CMSC 631 – Program Analysis and Understanding
Fall 2017

Abstract Interpretation

Based on lectures by David Schmidt, Alex Aiken, Tom Ball, and Cousot & Cousot

What is an Abstraction?

- A property from some domain

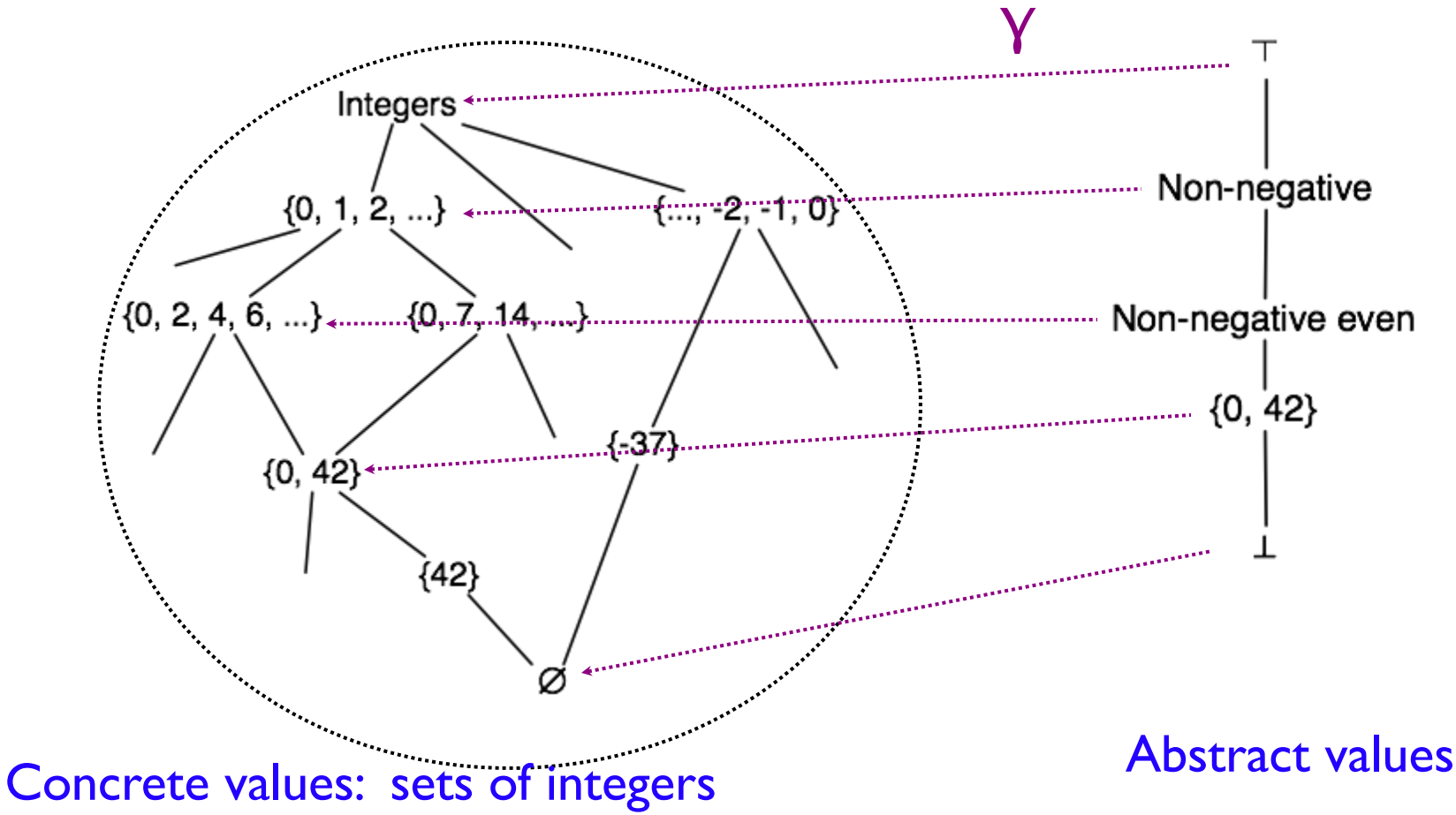
Blue (color)

Planet (classification)

