ETH Zürich

Chair of Software Engineering

Software Verification

Sebastian Nanz

Lecture 8: Abstract Interpretation

ETH Zürich

Chair of Software Engineering

Abstract Interpretation

Introduction

One framework to rule them all

- In the past lectures we have introduced a particular style of program analysis: data flow analysis.

- For these types of analyses, and others, a main concern is correctness: how do we know that a particular analysis produces sound results (does not miss possible errors)?

- In the following we discuss abstract interpretation, a general framework for describing program analyses and reasoning about their correctness.

Main ideas: Concrete computations

- An ordinary program describes computations in some concrete domain of values.
  - **Example:** program states that record the integer value of every program variable.

$$\sigma \in \text{State} = \text{Var} \to Z$$

- Possible computations can be described by the concrete semantics of the programming language used.

Main ideas: Abstract computations

- Abstract interpretation of a program describes computation in a different, abstract domain.
  - **Example:** program states that only record a specific property of integers, instead of their value: their sign, whether they are even/odd, or contained in [-32768, 32767] etc.

$$\sigma \in \text{AbstractState} = \text{Var} \to \{\text{even}, \text{odd}\}$$

- In order to obtain abstract computations, an abstract semantics for the programming language has to be defined.
- Abstract interpretation provides a framework for proving that the abstract semantics is sound with respect to the concrete semantics.

The collecting semantics

We assume the state of a program to be modeled as:

$$\sigma \in \text{State} = \text{Var} \to \mathbb{Z}$$

We will use the following notation for function update:

$$\sigma[x \mapsto k](y) = \begin{cases} k & \text{if } x = y \\ \sigma(y) & \text{otherwise} \end{cases}$$

We will write $[\![e]\!]\sigma$ to denote the value of an expression e in state σ.

We construct the collecting semantics as a function which gives for every program label the set of all possible states.

$$C : \text{Labels} \to \wp(\text{State})$$

Rules of the collecting semantics

$$\begin{tikzcd} {} \arrow[d, "l"] \\ {[x := e]} \arrow[d, "l'"] \\ {} \end{tikzcd}$$

$$C_{l'} = \{\sigma[x \mapsto n] \mid \sigma \in C_l \text{ and } [\![e]\!]\sigma = n\}$$

$$\begin{tikzcd} {} \arrow[d, "l"] & \\ {[b]} \arrow[r, "l_{false}"] \arrow[d, "l_{true}"] & {} \\ {} & \end{tikzcd}$$

$$C_{ltrue} = \{\sigma \mid \sigma \in C_l \text{ and } [\![b]\!]\sigma = true\}$$

$$C_{lfalse} = \{\sigma \mid \sigma \in C_l \text{ and } [\![b]\!]\sigma = false\}$$

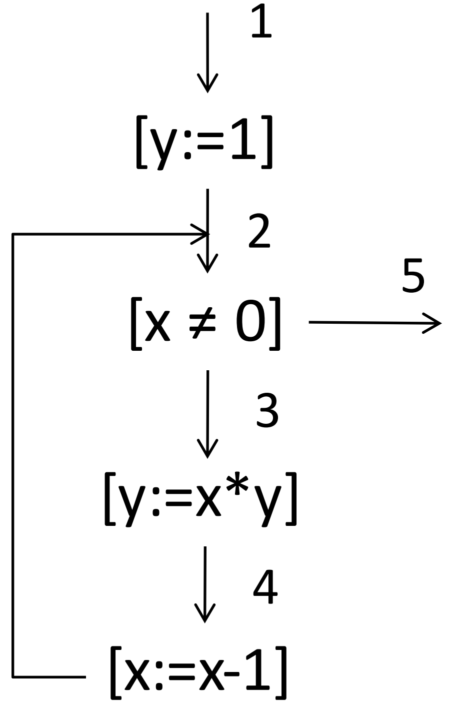
$$\begin{tikzcd} l_1 \arrow[dr] & & l_2 \arrow[dl] \\ & {} \arrow[d] & \\ & l & \end{tikzcd}$$

$$C_l = C_{l1} \cup C_{l2}$$

Note: In difference to the lecture on data flow analysis, labels are not on blocks, but on edges.

Example: Collecting semantics

Assume x > 0.



