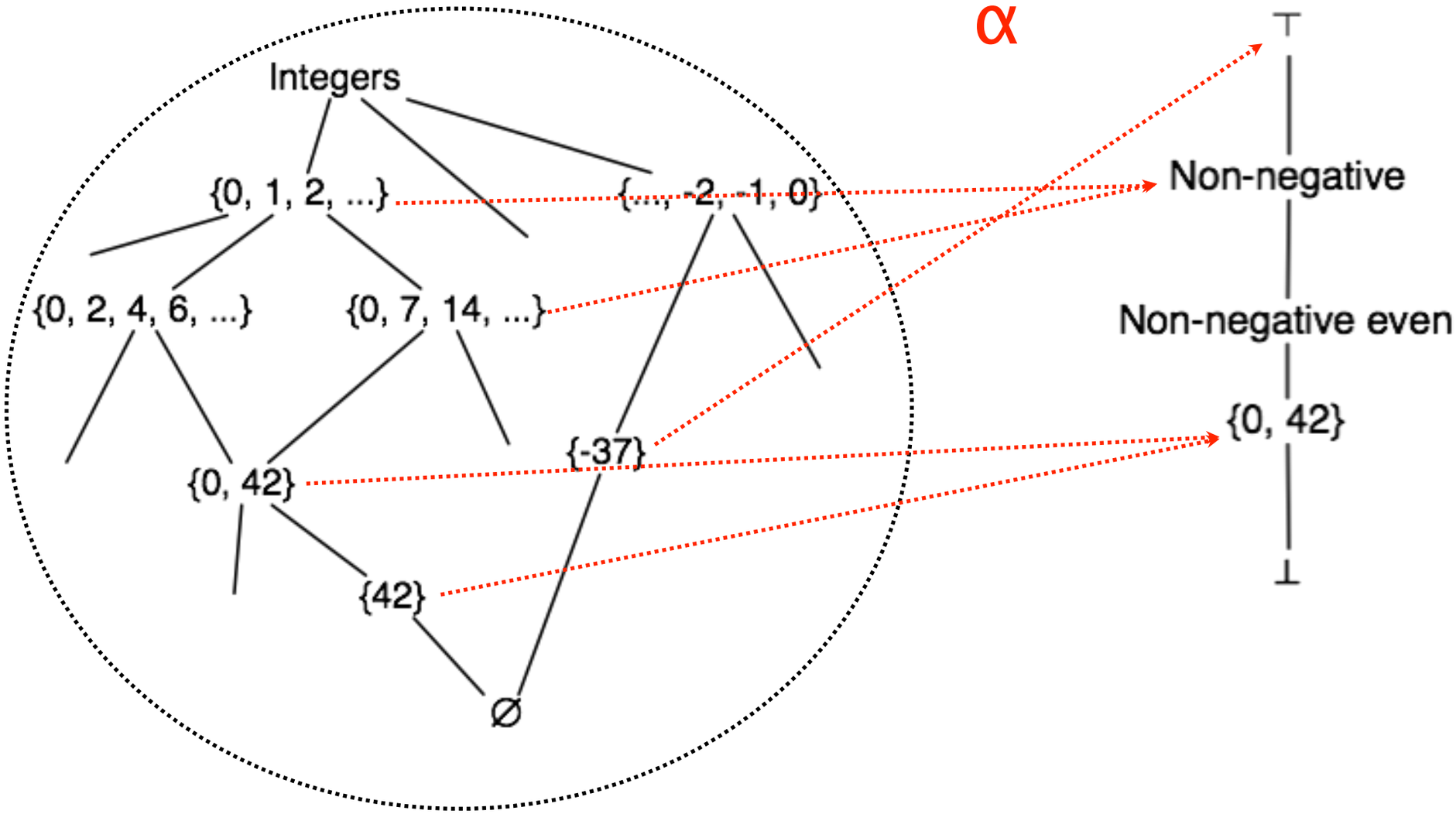
Example Abstraction



Concrete values: sets of integers Abstract values

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

Abstraction is Imprecise

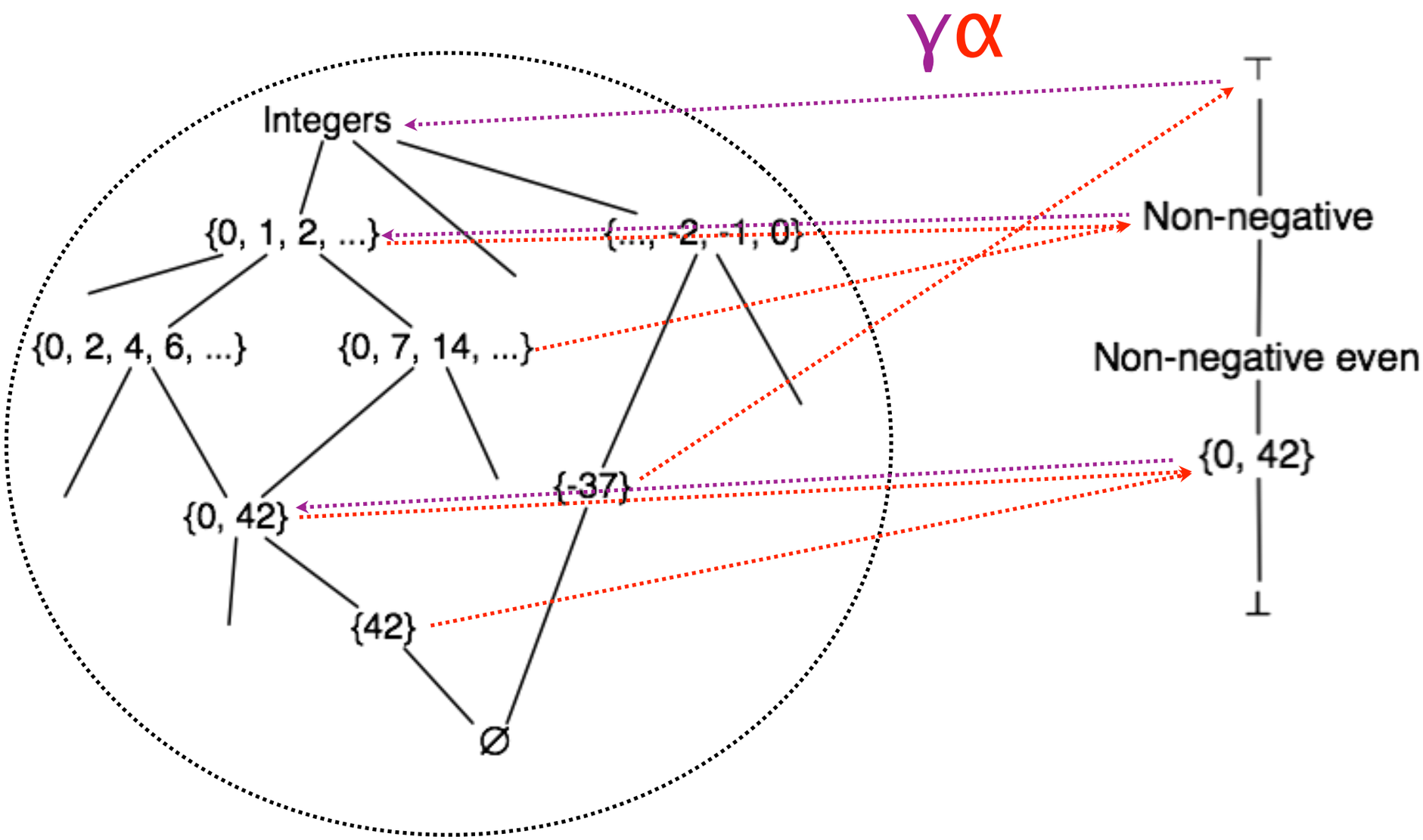


Concrete values: sets of integers

Abstract values

Abstraction function α maps each concrete set to the best (*least imprecise*) abstract value

Composing α and γ



Concrete values: sets of integers Abstract values

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

  (Sometimes $\alpha(\gamma(A)) \sqsubseteq A$ --- a *Galois Connection*)
- Also say $\forall x \in S, y \in A.\ \alpha(x) \sqsubseteq y \Leftrightarrow x \sqsubseteq \gamma(y)$
  - Exercise: Prove that this requirement is equivalent to the above two requirements

Concrete Language

- Concrete domain:
  - Sets of Integers : 2^Z
- Expressions: integers and multiplication
  - e ::= i | e * e | e + e | -e
- Standard semantics of the program
  - Eval : e → Z
  - Eval(i) = i
  - Eval(e1*e2) = Eval(e1) × Eval(e2)
  - …
- Exercise: write as big-step operational semantics

