

An Introduction to Numerical Relational Domains

Pietro Ferrara

Chair of Programming Methodology
ETH
Zurich, Switzerland

Analisi e Verifica di Programmi
Universita' Ca' Foscari, Venice, Italy

Outline

- 1. Recall/Introduction**
- 2. Abstract relational semantics**

Syntax

$C ::= C_1; C_2$
 $\text{if } (B) \text{ then } C_1 \text{ else } C_2$
 $\text{while}(B) C_1$
 $x = E$

$E ::= c$
 x
 $E_1 < \text{op} > E_2$

$B ::= \text{true}$
 false
 $B_1 \text{ OR } B_2$
 $B_1 \text{ AND } B_2$
 $E_1 < \text{comp} > E_2$

Concrete and Abstract Domain

- Concrete domain:

$$\text{Env} : [\text{Var} \rightarrow \mathbb{N}]$$
$$\langle \wp(\text{Env}), \subseteq, \emptyset, \text{Env}, \cup, \cap \rangle$$

- Abstract non-relational domain:

$$\overline{\text{Env}} : [\text{Var} \rightarrow \overline{\mathbb{N}}]$$

$$\langle \wp(\mathbb{N}), \subseteq, \emptyset, \mathbb{N}, \cup, \cap \rangle \begin{array}{c} \xleftarrow{\gamma_{\overline{\mathbb{N}}}} \\ \xrightarrow{\alpha_{\overline{\mathbb{N}}}} \end{array} \langle \overline{\mathbb{N}}, \leq_{\overline{\mathbb{N}}}, \perp, \top, \sqcup_{\overline{\mathbb{N}}}, \sqcap_{\overline{\mathbb{N}}} \rangle$$

Non-Relational Domains

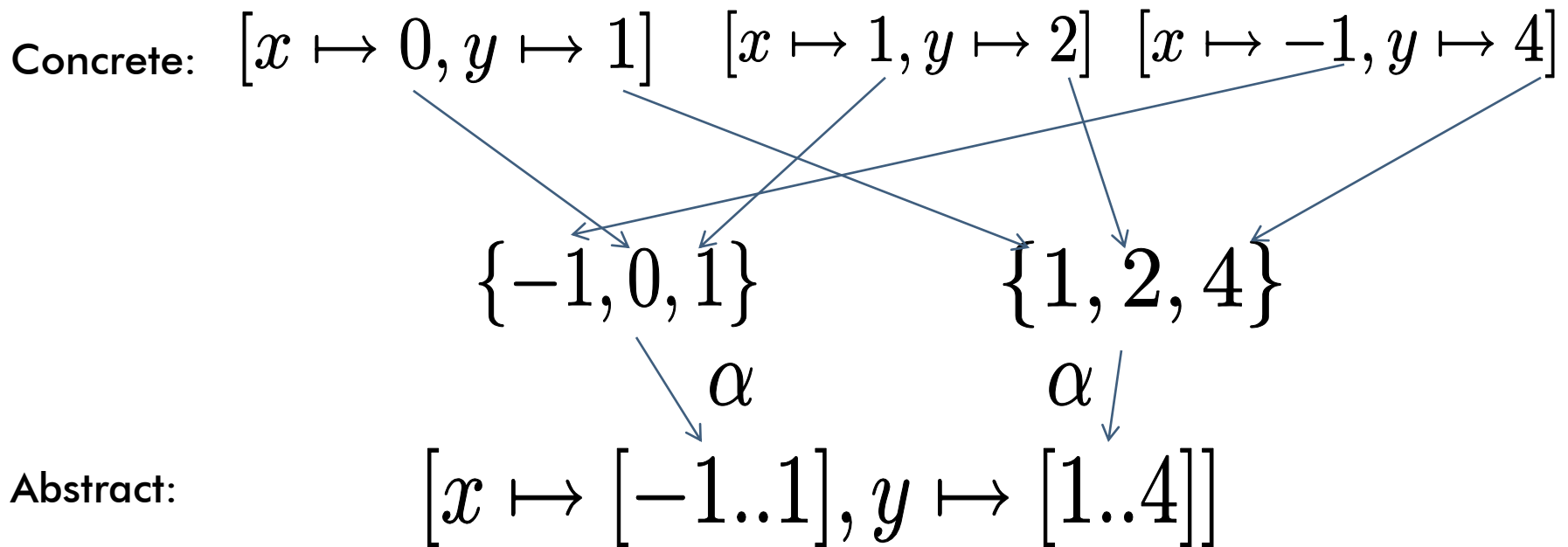
- A non-relational domain has to provide:
 - > Operators on lattices
 - Partial ordering
 - Upper and lower bound
 - Top and bottom elements
 - > Abstraction and concretization functions
 - > *eval_const*
 - > *eval_arithm*
 - > *eval_cond*

