

## Abstract Interpretation Non-Standard Semantics

Lecture 10-11  
CS263

CS 263

1

### The Plan

- We will introduce abstract interpretation by example.
- Starting with a miniscule language we will build up to a fairly realistic application.
- Along the way we will see most of the ideas and difficulties that arise in a big class of applications.

CS 263

3

### An Abstraction

- Assume that we are interested not in the value of the expression, but only in its sign:
  - positive (+), negative (-), or zero (0).
- We can define an abstract semantics that computes only the sign of the result
  - $\sigma: \text{Exp} \rightarrow \{-, 0, +\}$

$$\sigma(n) = \text{sign}(n)$$

$$\sigma(e_1 * e_2) = \sigma(e_1) \otimes \sigma(e_2)$$

$\otimes$	-	0	+
-	+	0	-
0	0	0	0
+	-	0	+

CS 263

5

### The Problem

- It is *extremely* useful to predict program behavior *statically* (without running the program.)
  - For optimizing compilers.
  - For software engineering tools.
- The semantics we studied so far give us the precise semantics
- However, precise static predictions are impossible.
  - The exact semantics is not computable.
- We must settle for approximate, but correct static analysis. (e.g. VC vs. WP)

CS 263

2

### A Tiny Language

- Consider the following language of arithmetic
  - $e ::= n \mid e_1 * e_2$
- The denotational semantics of this language
  - $\llbracket n \rrbracket = n$
  - $\llbracket e_1 * e_2 \rrbracket = \llbracket e_1 \rrbracket \times \llbracket e_2 \rrbracket$
- For this language the precise semantics is computable

CS 263

4

### Correctness of Sign Abstraction

- We can show that the abstraction is correct in the sense that it correctly predicts the sign
  - $\llbracket e \rrbracket > 0 \Leftrightarrow \sigma(e) = +$
  - $\llbracket e \rrbracket = 0 \Leftrightarrow \sigma(e) = 0$
  - $\llbracket e \rrbracket < 0 \Leftrightarrow \sigma(e) = -$
- Our semantics is abstract but precise
- Proof is by structural induction on expression  $e$ 
  - Each case repeats similar reasoning

CS 263

6

### Another View of Soundness

- We associate with each concrete value an abstract value:

$$\beta : \mathbb{Z} \rightarrow \{-, 0, +\}$$

- This is called the abstraction function
- Conversely we can also define the concretization function:

$$\begin{aligned} \gamma : \{-, 0, +\} &\rightarrow \mathcal{P}(\mathbb{Z}) \\ \gamma(+) &= \{n \in \mathbb{Z} \mid n > 0\} \\ \gamma(0) &= \{0\} \\ \gamma(-) &= \{n \in \mathbb{Z} \mid n < 0\} \end{aligned}$$

CS 263

7

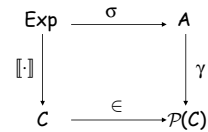
### Another View of Soundness (Cont.)

- Soundness can be stated succinctly

$$\forall e \in \text{Exp. } \llbracket e \rrbracket \in \gamma(\sigma(e))$$

(the true value of the expression is among the concrete values represented by the abstract value of the expression)

- Let  $C$  be the concrete domain (e.g.  $\mathbb{Z}$ ) and  $A$  be the abstract domain (e.g.  $\{-, 0, +\}$ )



CS 263

8

### Another View of Soundness (Cont.)

- Consider the generic abstraction of an operator  $\sigma(e_1 \text{ op } e_2) = \sigma(e_1) \text{ op } \sigma(e_2)$
- This is sound iff  $\forall a_1 \forall a_2. \gamma(a_1 \text{ op } a_2) \supseteq \{n_1 \text{ op } n_2 \mid n_1 \in \gamma(a_1), n_2 \in \gamma(a_2)\}$
- E.g.  $\gamma(a_1 \otimes a_2) \supseteq \{n_1 * n_2 \mid n_1 \in \gamma(a_1), n_2 \in \gamma(a_2)\}$
- This reduces the proof of correctness to one proof for each operator

CS 263

9

### Abstract Interpretation

- This is our first example of an abstract interpretation.
- We carry out computation in an abstract domain
- The abstract semantics is a sound approximation of the standard semantics.
- The concretization and abstraction functions establish the connection between the two domains.