$C_1 = \{\sigma \mid \sigma(x) > 0\}$

$C_2 = \{\sigma[y \mapsto 1] \mid \sigma \in C_1\} \cup$
$\quad \{\sigma[x \mapsto \sigma(x) - 1] \mid \sigma \in C_4\}$

$C_3 = C_2 \cap \{\sigma \mid \sigma(x) \neq 0\}$

$C_4 = \{\sigma[y \mapsto \sigma(x) \cdot \sigma(y)] \mid \sigma \in C_3\}$

$C_5 = C_2 \cap \{\sigma \mid \sigma(x) = 0\}$

Solving the equations

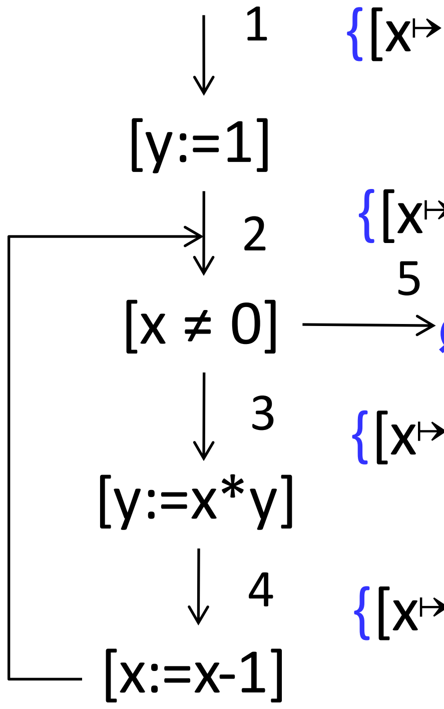
- The equation system we obtain has variables $C_1, \ldots, C_5$ which are interpreted over the complete lattice $\wp(\text{State})$.
- We can express the equation system as a monotone function $F : \wp(\text{State})^5 \to \wp(\text{State})^5$

$$F(C_1, \ldots, C_5) = (\{\sigma \mid \sigma(x) > 0\}, \ldots, C_2 \cap \{\sigma \mid \sigma(x) = 0\})$$

- Using Tarski's Fixed Point Theorem, we know that a least fixed point exists.
- We have seen: The least fixed point can be computed by repeatedly applying F, starting with the bottom element $\bot = (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)$ of the complete lattice until stabilization.

$$F(\bot) \sqsubseteq F(F(\bot)) \sqsubseteq \ldots \sqsubseteq F^n(\bot) = F^{n+1}(\bot)$$

Example: Fixed Point Computation



1: $\{[x \mapsto m, y \mapsto n] \mid m > 0\}$

2: $\{[x \mapsto m, y \mapsto 1] \mid m > 0\} \quad \cup \ \{[x \mapsto m-1, y \mapsto m] \mid m > 0\}$

5: $\varnothing \ \{[x \mapsto 0, y \mapsto m] \mid m > 0\}$ ... etc.

3: $\{[x \mapsto m, y \mapsto 1] \mid m > 0\}$

4: $\{[x \mapsto m, y \mapsto m] \mid m > 0\}$

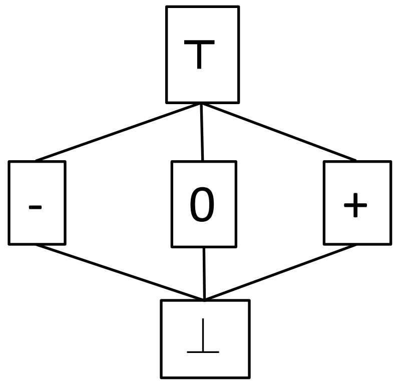
$$C_1 = \{\sigma \mid \sigma(x) > 0\}$$
$$C_2 = \{\sigma[y \mapsto 1] \mid \sigma \in C_1\} \cup \{\sigma[x \mapsto \sigma(x) - 1] \mid \sigma \in C_4\}$$
$$C_3 = C_2 \cap \{\sigma \mid \sigma(x) \neq 0\}$$
$$C_4 = \{\sigma[y \mapsto \sigma(x) \cdot \sigma(y)] \mid \sigma \in C_3\}$$
$$C_5 = C_2 \cap \{\sigma \mid \sigma(x) = 0\}$$

Domain for Sign Analysis

We want to focus on the sign of integers, using the domain

$\sigma \in$ AbstractState = Var -> Signs

where Signs is the following structure:



T represents all integers

+ the positive integers

- the negative integers

0 the set {0}

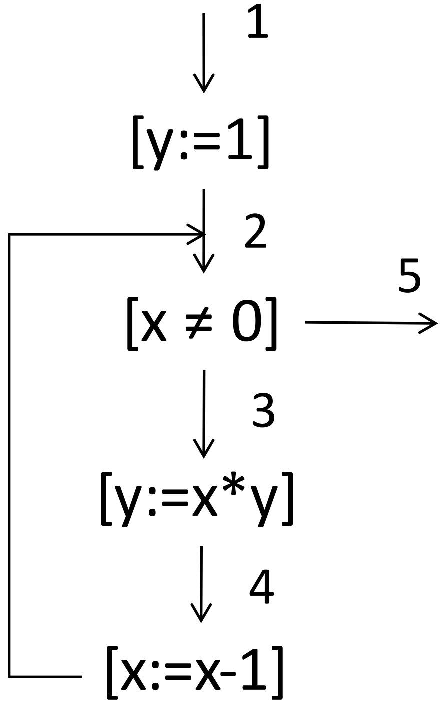
⊥ the empty set

How is such a structure called?