$\sqcup\!\sqcup$

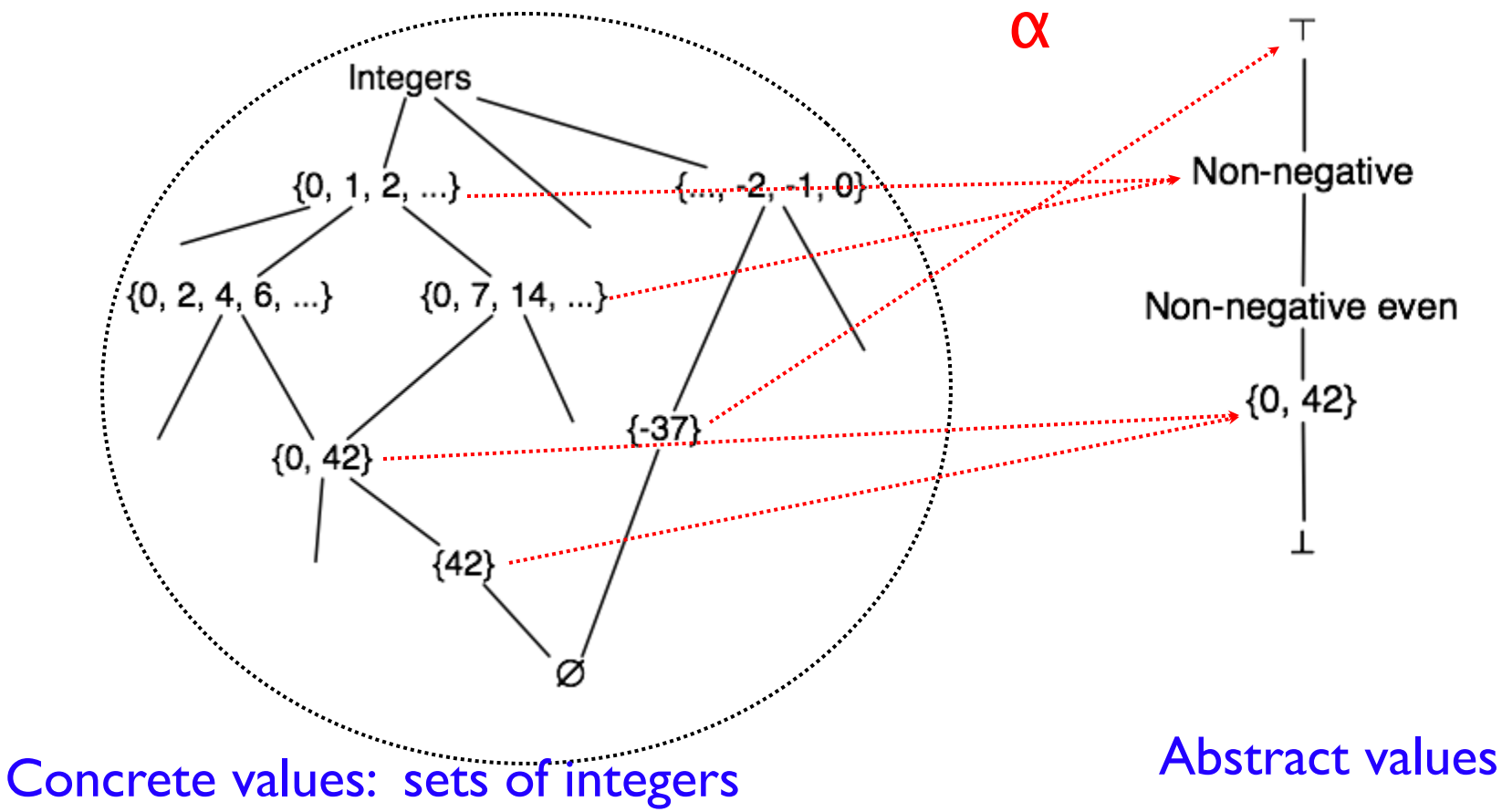
6000..7000km (radius)