Summary on Non-Relational Domains

- **Many different domains:**
 - > Sign
 - > Parity
 - > Congruences
 - > Integers
 - > Intervals
- **Work directly on values**
- **Evaluation of expressions and conditions**
- **No relational information!**

Why?

- Non-relational abstract domain quite similar to the concrete domain!
- Why it is not enough???



Abstraction

$[x \mapsto 0, y \mapsto 1] \quad [x \mapsto 1, y \mapsto 2] \quad [x \mapsto -1, y \mapsto 4]$

$\downarrow \alpha$

$[x \mapsto [-1..1], y \mapsto [1..4]]$

- **We loose the fact that:**
 - > when x was 0 , y was 1
 - > when x was 1 , y was 2
 - > when x was -1 , y was 4
- **i.e. that $y > x$!**
- **We (implicitly?) ignore relations**

An Example

$\text{arr.length} \mapsto [0.. + \infty]$

for(int i=0; i<arr.length; i++)

$i \mapsto [0..0]$ $i \mapsto [0..1]$ $i \mapsto [0..2]$ $i \mapsto [0.. + \infty]$

arr[i]=0;

$i \geq 0?$

↓

$[0.. + \infty] \geq [0..0]?$

true!

$i < \text{arr.length}?$

↓

$[0.. + \infty] < [0.. + \infty]?$

top!

- **We want to prove that:**

> i is positive when executing $\text{arr}[i]$

> i is less than the length array

Constraints Between Variables

- We need something more than values!
 - > Relations between variables
- Previous approach not well-suited

$$\overline{A}[\overline{E_1} < op > E_2, \overline{ev}] =$$
$$\overline{eval_arithm}(\overline{A}[\overline{E_1}, \overline{ev}], \overline{A}[\overline{E_2}, \overline{ev}], < op >)$$
$$\overline{T}[\overline{E_1} < comp > E_2, \overline{ev}] =$$
$$\overline{eval_cond}(\overline{A}[\overline{E_1}, \overline{ev}], \overline{A}[\overline{E_2}, \overline{ev}], < comp >)$$

What we need

- We need to
 - > Analyze assignments and conditions as a whole
 - i.e. without splitting their analysis
- The relational domain has to provide
 - > a primitive to assign an expression to a variable
 - > a primitive to evaluate a condition
- We need to define a different semantics
 - > Using these primitives

Intuition

- **State of a relational domain**

- > Set of constraints, e.g.:

- $i < \text{arr.length}$

- $i == j$

- $x + y > 5$

- **Concretization**

- > all the states that respect these constraints

$$\gamma(x + y > 5) = \{[x \mapsto 1, y \mapsto 5], [x \mapsto 7, y \mapsto -1], \dots\}$$

An Example

∅

```
for(int i=0; i<arr.length; i++)
```

$i < \text{arr.length}$

```
arr[i]=0;
```

$i < \text{arr.length}?$



$i < \text{arr.length}$ in the state
true!

- We want to prove that:
 - > i is less than the length array

Outline

1. Recall/Introduction
- 2. Abstract relational semantics**

Concatenation

$$\frac{\langle C_1, \bar{\sigma} \rangle \longrightarrow \bar{\sigma}_1}{\langle C_1; C_2, \bar{\sigma} \rangle \longrightarrow \langle C_2, \bar{\sigma}_1 \rangle}$$

Assignment

$$\frac{\overline{\sigma_1} = \overline{assign}(\overline{\sigma}, x, E)}{\langle x = E, \overline{\sigma} \rangle \rightarrow \overline{\sigma_1}}$$