A complete lattice

Example: Sign Analysis

Assume x > 0. Use the abstract domain for sign analysis.



$A_1 = [x \mapsto +, y \mapsto \top]$

$A_2 = A_1[y \mapsto +] \sqcup$
$\quad A_4[x \mapsto A_4(x) \ominus +]$

$A_3 = A_2$

$A_4 = A_3[y \mapsto A_3(x) \otimes A_3(y)]$

$A_5 = A_2 \sqcap [x \mapsto 0, y \mapsto \top]$

Abstract Interpretation

Foundations

Introductory example: Expressions

A little language of expressions

Syntax

e ::= n | e * e

Concrete semantics

C[n] = n

C[e * e] = C[e] · C[e]

Example

C[-3 * 2 * -5] = C[-3 * 2] · C[-5] = C[-3 * 2] · (-5) = ... = 30

Introductory example: Abstraction

Assume that we are not interested in the value of an expression but only in its sign:

- Negative: –
- Zero: 0
- Positive: +

Abstract semantics

A[n] = sign(n)

A[e * e] = A[e] ⊗ A[e]

⊗	-	0	+
-	+	0	-
0		0	0
+			+

Example

A[-3 * 2 * -5] = A[-3 * 2] ⊗ A[-5] = A[-3 * 2] ⊗ (-) = ... =

= (-) ⊗ (+) ⊗ (-) = (+)

Introductory example: Soundness

- We want to express that the abstract semantics correctly describes the sign of a corresponding concrete computation.
- For this we first link each concrete value to an abstract value:

Representation function

$\beta : Z \to \{-, 0, +\}$

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

Introductory example: Soundness

- Conversely, we can also link abstract values to the set of concrete values they describe:

Concretization function

$\gamma : \{-, 0, +\} \rightarrow \wp(Z)$

$$\gamma(s) = \begin{cases} \{n \mid n < 0\} & \text{if } s = \mathbf{-} \\ \{0\} & \text{if } s = \mathbf{0} \\ \{n \mid n > 0\} & \text{if } s = \mathbf{+} \end{cases}$$

- Soundness then describes intuitively that the concrete value of an expression is described by its abstract value:

$$\forall e.\ C[e] \in \gamma(A[e])$$

Extending the language

Syntax

e ::= n | e * e | e + e | -e

Abstract semantics

A[n] = sign(n)

A[-e] = ⊖A[e]

A[e + e] = A[e] ⊕ A[e]

	-	0	+
⊖	+	0	-

⊕	-	0	+
-	-	-	?
0		0	+
+			+

Observation: The abstract domain {-,0,+} is not closed under the interpretation of addition.

Extending the abstract domain

We have to introduce an additional abstract value:

⊤ "top" – (any value)

⊕	-	0	+	⊤
-	-	-	⊤	⊤
0		0	+	⊤
+			+	⊤
⊤				⊤

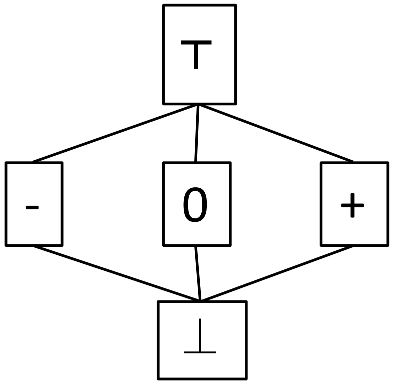
The new abstract domain

We can extend the concretization function to the new abstract domain {-,0,+, ⊤, ⊥} (add ⊥ for completeness):

$\gamma(\top) = \mathbf{Z}$ $\gamma(\bot) = \varnothing$

We obtain the following structure when drawing the partial order induced by

$a \leq b$ iff $\gamma(a) \subseteq \gamma(b)$



How is such a structure called?