Abstract Language

- Abstract domain: 0 and signs and "don't know"
  - a ::= 0 | + | - | T
- Programs: abstract values and multiplication
  - ae ::= a | ae*ae | ae + ae | -ae

- Semantics of the program
  - Define Acomp : ae → a
  - Let Aeval : e → a be Acomp • α
    - We'll define AEval directly next

Semantics of abstract expressions

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

  - AEval : e → {-, 0, +, T}

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

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

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

  - $\text{AEval}(-e1) = \underline{-}\,\text{AEval}(e1)$

Semantics of abstract operations

×	+	0	-	T
+	+	0	-	T
0	0	0	0	T
-	-	0	+	T
T	T	T	T	T

±	+	0	-	T
+	+	+	T	T
0	+	0	-	T
-	T	-	-	T
T	T	T	T	T

=	+	0	-	T
	-	0	+	T

Two Ways to Lose Information

- OK: Abstraction still precise enough
  - Eval((5 * 5) + 6) = 31
  - AEval((5*5) + 6) = (+ × +) ± + = +
    - 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) = (+ ± +) ± - = + ± - = ⊤
    - We 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

$\gamma(\perp) = \varnothing$

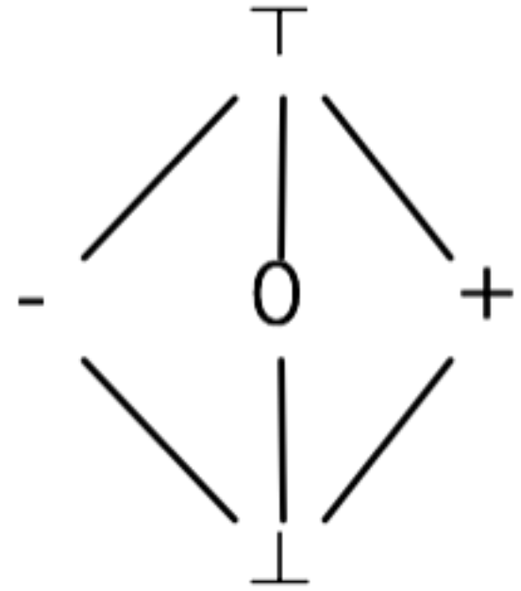
$\div$	+	0	-	$\top$	$\perp$
+	+	0	-	$\top$	$\perp$
0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
-	-	0	+	$\top$	$\perp$
$\top$	$\top$	0	$\top$	$\top$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Find the bug: the table is not correct.

Hint: what should be the result of 7 divided by 5?

The Abstract Domain

- Look, Ma, a lattice!
- We've got:
  - A set of elements {⊥, +, 0, -, ⊤}
  - A relation ⊑ that is
    - Reflexive
    - Anti-symmetric
    - Transitive
  - And
    - The least upper bound (lub, ⊔) and greatest lower bound (glb, ⊓) exists for any pair of elements
    - So it's a lattice



Abstraction and Concretization

- Concretization function γ

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

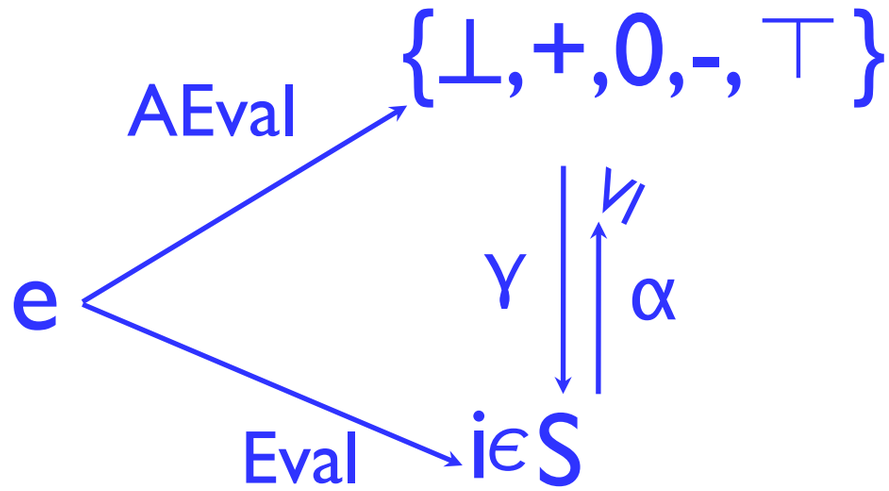
- Abstraction function maps concrete values (sets of integers) to the *smallest* valid abstract element
  - $\alpha(S) = \begin{pmatrix} - & \exists i \in S \,.\, i<0 \\ \bot & \text{otherwise} \end{pmatrix} \sqcup \begin{pmatrix} 0 & \exists i \in S \,.\, i=0 \\ \bot & \text{otherwise} \end{pmatrix}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