CS 263

10

### Adding Unary Minus and Addition

- We extend the language to  $e ::= n \mid e_1 * e_2 \mid - e$
- We define  $\sigma(- e) = \ominus \sigma(e)$

	-	0	+
$\ominus$	+	0	-

- Now we add addition:  $e ::= n \mid e_1 * e_2 \mid - e \mid e_1 + e_2$
- We define  $\sigma(e_1 + e_2) = \sigma(e_1) \oplus \sigma(e_2)$

$\oplus$	-	0	+
-	-	-	?
0	-	0	+
+	?	+	+

CS 263

11

### Adding Addition

- The sign values are not closed under addition
- What should be the value of " $+ \oplus -$ "?
- Start from the soundness condition:  $\gamma(+ \oplus -) \supseteq \{n_1 + n_2 \mid n_1 > 0, n_2 < 0\} = \mathbb{Z}$
- We don't have an abstract value whose concretization includes  $\mathbb{Z}$ , so we add one:  $\top$

$\oplus$	-	0	+	$\top$
-	-	-	$\top$	$\top$
0	-	0	+	$\top$
+	$\top$	+	+	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

CS 263

12

## Examples

- Abstract computation might lose information

$$\llbracket (1+2) + -3 \rrbracket = 0$$

$$\sigma(\llbracket (1+2) + -3 \rrbracket) = (\sigma(1) \oplus \sigma(2)) \oplus \sigma(-3) = (+ \oplus +) \oplus - = \top$$

- We lose some precision
- But this will simplify the computation of the abstract answer in cases when the precise answer is not computable

CS 263

13

## Adding Division

- Fairly straightforward except for division by 0
  - We say that there is no answer in that case
  - $\gamma(+ \odot 0) = \{ n \mid n = n_1 / 0, n_1 > 0 \} = \emptyset$
- We introduce  $\perp$  to be the abstraction of the  $\emptyset$ 
  - We also use the same abstraction for non-termination!

$\odot$	-	0	+	$\top$	$\perp$
-	+	0	-	$\top$	$\perp$
0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
+	-	0	+	$\top$	$\perp$
$\top$	$\top$	$\top$	$\top$	$\top$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

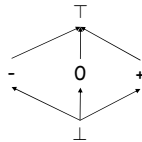
CS 263

14

## The Abstract Domain

- Our abstract domain forms a lattice
  - A partial order is induced by  $\gamma$ 

$$a_1 \leq a_2 \text{ iff } \gamma(a_1) \subseteq \gamma(a_2)$$
    - We say that  $a_1$  is more precise than  $a_2$ !
  - Every finite subset has a least-upper bound (lub) and a greatest-lower bound (glb)



CS 263

15

## Lattice Facts

- A lattice is complete when all subsets have lub and glb
  - Even infinite ones
- Every finite lattice is complete
- Every complete lattice is a domain
  - Since a chain is a subset
- Not every domain is a complete lattice
  - Might not even be a lattice

CS 263

16

## More Lattice Facts

- Early work in denotational semantics used lattices
  - But it was later seen that only chains need to have lub
  - And there was no need for  $\top$  and glb
- In abstract interpretation we'll use  $\top$  to denote "I don't know"
  - Corresponds to all values in the concrete domain

CS 263

17

## More Definitions

- We can start with the abstraction function

$$\beta : C \rightarrow A \text{ (maps a concrete value to the best abstract value)}$$
  - $A$  must be a lattice
- From here we can derive the concretization function

$$\gamma : A \rightarrow \mathcal{P}(C)$$

$$\gamma(a) = \{ x \in C \mid \beta(x) \leq a \}$$

CS 263

18

### Example

- Consider our sign lattice

$$\beta(n) = \begin{cases} + & \text{if } n > 0 \\ 0 & \text{if } n = 0 \\ - & \text{if } n < 0 \end{cases}$$