Example Abstraction



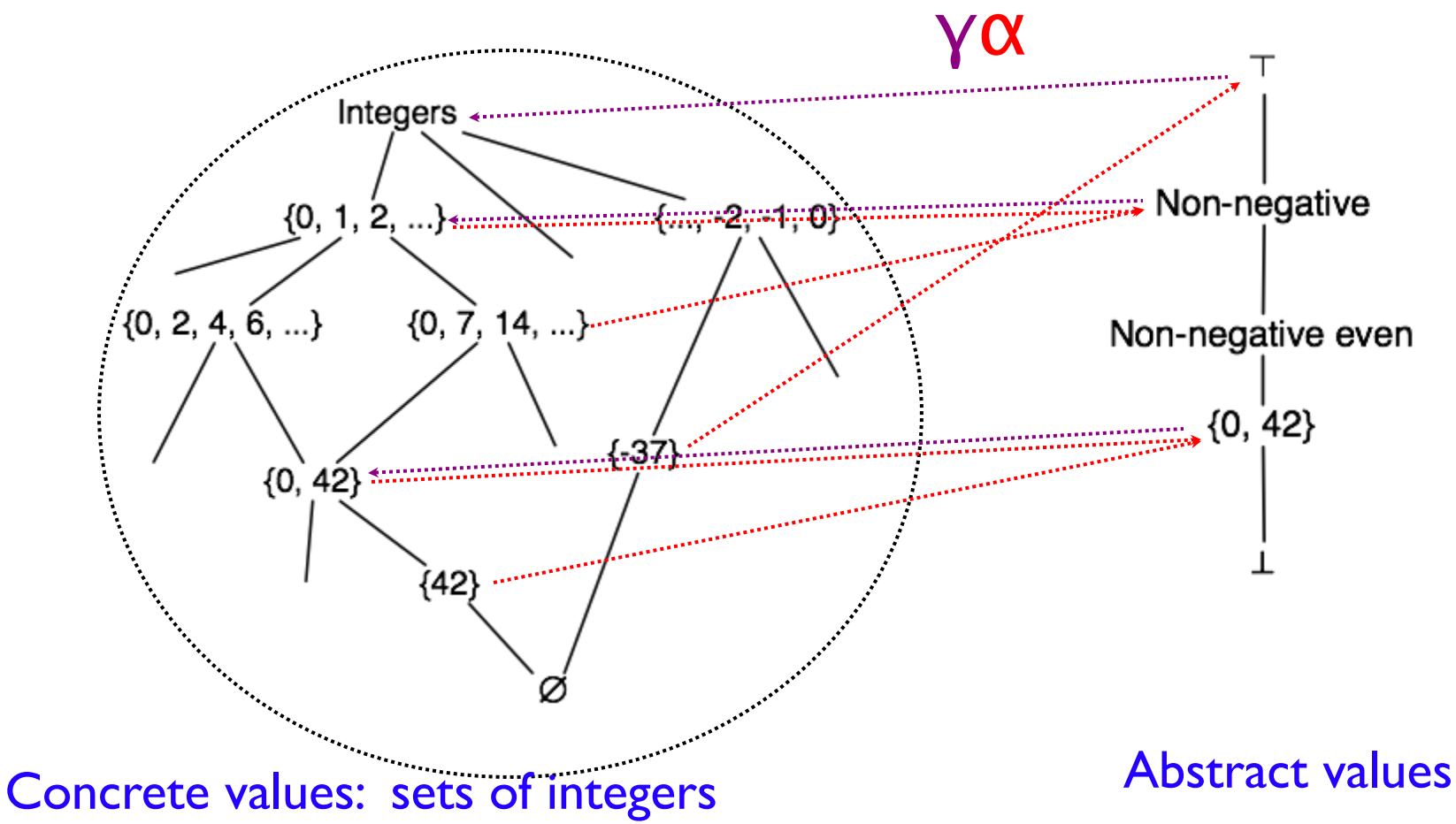
Concretization function γ maps each abstract value to concrete values it represents

Abstraction is Imprecise



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

Plan

- A simple example
  - Approximating the sign of an arithmetic expression
- A more realistic example
  - Approximating sets of integers by ranges in a while language
- Convergence and precision
  - Widening and narrowing

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)
  - …

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 : e → ae and Abseval : ae → a
  - Let AEval : e → a be Abseval • Acomp
    - Abstract concrete constants, then evaluate abstractly
    - But we define AEval directly next

Semantics of abstract expressions

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

  - AEval : e → {-, 0, +, T}
  - AEval(i) = $\begin{cases} + & i > 0 \\ 0 & i = 0 \\ - & i < 0 \end{cases}$
  - AEval(e1*e2) = AEval(e1) $\underline{\times}$ AEval(e2)
  - AEval(e1+e2) = AEval(e1) $\underline{+}$ AEval(e2)
  - AEval(-e1) = $\underline{-}$ AEval(e1)

Semantics of abstract operations

×	+	0	-	T
+	+	0	-	T
0	0	0	0	0
-	-	0	+	T
T	T	0	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) = (+ $\underline{\times}$ +) $\underline{+}$ + = +
    - 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 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?
  - If we divide any integer in a set by 0, the result is the empty set, since x ÷ 0 is undefined

$\gamma(\bot) = \varnothing$

÷	+	0	-	⊤	⊥
+	?	0	?	⊤	⊥
0	⊥	⊥	⊥	⊥	⊥
-	?	0	?	⊤	⊥
⊤	⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	⊥	⊥	⊥

What should the ? be?