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

Proving soundness

- To prove soundness, we rely on the facts that
  - α and γ form a Galois insertion
  - And abstract operations $\underline{op}$ are locally correct
    - $\gamma(\underline{op}(a1, ..., an)) \sqsupseteq op(\gamma(a1), ..., \gamma(an))$
    - Note: We've extended op pointwise to sets
      - I.e., if S and T are sets, $S+T = \{s+t \mid s \in S, t \in T\}$

Proof: Show Eval(e) ∈ 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
    - ∈ γ(AEval(e1)) op γ(AEval(e2)) by induction
    - ⊆ γ(AEval(e1) op AEval(e2)) by local correctness of op
    - = γ(AEval(e1 op e2)) by definition of AEval

Static analysis

- Static analysis aims to reason about *all* of a program's executions
  - So far we have implicitly considered just a single one
- Approach:
  - Define an operational semantics that defines all program executions; called the *collecting semantics*
  - Define an abstract interpretation for this semantics
    - By the soundness of abstract interpretation, we are sure that our conclusions apply to all possible program executions

Collecting semantics

- Lift semantics judgments to a set of stores
  - $\langle a, S \rangle \rightarrow N$
    - In state $\sigma \in S$, arithmetic expression a evaluates to $n \in N$
  - $\langle b, S \rangle \rightarrow 2^{bv}$
    - In state $\sigma \in S$, boolean expression b evaluates to $bv \in \{true, false\}$
  - $\langle c, S \rangle \rightarrow S'$
    - In state $\sigma \in S$, command c executes producing some state $\sigma' \in S'$
- Most rules are straightforward liftings

$$\frac{}{\langle n, S \rangle \rightarrow \{n\}} \qquad \frac{}{\langle X, S \rangle \rightarrow \{ n \mid \sigma \in S \wedge n = \sigma(X) \}}$$

More (straightforward) rules

$$\frac{}{\langle \text{skip}, S\rangle \rightarrow S}$$

$$\frac{\langle c0, S\rangle \rightarrow S0 \qquad \langle c1, S0\rangle \rightarrow S1}{\langle c0; c1, S\rangle \rightarrow S1}$$

$$\frac{\langle a, S\rangle \rightarrow N \qquad S' = \{\, \sigma' \mid (n \in N) \wedge (\sigma \in S) \wedge \sigma' = \sigma[X \mapsto n] \,\}}{\langle X{:=}a, S\rangle \rightarrow S'}$$

Conditionals

$$T = \{ \sigma \mid \sigma \in S \wedge \langle b, \{\sigma\} \rangle \rightarrow \{true\} \}$$

$$F = \{ \sigma \mid \sigma \in S \wedge \langle b, \{\sigma\} \rangle \rightarrow \{false\} \}$$

$$\frac{\langle c0, T \rangle \rightarrow S1 \quad \langle c1, F \rangle \rightarrow S2}{\langle \text{if } b \text{ then } c0 \text{ else } c1, S \rangle \rightarrow S1 \cup S2}$$

Loops

$$T = \{\ \sigma \mid \sigma \in S \wedge \langle b, \{\sigma\}\rangle \rightarrow \{true\}\ \}$$

$$F = \{\ \sigma \mid \sigma \in S \wedge \langle b, \{\sigma\}\rangle \rightarrow \{false\}\ \}$$

$$\frac{\langle c, T\rangle \rightarrow S1 \quad S1 \cup S = S}{\langle \text{while } b \text{ do } c, S\rangle \rightarrow F}$$

Found a fixed point

$$T = \{\ \sigma \mid \sigma \in S \wedge \langle b, \{\sigma\}\rangle \rightarrow \{true\}\ \}$$

$$\frac{\langle c, T\rangle \rightarrow S1 \quad S1 \cup S \neq S \qquad \langle \text{while } b \text{ do } c, S1 \cup S\rangle \rightarrow S2}{\langle \text{while } b \text{ do } c, S\rangle \rightarrow S2}$$

Work out an example

- Example program c is

  while (x < 100) { x := x + 2 }

- Suppose we compute $\langle c, S \rangle \rightarrow S'$ with $S = \{\sigma\}$
  - If σ is $[x \mapsto 0]$ then what is S' ?
  - What is the fixed point of S at the beginning of the loop?