- $\gamma(a) = \{ n \mid \beta(n) \leq a \}$ 
  - Example:  $\gamma(+)=\{n \mid \beta(n) \leq +\} = \{n \mid \beta(n) = +\} = \{n \mid n > 0\}$
  - $\gamma(\top) = \{n \mid \beta(n) \leq \top\} = \mathbb{Z}$
  - $\gamma(\perp) = \{n \mid \beta(n) \leq \perp\} = \emptyset$

CS 263

19

### Abstraction and Concretization

- We argued that for correctness

$$\gamma(a_1 \text{ op } a_2) \supseteq \gamma(a_1) \text{ op } \gamma(a_2)$$

- We wish for the set on the left to be as small as possible
- To reduce the loss of information through abstraction
- For each set  $S \subseteq C$ , define  $\alpha(S)$  as follows:
  - Pick  $S'$  the smallest that includes  $S$  and is in the image of  $\gamma$
  - Define  $\alpha(S) = \gamma^{-1}(S')$
  - Then we define:  $a_1 \text{ op } a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2))$

- Another definition for the abstraction for sets

$$\alpha : \mathcal{P}(C) \rightarrow A$$

$$\alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \}$$

CS 263

20

### Example

- Consider our sign lattice

$$\beta(n) = \begin{cases} + & \text{if } n > 0 \\ 0 & \text{if } n = 0 \\ - & \text{if } n < 0 \end{cases}$$

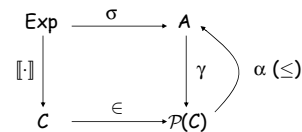
- $\alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \}$ 
  - Example:  $\alpha(\{1, 2\}) = \text{lub} \{ + \} = +$
  - $\alpha(\{1, 0\}) = \text{lub} \{ +, 0 \} = \top$
  - $\alpha(\emptyset) = \text{lub} \{ \} = \perp$

CS 263

21

### Correctness Condition

- In general, abstract interpretation satisfies the following diagram

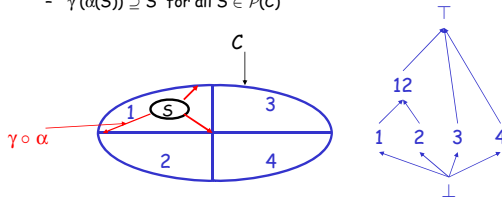


CS 263

22

### Galois Connections

- A Galois connection between complete lattices  $A$  and  $\mathcal{P}(C)$  is a pair of functions  $\alpha$  and  $\gamma$  such that:
  - $\gamma$  and  $\alpha$  are monotonic (with the  $\subseteq$  ordering on  $\mathcal{P}(C)$ )
  - $\alpha(\gamma(a)) = a$  for all  $a \in A$
  - $\gamma(\alpha(S)) \supseteq S$  for all  $S \in \mathcal{P}(C)$



CS 263

23

### More on Galois Connections

- All Galois connections are monotonic
- In a Galois connection one function uniquely determines the other

CS 263

24

### Correctness Conditions

- Three conditions define a correct abstract interpretation
- 1.  $\alpha$  and  $\gamma$  are monotonic
- 2.  $\alpha$  and  $\gamma$  form a Galois connection
- 3. Abstraction of operations is correct  

$$a_1 \text{ op } a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2))$$

CS 263

25

### Review

- We introduced abstract interpretation
- An abstraction mapping from concrete to abstract values
  - Has a concretization mapping which forms a Galois connection
- Next: We push these ideas from expressions to programs

CS 263

26

### Abstract Interpretation for Imperative Programs

- So far we abstracted the value of expressions
- We want now to abstract the state at each point in the program
- First we define the concrete semantics that we are abstracting
  - We use a collecting semantics

CS 263

27

### The Collecting Semantics

- Recall
  - A state  $\sigma \in \Sigma = \text{Var} \rightarrow \mathbb{Z}$
  - States vary from program point to program point
- We introduce a set of program points: Labels
- We want to answer questions like:
  - Is  $x$  always positive at label  $i$ ?
  - Is  $x$  always greater or equal to  $y$  at label  $j$ ?
- To answer these questions it helps to construct  

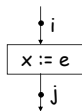
$$C \in \text{Contexts} = \text{Labels} \rightarrow \mathcal{P}(\Sigma)$$
  - For each label, all the states at that label
  - This is called the collecting semantics of the program
- How can we define the collecting semantics?