A complete lattice

Construction of complete lattices

- If we know some complete lattices, we can construct new ones by combining them
- Such constructions become important when designing new analyses with complex analysis domains

Example: Total function space

Let $(D_1, \sqsubseteq_1)$ be a complete lattice and let S be a set. Then $(D, \sqsubseteq)$, defined as follows, is a complete lattice:

- $D = S \rightarrow D_1$ ("space of total functions")
- $f \sqsubseteq f'$ iff $\forall\ s \in S : f(s) \sqsubseteq_1 f'(s)$ ("point-wise ordering")

The framework of abstract interpretation

- Starting from a concrete domain C, define an abstract domain $(A, \sqsubseteq)$, which must be a complete lattice
- Define a representation function β that maps a concrete value to its best abstract value

$$\beta : C \to A$$

- From this we can derive the concretization function γ

$$\gamma : A \to \wp(C)$$

$$\gamma(a) = \{c \in C \mid \beta(c) \sqsubseteq a\}$$

and abstraction function α for sets of concrete values

$$\alpha : \wp(C) \to A$$

$$\alpha(C) = \bigsqcup \{\beta(c) \mid c \in C\}$$

Galois connections

- The following properties of α and γ hold:

Monotonicity

(1) α and γ are monotone functions

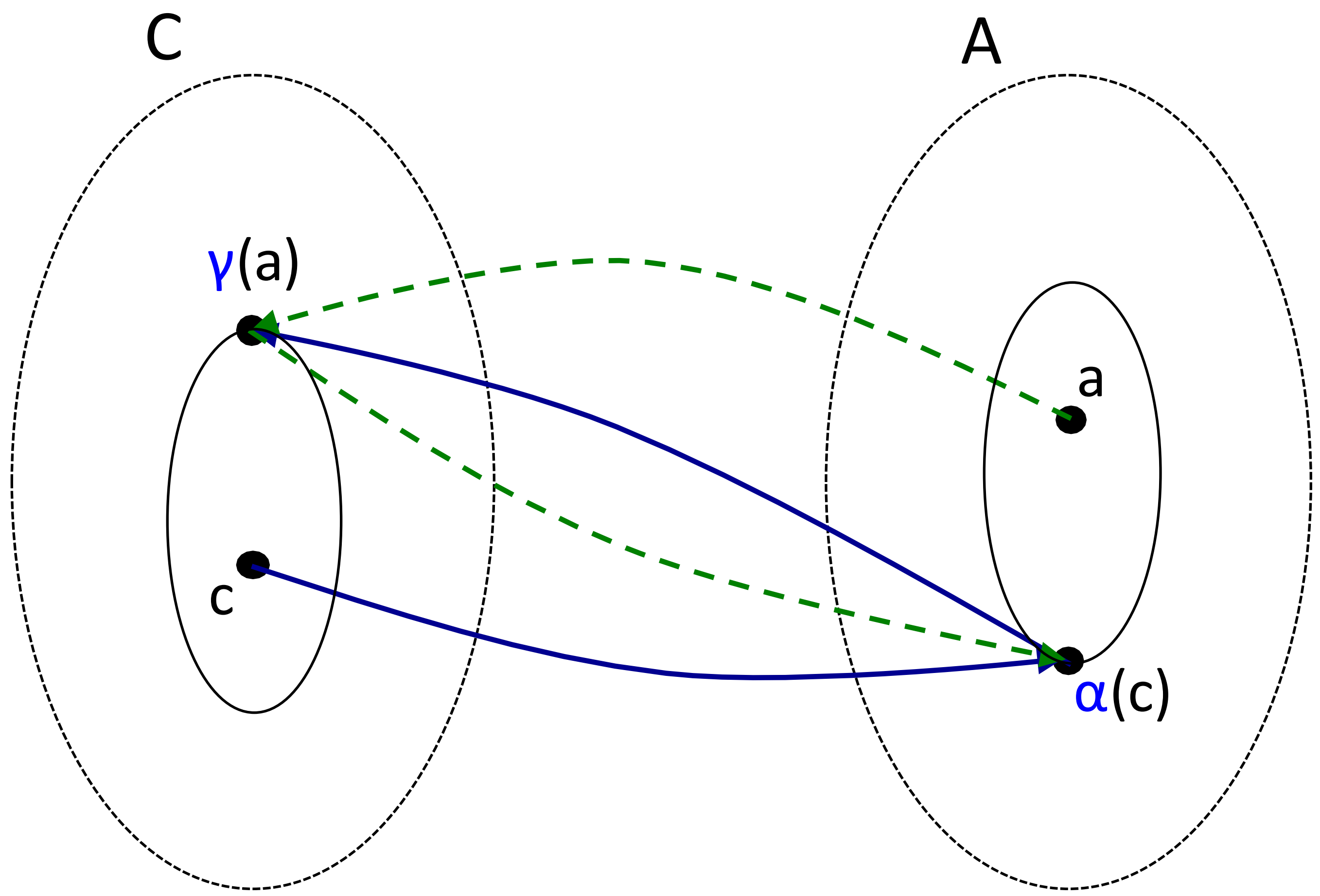
Galois connection

(2) $c \subseteq \gamma(\alpha(c))$ for all $c \in \wp(C)$

(3) $a \sqsupseteq \alpha(\gamma(a))$ for all $a \in A$

- Galois connection: This property means intuitively that the functions α and γ are "almost inverses" of each other.