Soundness of Collecting Semantics

- Theorem: For all S, c, σ ∈ S, and σ' ∈ Store
  - $\langle c, \sigma \rangle \rightarrow \sigma'$ *iff* $\langle c, S \rangle \rightarrow S'$ and $\sigma' \in S'$
- Thus, collecting semantics directly computes the result of all possible executions of c in stores S
  - But it's uncomputable!
- Goal: perform an abstract interpretation of the collecting semantics
  - Computable, and thus, by soundness, approximates the result of all runs

Abstract domains

- Abstract values, and stores
  - B ::= true | false | T | ⊥
  - N ::= + | 0 | - | T | ⊥
  - S: Var → A
- B and N and S are all lattices
  - Proof as an exercise
- Note that S treats each variable independently
  - Cannot characterize stores in which the values of variables are always correlated

Command execution

$$\frac{}{\langle \text{skip}, \underline{S} \rangle \rightarrow \underline{S}}$$

$$\frac{\langle a, \underline{S} \rangle \rightarrow \underline{N}}{\langle X{:=}a, \underline{S} \rangle \rightarrow \underline{S}[X \mapsto \underline{N}]}$$

$$\frac{\langle c0, \underline{S} \rangle \rightarrow \underline{S0} \quad \langle c1, \underline{S0} \rangle \rightarrow \underline{S1}}{\langle c0; c1, \underline{S} \rangle \rightarrow \underline{S1}}$$

All states such that b holds

$$\frac{\langle c0, \underline{S|b} \rangle \rightarrow \underline{S0} \quad \langle c1, \underline{S|\neg b} \rangle \rightarrow \underline{S1}}{\langle \text{if b then c0 else c1}, \underline{S} \rangle \rightarrow \underline{S0} \sqcup \underline{S1}}$$

Loops

$$\frac{\langle c, \underline{S}|b\rangle \rightarrow \underline{S1} \quad \underline{S1} \sqcup \underline{S} = \underline{S} \qquad \underline{F} = \underline{S}|\neg b}{\langle \text{while } b \text{ do } c, \underline{S}\rangle \rightarrow \underline{F}}$$

$$\frac{\langle c, \underline{S}|b\rangle \rightarrow \underline{S1} \quad \underline{S1} \sqcup \underline{S} \neq \underline{S} \qquad \langle \text{while } b \text{ do } c, \underline{S1} \sqcup \underline{S}\rangle \rightarrow \underline{S2}}{\langle \text{while } b \text{ do } c, \underline{S}\rangle \rightarrow \underline{S2}}$$

Soundness

- Soundness now refers to the collecting semantics, rather than the standard semantics
  - If $\underline{S} = \alpha(S)$ then $\langle c, S \rangle \rightarrow S2$ implies $\langle c, \underline{S} \rangle \rightarrow \underline{S2}$

    where $\alpha(S2) \sqsubseteq \underline{S2}$

    - Alternatively, that $S2 \subseteq \gamma(\underline{S2})$

The Intervals Domain

- Abstract domain of integer ranges (for variable x)

$$A ::= \{[l,u] \mid l \in Z \cup -\infty, u \in Z \cup +\infty, l \leq u\}$$

$$[l_1, u_1] \sqsubseteq [l_2, u_2] \Leftrightarrow l_2 \leq l_1 \wedge u_1 \leq u_2$$