CS 263

28

### Defining the Collecting Semantics

- We first define relations between the collecting semantics at different labels
  - We do it for a flowchart program
  - It can be done for IMP with careful definition of program points
- Define a label on each edge in the flowchart
- For assignment



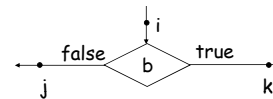
$$C_j = \{\sigma[x := n] \mid \sigma \in C_i \wedge \llbracket e \rrbracket \sigma = n\}$$

CS 263

29

### Defining the Collecting Semantics

- For conditionals



$$C_j = \{\sigma \mid \sigma \in C_i \wedge \llbracket b \rrbracket \sigma = \text{false}\}$$

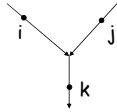
$$C_k = \{\sigma \mid \sigma \in C_i \wedge \llbracket b \rrbracket \sigma = \text{true}\}$$

CS 263

30

### Defining the Collecting Semantics

- For a join



$$C_k = C_i \cup C_j$$

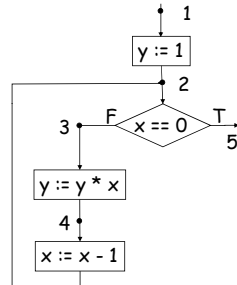
- Verify that these relations are monotonic
  - If we increase a  $C_i$ , all other  $C_j$  can only increase

CS 263

31

### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



$$\begin{aligned} C_1 &= \{\sigma \mid \sigma(x) \geq 0\} \\ C_2 &= \{\sigma[y:=1] \mid \sigma \in C_1\} \\ &\quad \cup \{\sigma[x:=\sigma(x)-1] \mid \sigma \in C_4\} \\ C_3 &= C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\ C_5 &= C_2 \cap \{\sigma \mid \sigma(x) = 0\} \\ C_4 &= \{\sigma[y:=\sigma(y)*\sigma(x)] \mid \sigma \in C_3\} \end{aligned}$$

CS 263

32

### The Collecting Semantics

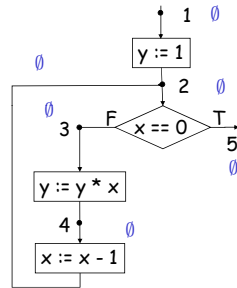
- We have an equation with the unknown  $C$ 
  - The equation is defined by a monotonic and continuous function on the domain Labels  $\rightarrow \mathcal{P}(\Sigma)$
- We can use the least fixed-point theorem
  - We start with  $C^0 = \lambda L. \emptyset$
  - We apply the relations between  $C_i$  and  $C_j$  to construct  $C_i^k$  from  $C_j^{k-1}$
  - We stop when  $C^k = C^{k-1}$
  - The problem is that we'll go on forever for most programs
  - But we know the fixed point exists

CS 263

33

### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



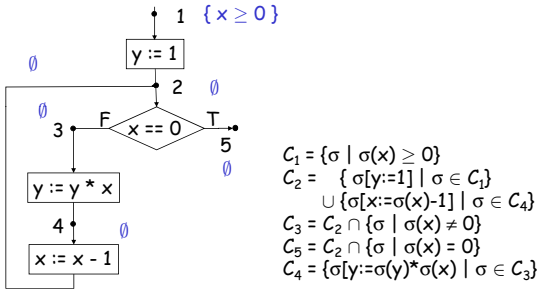
$$\begin{aligned} C_1 &= \{\sigma \mid \sigma(x) \geq 0\} \\ C_2 &= \{\sigma[y:=1] \mid \sigma \in C_1\} \\ &\quad \cup \{\sigma[x:=\sigma(x)-1] \mid \sigma \in C_4\} \\ C_3 &= C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\ C_5 &= C_2 \cap \{\sigma \mid \sigma(x) = 0\} \\ C_4 &= \{\sigma[y:=\sigma(y)*\sigma(x)] \mid \sigma \in C_3\} \end{aligned}$$