If

$$\frac{\langle C_1, \overline{\text{testTrue}(\bar{\sigma}, B)} \rangle \longrightarrow \bar{\sigma}_1, \langle C_2, \overline{\text{testTrue}(\bar{\sigma}, !B)} \rangle \longrightarrow \bar{\sigma}_2}{\langle \text{if } (B) \text{ then } C_1 \text{ else } C_2, \bar{\sigma} \rangle \longrightarrow \bar{\sigma}_1 \sqcup \bar{\sigma}_2}$$

While

$$\frac{\overline{\sigma_1} = \text{lfp } \overline{\sigma_2}. \overline{\text{testTrue}}(\overline{\sigma}, !B) \sqcup \{\overline{\sigma_3} : \langle C_1, \overline{\text{testTrue}}(\overline{\sigma_2}, B) \rangle \longrightarrow \overline{\sigma_3}\}}{\langle \text{while } (B) C_1, \overline{\sigma} \rangle \longrightarrow \overline{\sigma_1}}$$

Summary

- A relational domain has to provide:
 - > Operators on lattices
 - Partial ordering
 - Upper and lower bound
 - Top and bottom elements
 - > Abstraction and concretization functions
 - > *assign*
 - > *testTrue*

Implementation

```
public interface State {  
    public boolean lessEqual(State val);  
    public State lub(State val1, State val2);  
    public State glb(State val1, State val2);  
    public State widening(State val1, State val2);  
  
    public void assign(Variable var, Expression expr);  
  
    public void testTrue(ConditionalExpression condition);  
}
```

The final solution

- We can trace all the constraints!
- More generic than non-relational domains
- Relational domains solve all problems of static analysis of numerical information!!!

No!!!

Approximation

- $x > y$: linear complexity
 - > If we do not refine too much the analysis
- $\pm x \pm y < k$: cubic complexity
 - > Not so fast, but it is OK!
- $\sum a_i * x_i \geq k$: exponential complexity!
- And what about non-linear relations?
 - > Logarithmic?
 - > Exponential?...
- We need focused relational domains!

General Idea of Abstraction

- **Something uncomputable/too complex:**
 - > Approximation/Abstraction
 - > Computable!
- **Problem:**
 - > False alarms
 - The abstraction does not validate the property
 - The property is always validated at runtime
- **Abstract domains focused on a property**
- **Different relational domains!**

Representation and Implementation

- Many different ways of representing and implementing a relational domain:
 - > Graphs
 - > Matrixes
 - > Functions
 - > ...
- Cartesian plan
 - > Graphic representation of the domain
 - > Intuition