$$[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$$

- Abstraction function -- $\alpha : D \longrightarrow A$

$$\alpha(X) = [\min(\{v \mid x \mapsto v \in X\}),$$

$$\max(\{v \mid x \mapsto v \in X\})]$$

- Concretization function -- $\gamma : A \longrightarrow D$

$$\gamma([l,u]) = \{x \mapsto i \mid l \leq i \leq u\}$$

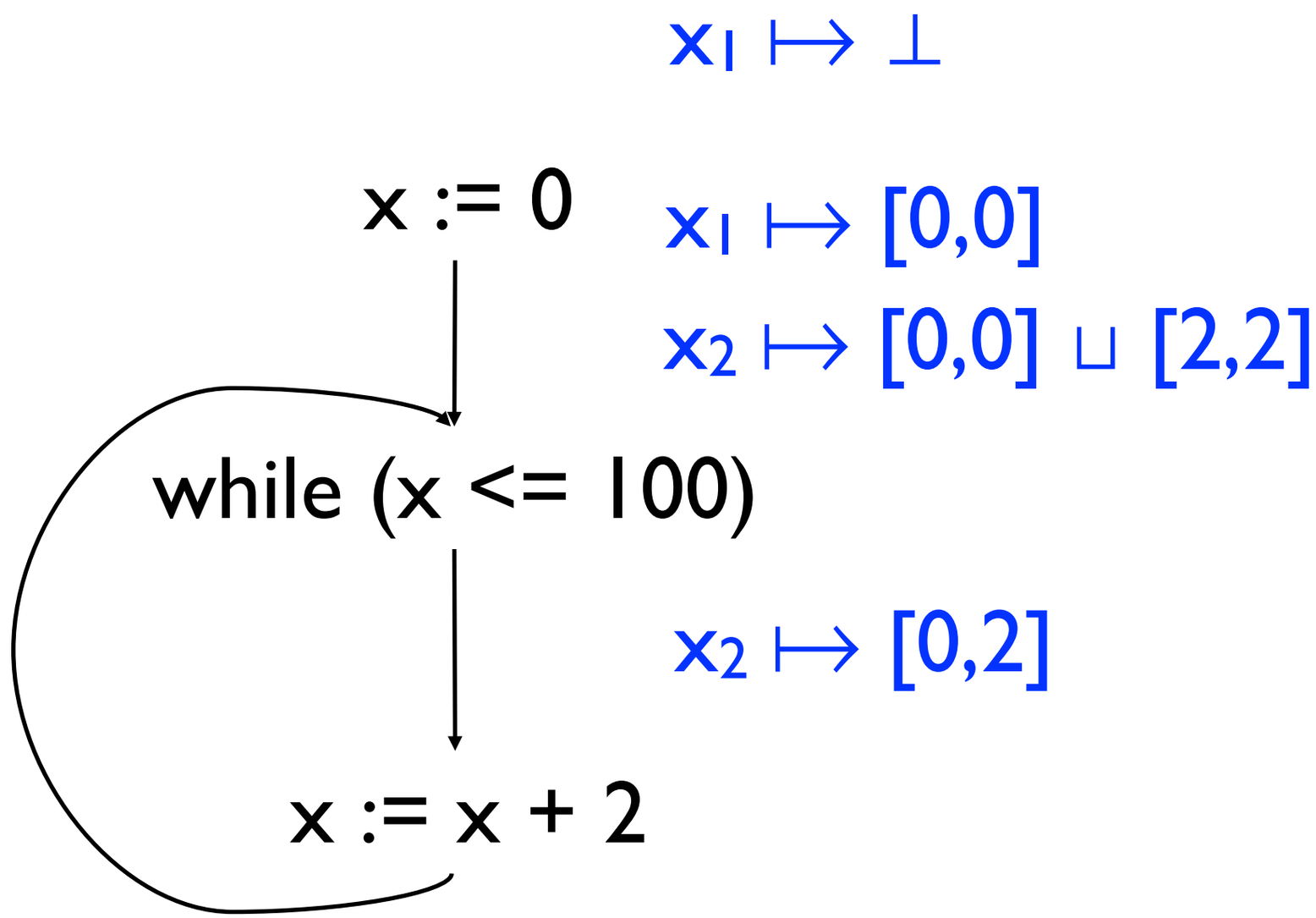
Galois Insertion?

- Recall:
  - $x \sqsubseteq \gamma(\alpha(x))$
  - $y = \alpha(\gamma(y))$
    - Examples:
      - $x = \{-2, 8, -5\}$
        - $\alpha\{x\} = [-5, 8]$ and $\gamma(\alpha(x)) = \{-5, -4, \ldots, 8\}$
      - $y = [-8, 8]$
        - $\gamma\{y\} = \{-8, -7, \ldots, 7, 8\}$ and $\alpha(\gamma(y)) = [-8,8]$

Abstract semantics

Left as an exercise ...