CS 263

34

### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



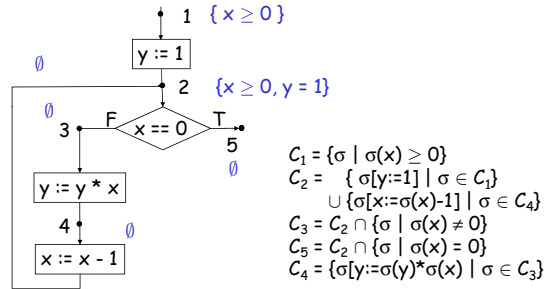
$$\begin{aligned} C_1 &= \{\sigma \mid \sigma(x) \geq 0\} \\ C_2 &= \{\sigma[y:=1] \mid \sigma \in C_1\} \\ &\quad \cup \{\sigma[x:=\sigma(x)-1] \mid \sigma \in C_4\} \\ C_3 &= C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\ C_5 &= C_2 \cap \{\sigma \mid \sigma(x) = 0\} \\ C_4 &= \{\sigma[y:=\sigma(y)*\sigma(x)] \mid \sigma \in C_3\} \end{aligned}$$

CS 263

35

### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



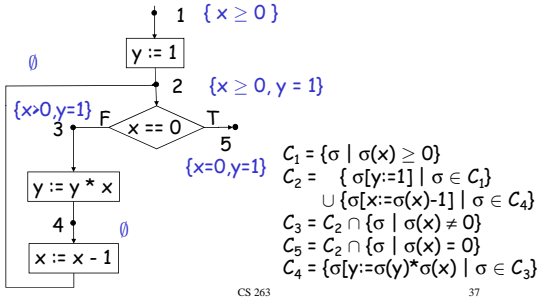
$$\begin{aligned} C_1 &= \{\sigma \mid \sigma(x) \geq 0\} \\ C_2 &= \{\sigma[y:=1] \mid \sigma \in C_1\} \\ &\quad \cup \{\sigma[x:=\sigma(x)-1] \mid \sigma \in C_4\} \\ C_3 &= C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\ C_5 &= C_2 \cap \{\sigma \mid \sigma(x) = 0\} \\ C_4 &= \{\sigma[y:=\sigma(y)*\sigma(x)] \mid \sigma \in C_3\} \end{aligned}$$

CS 263

36

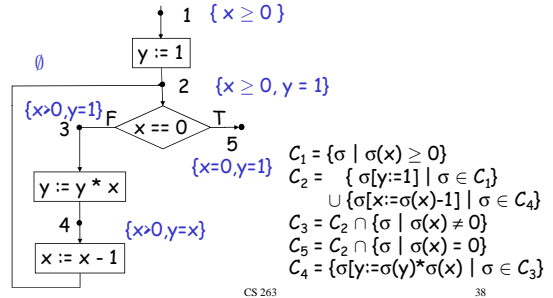
### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



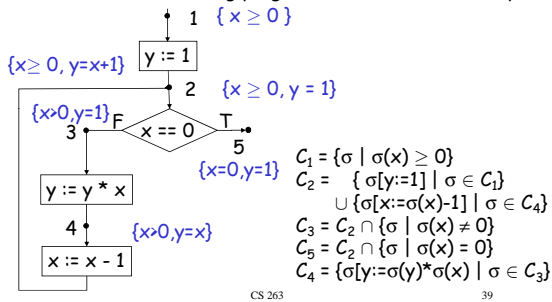
### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



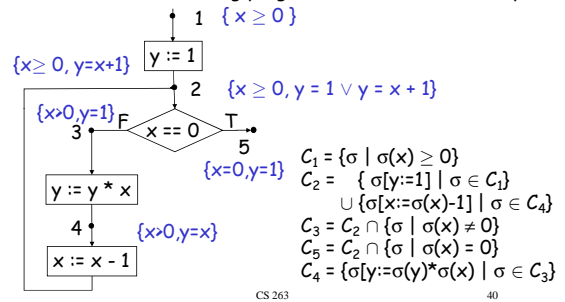
### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