Figure: Galois connection



Galois insertions

- For a Galois connection, there may be several elements of A that describe the same element in C
- As a result, A may contain elements which are irrelevant for describing C
- The concept of Galois insertion fixes this:

Monotonicity

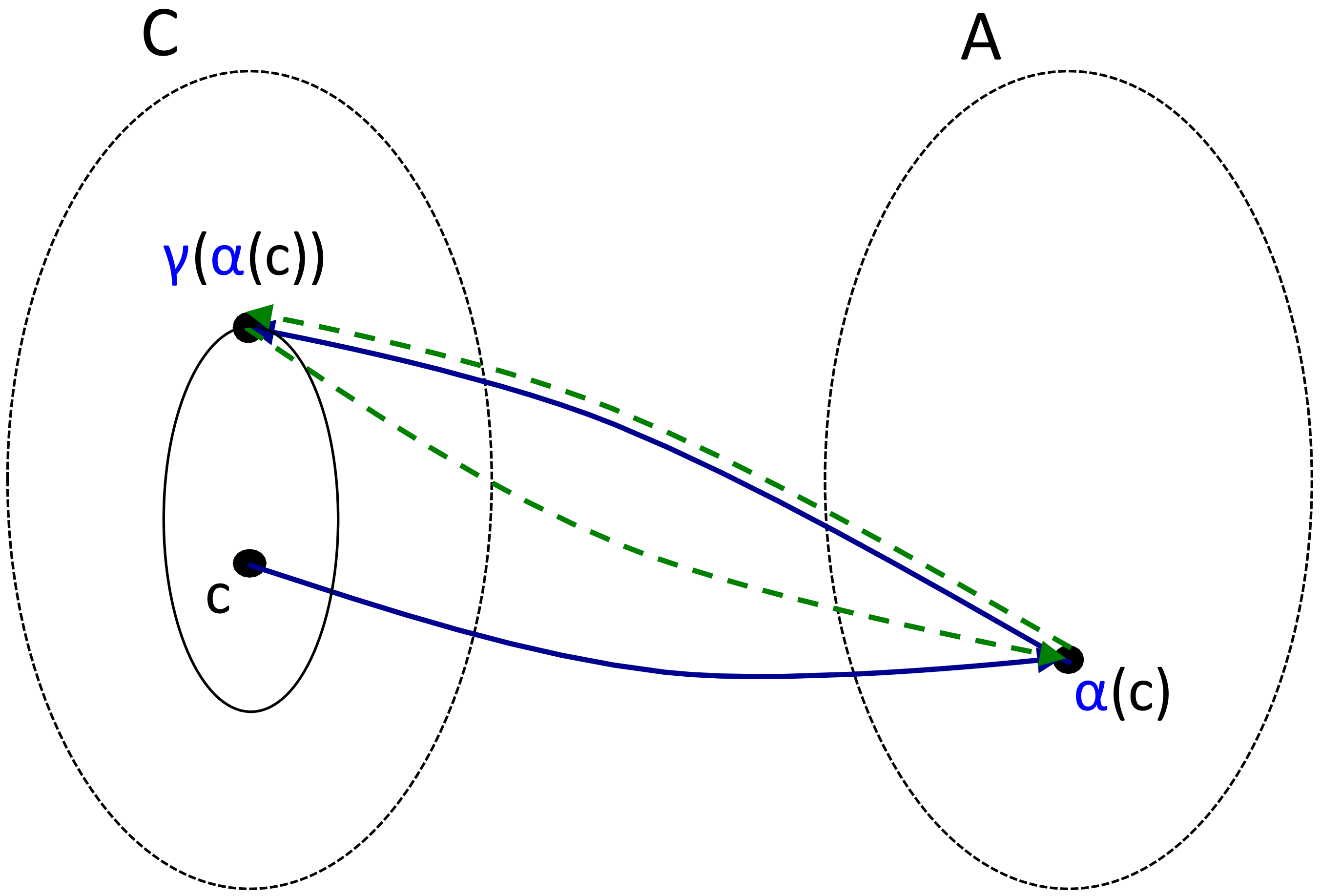
(1) α and γ are monotone functions

Galois insertion

(2) $c \subseteq \gamma(\alpha(c))$ for all $c \in \wp(C)$

(3) $a = \alpha(\gamma(a))$ for all $a \in A$

Figure: Galois insertion



Induced Operations

- A Galois connection can be used to induce the abstract operations from the concrete ones.