Abstract Interpretation

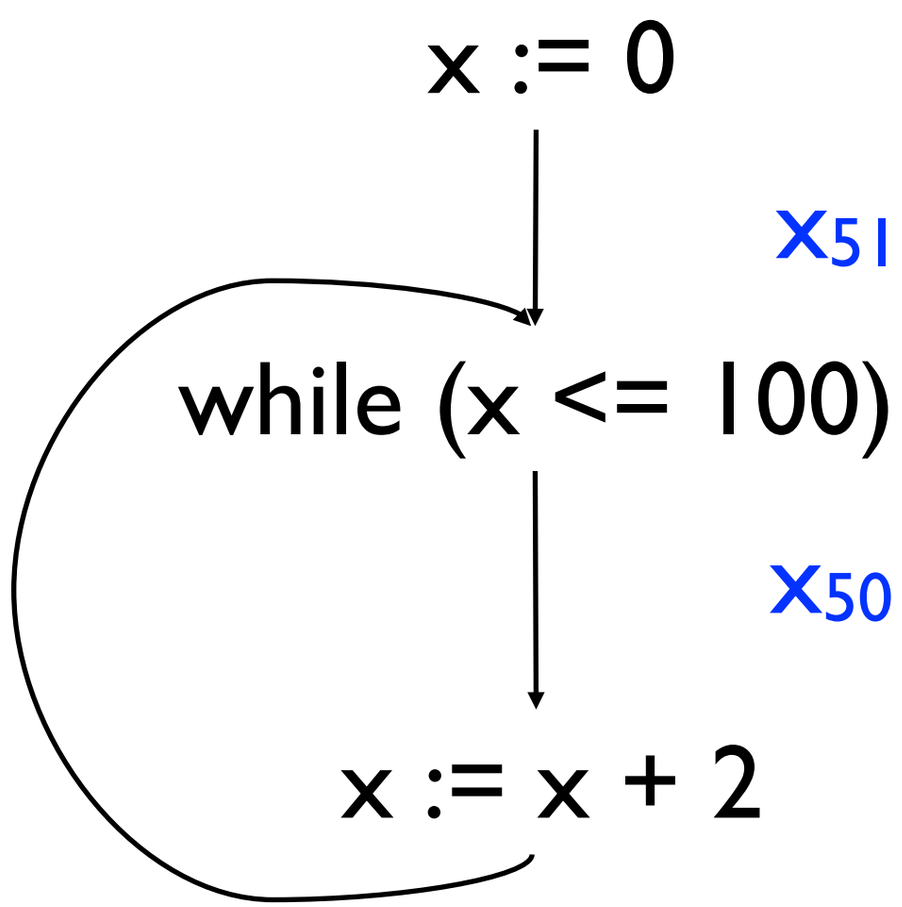


$x_1 \mapsto [2,2]$

$x_2 \mapsto [2,4]$

Abstract Interpretation

$x_1 \mapsto \bot$



$x_{51} \mapsto [0,100] \sqcup [2,102]$

$x_{50} \mapsto [0,100]$

$x_{50} \mapsto [2,102]$

$x_{final} \mapsto [101,102]$

Precision

- Abstract interpretation for loop entry
  - $(x \mapsto [0, 102] \in A)$
  - $\gamma([0, 102]) = \{0, 1, 2, .., 102\}$

- But collecting semantics gives
  - $\{0, 2, 4, \ldots 102\}$

Convergence

- How do we know that we will reach a fixed point?
  - We could pick A to be a finite lattice
  - Or, A could be an infinite lattice with no infinite ascending chain
- But intervals satisfy neither of these conditions
- What about speed of convergence?
  - Example took 50 iterations to converge
  - Can we do better?

Widening and Narrowing

- Widening guarantees convergence even for infinite lattices
  - But loses precision
  - Also usually improves rate of convergence even for finite lattices

- Narrowing recovers precision lost by widening

Widening : ∇

- Given a lattice L, a widening $\nabla : L \times L \longrightarrow L$ must satisfy
  - $\forall x, y \in L.\ x \sqsubseteq x \nabla y$
  - $\forall x, y \in L.\ y \sqsubseteq x \nabla y$

  For all chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$,

  $$y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$$

  Is not strictly increasing
- Similar role to a lub

Example Widening for Intervals

$\perp \nabla X = X$

$X \nabla \perp = X$

$[l_1, u_1] \nabla [l_2, u_2] =$

$[\text{if } l_2 < l_1 \text{ then } -\infty \text{ else } l_1, \text{if } u_2 > u_1 \text{ then } +\infty \text{ else } u_1]$