### Collecting Semantics: Example

- Consider the following program (assume  $x \geq 0$  initially)



### Abstract Interpretation

- We pick a complete lattice  $A$  (abstractions for  $\mathcal{P}(\Sigma)$ )

- Along with a monotonic abstraction  $\alpha: \mathcal{P}(\Sigma) \rightarrow A$
- Alternatively, pick  $\beta: \Sigma \rightarrow A$
- This uniquely defines its Galois connection  $\gamma$

- We take the relations between  $C_i$  and move them to the abstract domain:

$$a \in \text{Labels} \rightarrow A$$

- Assignment

Concrete:  $C_j = \{\sigma[x := n] \mid \sigma \in C_i \wedge \llbracket e \rrbracket \sigma = n\}$   
 Abstract:  $a_j = \alpha \{ \sigma[x := n] \mid \sigma \in \gamma(a_i) \wedge \llbracket e \rrbracket \sigma = n \}$

CS 263

41

### Abstract Interpretation

- Conditional

Concrete:  $C_j = \{\sigma \mid \sigma \in C_i \wedge \llbracket b \rrbracket \sigma = \text{false}\}$  and  
 $C_k = \{\sigma \mid \sigma \in C_i \wedge \llbracket b \rrbracket \sigma = \text{true}\}$

Abstract:  $a_j = \alpha \{ \sigma \mid \sigma \in \gamma(a_i) \wedge \llbracket b \rrbracket \sigma = \text{false}\}$  and  
 $a_k = \alpha \{ \sigma \mid \sigma \in \gamma(a_i) \wedge \llbracket b \rrbracket \sigma = \text{true}\}$

- Join

Concrete:  $C_k = C_i \cup C_j$   
 Abstract:  $a_k = \alpha (\gamma(a_i) \cup \gamma(a_j)) = \text{lub} \{a_i, a_j\}$

CS 263

42

### Least Fixed-Points in the Abstract Domain

- Now we have a recursive equation with unknown "a"
  - Defined by a monotonic and continuous function on the domain  $Labels \rightarrow A$
- We can use the least fixed-point theorem:
  - Start with  $a^0 = \lambda L. \perp$
  - Apply the monotonic function to compute  $a^{k+1}$  from  $a^k$
  - Stop when  $a^{k+1} = a^k$
- Exactly the same computation as for the collecting semantics
  - What is new ?

CS 263

43

### Least Fixed Point in Abstract Domain

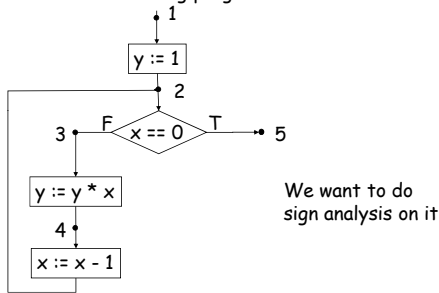
- We have a hope of termination
  - The classic setup is when  $A$  has only uninteresting chains (finite number of elements in each chain)
    - We say that  $A$  has finite height (say  $h$ )
- In this case the computation takes at most  $O(h * |Labels|^2)$  steps
  - At each step "a" makes progress on at least one label
  - We can only make progress  $h$  times
  - And each time we must compute  $|Labels|$  elements
- This is a quadratic analysis: good news

CS 263

44

### Abstract Interpretation: Example

- Consider the following program



CS 263

45

### The Abstract Domain for Sign Analysis

- Consider the complete lattice  $S = \{ \perp, -, 0, +, \top \}$
- From it construct the complete lattice  $A = \{x, y\} \rightarrow S$ 
  - With point-wise ordering as usual
  - The abstract state consists of the sign for  $x$  and  $y$
- We start with  $a^0 = \lambda L. \lambda v \in \{x, y\}. \perp$