$$\begin{tikzcd} A \arrow[r, "\alpha \circ op \circ \gamma"] & A \arrow[d, "\gamma"] \\ \wp(C) \arrow[u, "\alpha"] \arrow[r, "op"'] & \wp(C) \end{tikzcd}$$

abstract execution

concrete execution

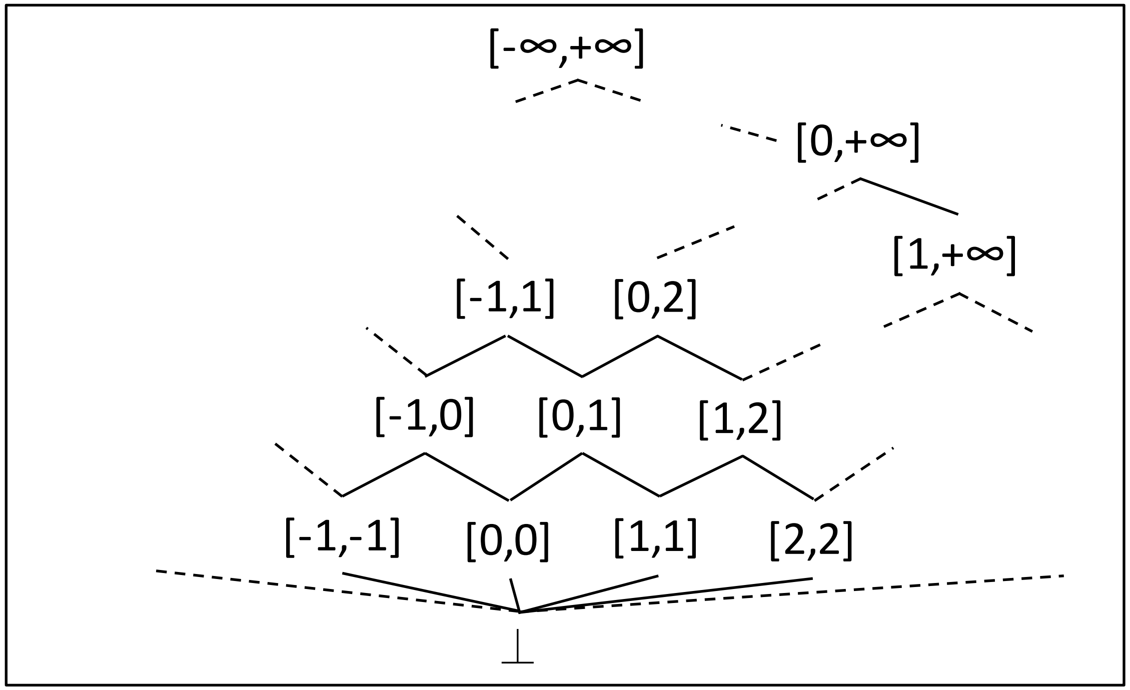
- We can show that the induced operation $\underline{\mathbf{op}} = \alpha \circ op \circ \gamma$ is the most precise abstract operation in this setting.
- The induced operation might not be computable. In this case we can define an upper approximation $op^{\#}$, $\underline{\mathbf{op}} \sqsubseteq op^{\#}$, and use this as abstract operation.

Abstract Interpretation

Widening

Range analysis

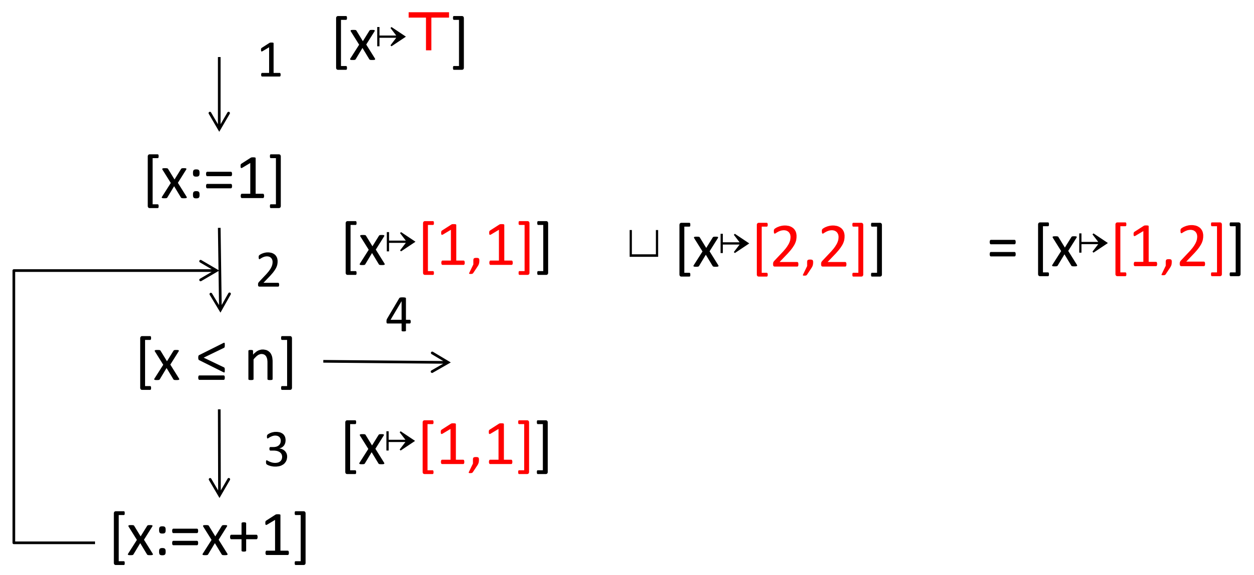
- To introduce the notion of widening, we have a look at range analysis, which provides for every variable an over-approximation of its integer value range.
- We are left with the task of choosing a suitable abstract domain: the interval lattice suggests itself.