- *Hint*: what is the result of 5 divided by 7?

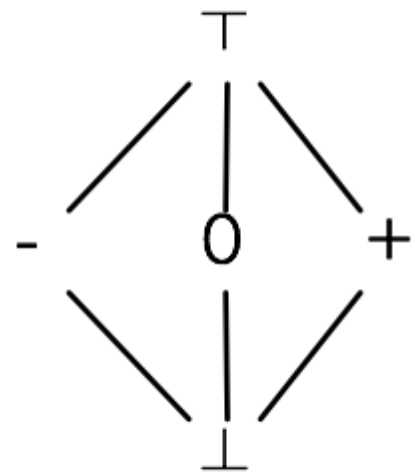
Why is the second-to-last row all ⊥ ?

- *Hint*: what numbers are included in ⊤ ?

Could gain precision by extending a with *"0 or +"* and *"0 or –"*

The Abstract Domain

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

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

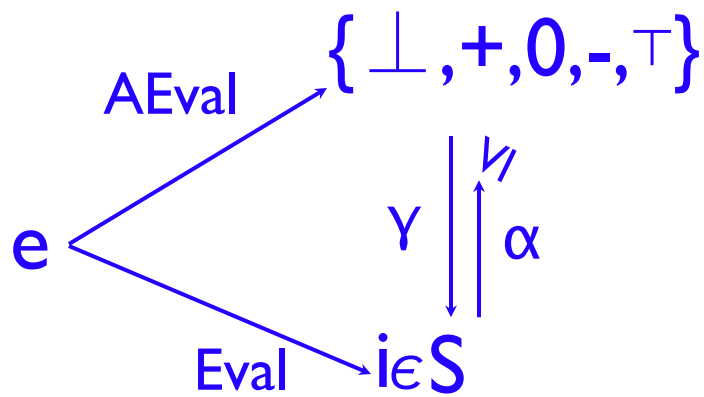
- Abstraction function maps concrete values (sets of integers) to the *smallest* valid abstract element
  - $\alpha(S) = \begin{pmatrix} - & \exists i \in S \,.\, i<0 \\ \bot & \text{otherwise} \end{pmatrix} \sqcup \begin{pmatrix} 0 & \exists i \in S \,.\, i=0 \\ \bot & \text{otherwise} \end{pmatrix}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

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

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

A Simple Imperative Language

- For arithmetic language
  - Number of operations in Aeval was the same as eval
  - No loops, so convergence is trivial
- Slightly more realistic
  - c ::= skip | c; c | x := e | if0 e then c else c | while0 e do c
  - e ::= … | e < e | … etc.
- Standard concrete (big step) semantics
- Goal: approximate the *collecting semantics* of c

Concrete Semantics

- Semantics over states and numbers
  - $\langle e, \sigma \rangle \rightarrow n$
  - $\langle c, \sigma \rangle \rightarrow \sigma$
- Standard rules

$$\frac{}{\langle x, \sigma \rangle \rightarrow \sigma(x)} \qquad \frac{}{\langle n, \sigma \rangle \rightarrow n}$$

$$\frac{\langle e, \sigma \rangle \rightarrow n}{\langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto n]} \qquad \frac{}{\langle skip, \sigma \rangle \rightarrow \sigma}$$

$$\frac{\langle c0, \sigma \rangle \rightarrow \sigma 0 \quad \langle c1, \sigma 0 \rangle \rightarrow \sigma 1}{\langle c0; c1, \sigma \rangle \rightarrow \sigma 1}$$

Collecting Semantics

- Resembles collecting semantics
  - $\langle e, S \rangle \rightarrow N$
  - $\langle c, S \rangle \rightarrow S'$
  - Where S is a set of states, and N is a set of numbers
- Many 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

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

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

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

Conditionals

$$\frac{\begin{array}{c} T = \{ \sigma \mid \sigma \in S \wedge \langle e, \{\sigma\}\rangle \rightarrow \{0\} \} \\ F = \{ \sigma \mid \sigma \in S \wedge \langle e, \{\sigma\}\rangle \rightarrow \{n\} \wedge n \neq 0\} \\ \langle c0, T\rangle \rightarrow S1 \quad \langle c1, F\rangle \rightarrow S2 \end{array}}{\langle \text{if0 } e \text{ then } c0 \text{ else } c1, S\rangle \rightarrow S1 \cup S2}$$