Intervals

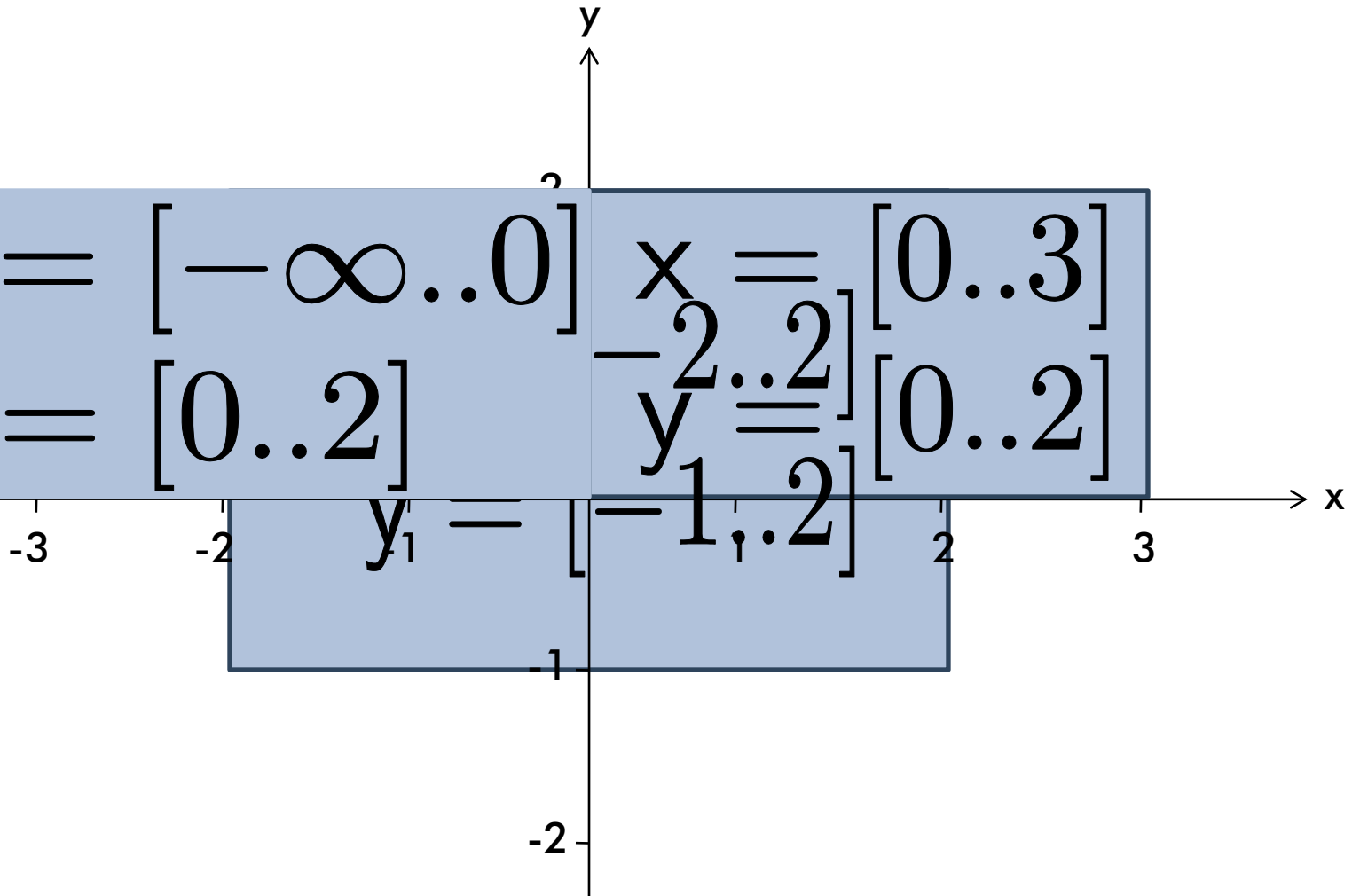
$$x = [-\infty..0]$$

$$y = [0..2]$$

$$x = [0..3]$$

$$y = [-2..2]$$

$$y_1 = [-1..2]$$



Closure

- **Problem:**
 - > Constraints between variables
 - > A set of constraints may contain other implicit constraints
 - e.g. transitive property of $=$, $>$, ...
- **E.g.**
$$\begin{array}{c} x > y, y > z \\ \Downarrow \\ x > y, y > z, x > z \end{array}$$
- **An iteration is not enough!**

Fixpoint

- E.g.

$$x > y, y > z, z == w$$



$$x > y, y > z, z == w, x > z, y > w$$



$$x > y, y > z, z == w, x > z, y > w, x > w$$

- We need to compute... a fixpoint!!!
- At each iteration something more refined
> i.e. less or equal w.r.t. the partial order

Formal definition

$$\begin{aligned} \textit{closure} &: [\Sigma \rightarrow \Sigma] \\ \textit{closure}(\sigma) &= \textit{lfp } \sigma_1. \rho(\sigma_1) \end{aligned}$$

$$\begin{aligned} \rho &: [\Sigma \rightarrow \Sigma] \\ \rho(\sigma) &\leq \sigma \end{aligned}$$

Concretization function

- Relational domains = set of constraints
- Concretization:
 - > all the states that respect the constraints

$$\gamma(\{c_1, c_2, c_3, \dots, c_n\}) = \{\sigma : \forall i \in [1..n] : eval_cond(c_i, \sigma) = true\}$$

- E.g. $\gamma(\{x > y, y > z\}) =$
 - $[x \mapsto 10, y \mapsto 9, z \mapsto 8]$
 - $[x \mapsto 10, y \mapsto 5, z \mapsto 2]$
 - $[x \mapsto 10, y \mapsto 5, z \mapsto 2, w \mapsto -1]$
 - ...

Lattices operators

- Think about the concretization
- Partial order:
 - > \supseteq on constraints
- Upper bound
 - > \cap on constraints
- Lower bound
 - > \cup on constraints
- Easy!
- Sure?

Implementation

- Different constraints may be ordered

$$\{x - y \geq 1\} \leq \{x - y \geq 0\}$$

- Constraints represented by

- > graphs

- > matrixes

- > functions

- > ...

- The implementation is not easy!
- Performances are critical