Interval = $\{\bot\} \cup \{[x,y] \mid x \leq y,\ x \in \mathbf{Z} \cup \{\infty\},\ y \in \mathbf{Z} \cup \{\infty\}\}$

Example

Consider the following program:



➢ At program point 2, the following sequence of abstract states arises: [x↦[1,1]], [x↦[1,2]], [x↦[1,3]], ...

Consequence: The analysis never terminates (or, if n is statically known, converges only very slowly).

The ascending chain condition

- Using an arbitrary complete lattice as abstract domain, the solution is not computable in general.
- The reason for that is the fact that the value space might be unbounded, containing infinite ascending chains:

  $(l_n)_n$ is such that $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \cdots$,

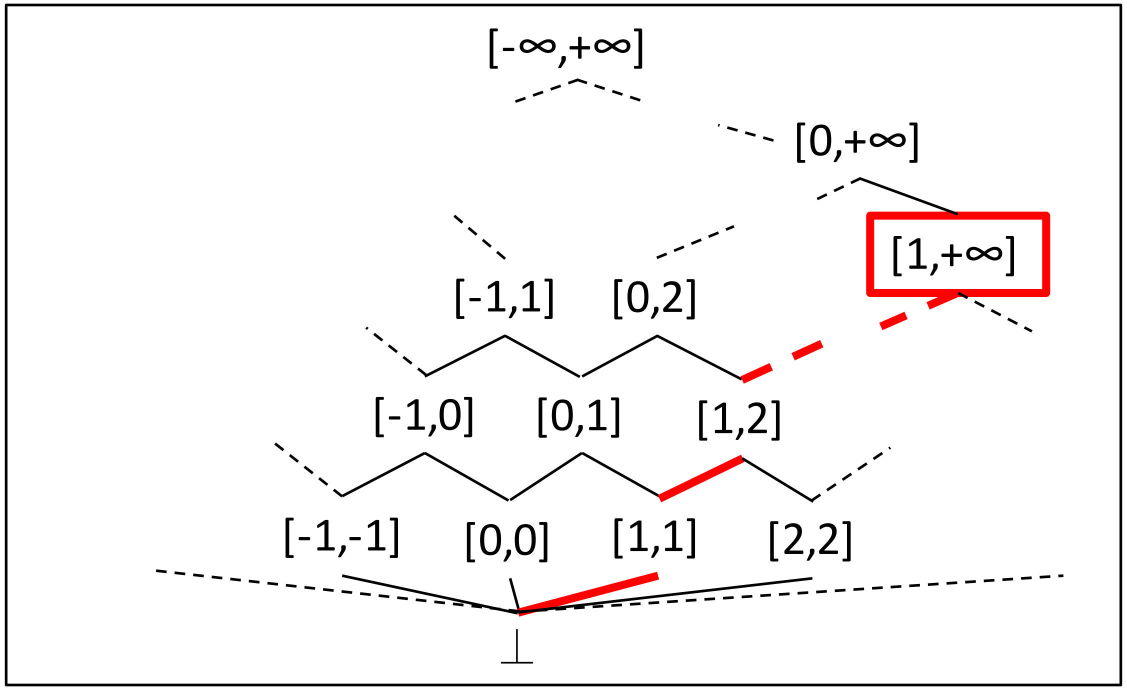
  but there exists *no* n such that $l_n = l_{n+1} = \cdots$
- If we replace it with an abstract space that is finite (or does not possess infinite ascending chains), then the computation is guaranteed to terminate.
- In general, we want an abstract domain to satisfy the ascending chain condition, i.e. each ascending chain eventually stabilises:

  if $(l_n)_n$ is such that $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \cdots$,

  then there exists n such that $l_n = l_{n+1} = \cdots$

Non-termination

- The reason for the non-termination in the example is that the interval lattice contains infinite ascending chains.