Given a sequence of loop iterates (joins per iteration)

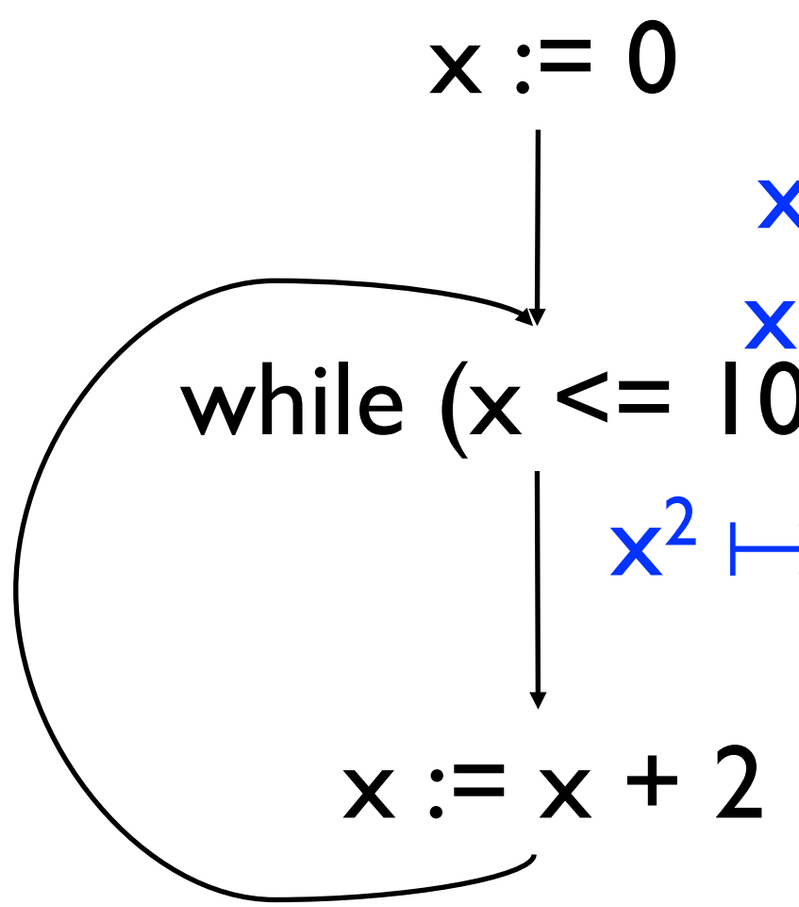
$x^0, x^1, \ldots, x^i, \ldots$

Use widening instead to compute

$y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}$

Widening Example

$x \mapsto \bot$



$x^1 \mapsto \bot \nabla [0,0] = [0,0]$

$x^2 \mapsto x^1 \nabla [2,2] = [0,+\infty]$

$x^2 \mapsto x^1 \nabla [0,+\infty] \sqcap [-\infty, 100]$

$= [0,100]$

$x^1 \mapsto [2,2]$

$x^2 \mapsto [2,102]$

Narrowing : Δ

- Recover lost precision due to widening
  - When you overshoot the real fixed point, narrowing can bring you back
- Skipping this topic for now ...

Other abstract domains

- We have seen signs, and intervals
  - Latter also known as "boxes"
- Relational domains involve multiple variables
  - Convex polyhedra: $a_1x_1 + a_2x_2 + ... + a_kx_k \geq c_i$
    - Special case, octagons: $ax + by \geq c \qquad a,b \in \{-1,0,1\}$
- Many more domains for other PL features
  - Arrays, pointers, etc.
- Cool paper: abstracting abstract machines
  - van Horn and Might, ICFP'10

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

- But can also be used to model many other analysis
  - CFA, type inference etc.

Conclusions

- Galois connections with finite lattices or Widening/ Narrowing?
  - Typically some combination of the two

- Theory is completely general
  - What are good choices for modeling data structures and the heap? Higher-order functions? Objects?

- Picking the right abstract domains; finding the right widening/narrowing can be tricky

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)

- But there are successful tools:
  - ASTREE has proved the absence of runtime errors in the primary control software of the Airbus A340
    - Polyspace, Inc. has several high-profile customers
  - PolySpace C and Ada verifiers