CS 263

46

### Example

Label	Iterations $\rightarrow$									
1	x	+								+
	y	$\top$								$\top$
2	x	$\perp$	+			$\top$				$\top$
	y	$\perp$	+					$\top$		$\top$
3	x	$\perp$		+		$\top$				$\top$
	y	$\perp$		+				$\top$		$\top$
4	x	$\perp$			+		$\top$			$\top$
	y	$\perp$			+		$\top$			$\top$
5	x	$\perp$				0				0
	y	$\perp$				+			$\top$	$\top$

CS 263

47

### Notes

- We abstracted the state of each variable independently
  - $A = \{x, y\} \rightarrow \{ \perp, -, 0, +, \top \}$
- We lost relationships between variables
  - E.g., that at a point  $x$  and  $y$  are always of the same sign
  - In the previous abstraction we get  $\{x := \top, y := \top\}$  at 2
- We can also abstract the state as a whole
  - $A = \mathcal{P}(\{ \perp, -, 0, +, \top \} \times \{ \perp, -, 0, +, \top \})$
  - For the previous example we now get the abstraction  $\{(-, -), (0, 0), (+, +)\}$  at 2

CS 263

48

## Other Abstract Domains

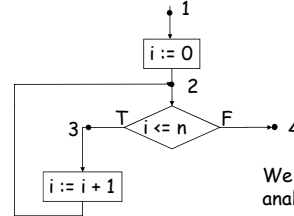
- Range analysis
  - Lattice of ranges:  $R = \{ \perp, [n..m], (-\infty, m], [n, +\infty), \top \}$
  - It is a complete lattice
    - $[n..m] \sqcup [n'..m'] = [\min(n, n').. \max(m, m')]$
    - $[n..m] \cap [n'..m'] = [\max(n, n').. \min(m, m')]$
    - With appropriate care in dealing with  $\infty$
  - $\beta : \mathbb{Z} \rightarrow R$  such that  $\beta(n) = [n..n]$
  - $\alpha : \mathcal{P}(\mathbb{Z}) \rightarrow R$  such that  $\alpha(S) = \text{lub} \{ \beta(n) \mid n \in S \} = [\min(S).. \max(S)]$
  - $\gamma : R \rightarrow \mathcal{P}(\mathbb{Z})$  such that  $\gamma(r) = \{ n \mid n \in r \}$
- This lattice has infinite-height chains
  - So the abstract interpretation might not terminate!

CS 263

49

## Example of Non-Termination

- Consider this (common) program fragment



We want to do range analysis for it

CS 263

50

## Example of Non-Termination

- Consider the sequence of abstract states at point 2
  - $[0..0], [0..1], [0..2], \dots$
  - The analysis never terminates
  - Or terminates very late if the loop bound is known statically
- It is time to approximate even more: widening
- We redefine the join (lub) operator of the lattice to ensure that from  $[0..0]$  upon union with  $[1..1]$  the result is  $[0..+\infty)$  and not  $[0..1]$
- Now the sequence of states is
  - $[0..0], [0, +\infty), [0, +\infty)$  Done (no more infinite chains)

CS 263

51

## Other Abstract Domains

- Linear relationships between variables
  - A convex polyhedron is a subset of  $\mathbb{Z}^k$  whose elements satisfy a number of inequalities:  $a_1 x_1 + a_2 x_2 + \dots + a_k x_k \geq c$
  - This is a complete lattice. Use linear programming methods for computing lub
- Linear relationships with at most two variables
  - Like convex polyhedra but with at most two variables per constraint
  - Octagons:  $x \pm y \preceq c$  have efficient algorithms
- Modulo constraints
  - E.g. even and odd

CS 263

52

## Summary of Abstract Interpretation

- AI is a very powerful technique that underlies a large number of program analyses
- AI can also be applied to functional and logic programming languages
- There are a few success stories
  - Strictness analysis for lazy functional languages
  - PolySpace for linear constraints
- In most other cases however AI is still slow
- When the lattices have infinite height and widening heuristics are used the result becomes unpredictable

CS 263

53