- **Trick, if we cannot eliminate ascending chains:** We redefine the join operator of the lattice to jump to the extremal value more quickly.

Before: $[1,1] \sqcup [2,2] = [1,2]$ Now: $[1,1] \nabla [2,2] = [1,+\infty]$

Widening

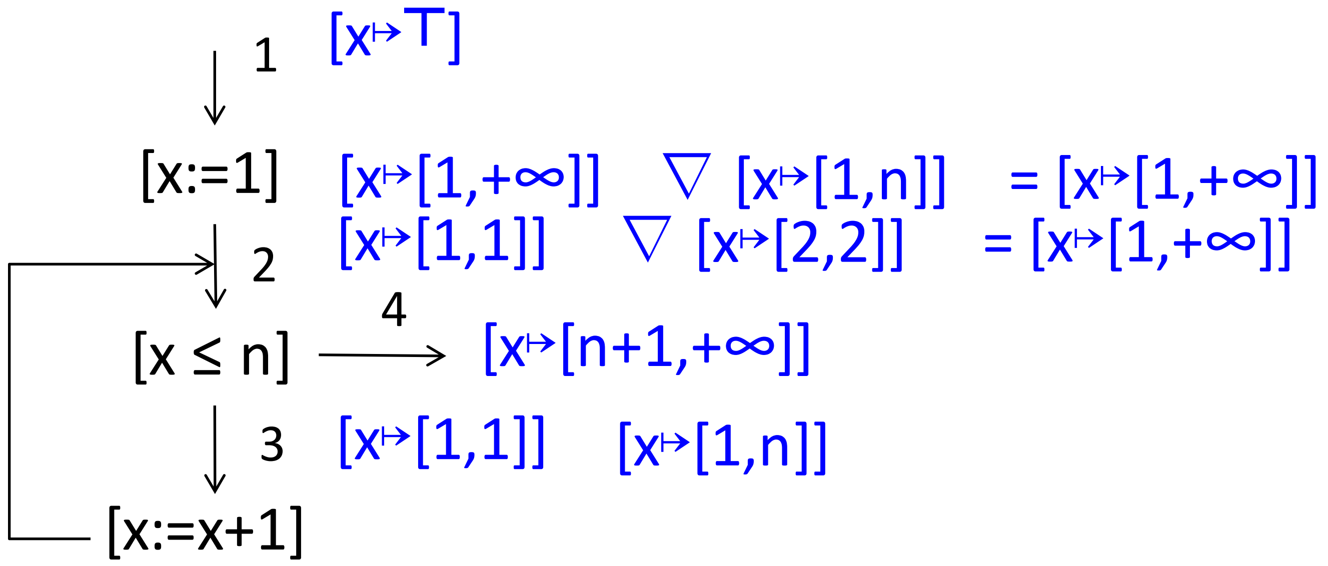
A widening ∇ : D x D -> D on a partially ordered set (D, $\sqsubseteq$) satisfies the following properties:

1. For all x, y $\in$ D. $x \sqsubseteq x \nabla y$ and $y \sqsubseteq x \nabla y$
2. For all ascending chains $x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \cdots$ the ascending chain $y_1 = x_1 \sqsubseteq y_2 = y_1 \nabla x_2 \sqsubseteq \cdots \sqsubseteq y_{n+1} = y_n \nabla x_{n+1}$ eventually stabilizes.

➢ Widening is used to accelerate the convergence towards an upper approximation of the least fixed point.

Example (continued)

- Assume we have a widening operator ∇ that is defined such that $[1,1] \nabla [2,2] = [1, +\infty]$



1
[x↦T]
[x:=1]
[x↦[1,+∞]] ∇ [x↦[1,n]] = [x↦[1,+∞]]
[x↦[1,1]] ∇ [x↦[2,2]] = [x↦[1,+∞]]
2
4
[x ≤ n]
[x↦[n+1,+∞]]
3
[x↦[1,1]]
[x↦[1,n]]
[x:=x+1]

- The analysis converges quickly.

Reading

Patrick Cousot and Radhia Cousot. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*. In: POPL'77, pages 238-252. ACM Press, 1977

Neil D. Jones, Flemming Nielson: *Abstract Interpretation: a Semantics-Based Tool for Program Analysis*, 1994

Flemming Nielson, Hanne Riis Nielson, Chris Hankin: *Principles of Program Analysis*, Springer, 2005.

Chapter 1: Section 1.5

Chapter 4 (advanced material)