Loops

$$\frac{T = \{ \sigma \mid \sigma \in S \wedge \langle e, \{\sigma\}\rangle \rightarrow \{0\} \} \quad \langle c, T\rangle \rightarrow S1 \quad S1 \cup S \neq S \quad \langle \text{while0 } e \text{ do } c, S1 \cup S\rangle \rightarrow S2}{\langle \text{while0 } e \text{ do } c, S\rangle \rightarrow S2}$$

$$\frac{T = \{ \sigma \mid \sigma \in S \wedge \langle e, \{\sigma\}\rangle \rightarrow \{0\} \} \quad \langle c, T\rangle \rightarrow S1 \quad S1 \cup S = S \quad F = \{ \sigma \mid \sigma \in S \wedge \langle e, \{\sigma\}\rangle \rightarrow \{n\} \wedge n \neq 0\}}{\langle \text{while0 } e \text{ do } c, S\rangle \rightarrow F}$$

Found a fixed point

Work out an example

- Example program c is

  while (x < 4) { 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, $\sigma \in S$, and σ'
  - $\langle c, \sigma \rangle \rightarrow \sigma'$ *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
  - $\underline{N}$ ::= + | 0 | - | T | ⊥
  - $\underline{S}$: Var → $\underline{N}$
- $\underline{N}$ and $\underline{S}$ are lattices
  - Proof as an exercise
- Note that $\underline{S}$ treats each variable independently
  - Cannot characterize stores in which the values of variables are always correlated

Command execution

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

$$\frac{\langle \text{e}, \underline{S}\rangle \rightarrow \underline{N}}{\langle \text{x:=e}, \underline{S}\rangle \rightarrow \underline{S}[\text{x}\mapsto\underline{N}]}$$

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

All states such that e is zero

$$\frac{\langle \text{c0}, \underline{S}|\underline{e}=0\rangle \rightarrow \underline{S0} \qquad \langle \text{c1}, \underline{S}|\underline{e}\neq 0\rangle \rightarrow \underline{S1}}{\langle \text{if0 e then c0 else c1}, \underline{S}\rangle \rightarrow \underline{S0} \sqcup \underline{S1}}$$

Loops

$$\frac{\langle c, \underline{S}|\underline{e}=0\rangle \rightarrow \underline{S1} \quad \underline{S1} \sqcup \underline{S} \neq \underline{S} \qquad \langle \text{while0 } e \text{ do } c, \underline{S1} \sqcup \underline{S}\rangle \rightarrow \underline{S2}}{\langle \text{while0 } e \text{ do } c, \underline{S}\rangle \rightarrow \underline{S2}}$$

$$\frac{\langle c, \underline{S}|e=0\rangle \rightarrow \underline{S1} \quad \underline{S1} \sqcup \underline{S} = \underline{S} \qquad \underline{F} = \underline{S}|\underline{e}\neq 0}{\langle \text{while0 } e \text{ do } c, \underline{S}\rangle \rightarrow \underline{F}}$$

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 single variable)
  - $A ::= \{[l,u] \mid l \in Z \cup -\infty \wedge u \in Z \cup +\infty \wedge l \leq u\}$
  - $[l_1, u_1] \sqsubseteq [l_2, u_2] \iff l_2 \leq l_1 \wedge u_1 \leq u_2$
  - $[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$

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

$$\alpha(S) = [\ \min(\{v \mid v \in S\}),\ \max(\{v \mid v \in S\})]$$

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

$$\gamma([l,u]) = \{n \mid l \leq n \leq u\}$$

Galois Insertion?

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

Abstract Interpretation

$x \mapsto \bot$

x := 0

$x \mapsto [0,0] \sqcup [2,2]$

while (x <= 100)

$x \mapsto [0,2]$

x := x + 2

$x \mapsto [2,2] \sqcup [2,4]$

Abstract Interpretation

$x \mapsto \bot$

x := 0

$x \mapsto [0,100] \sqcup [102,102]$

while (x <= 100)

$x \mapsto [0,100]$

x := x + 2

$x \mapsto [2,100] \sqcup [102,102]$

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 our choice of A satisfies 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 $\nabla : L \times L \longrightarrow L$ requires
  - $\forall x, y \in L.\ x \sqsubseteq x \nabla y$
  - $\forall x, y \in L.\ y \sqsubseteq x \nabla y$

  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 to role of lub $\sqcup$

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 iterates for a loop

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

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 := 0

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

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

while (x <= 100)

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

x := x + 2

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

$x^2 \mapsto [2,2] \nabla [2,102] = [2, +\infty]$

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 C and Ada verifiers
- Our own tool: http://www.cs.umd.edu/~mwh/papers/wei17evaluating.html