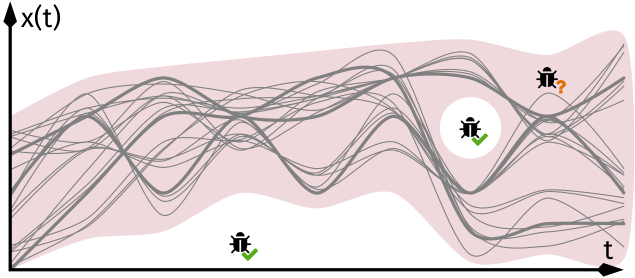


Abstract Interpretation

Software Quality Assurance — Static Code Analysis, II | Florian Sihler | December 10, 2025

Software Engineering
Programming Languages

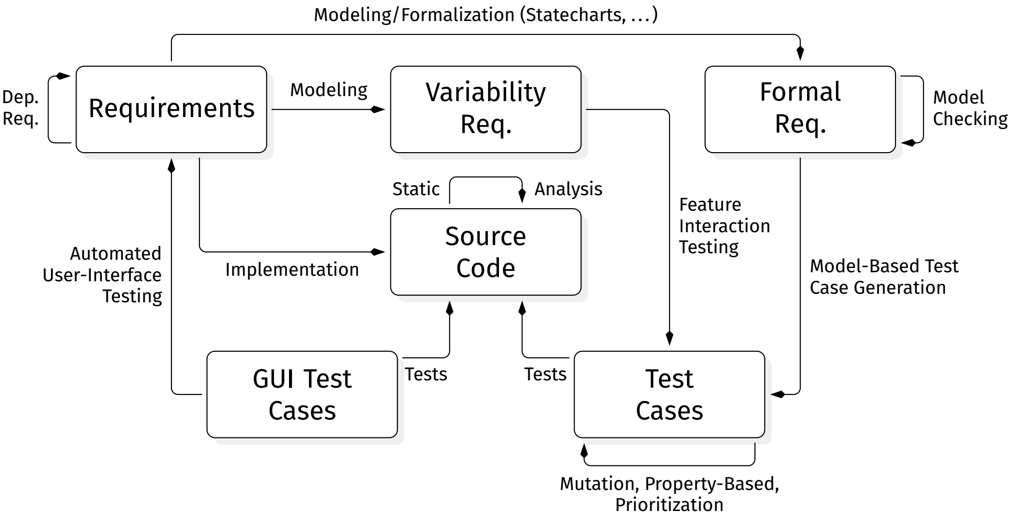
universität ulm

1. The Why

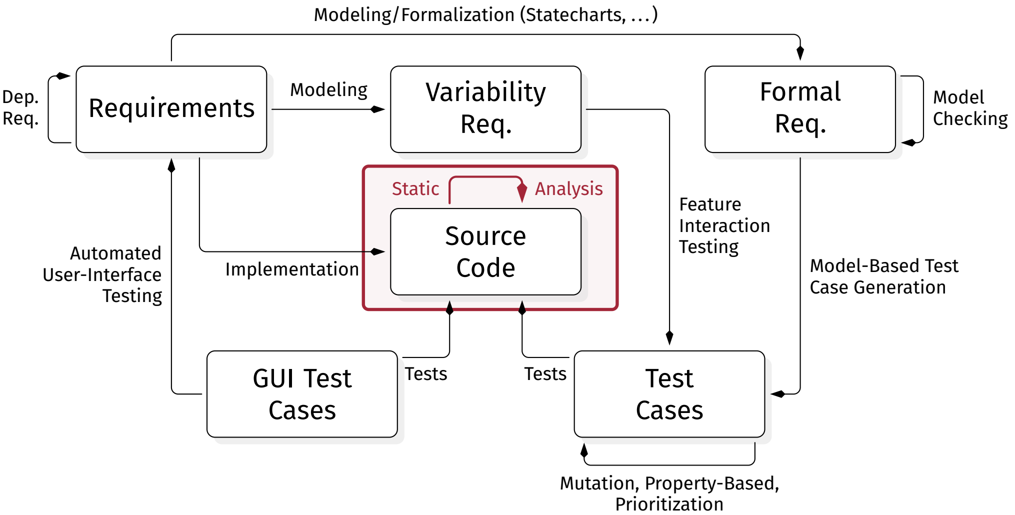
Why am I even here?

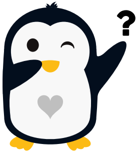
Embedding a Landscape

Embedding a Landscape

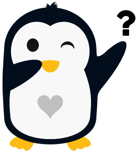


Embedding a Landscape



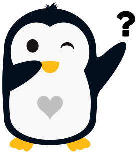


What is static analysis?



What is static analysis?

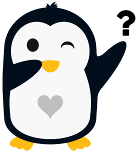
Discover *syntactic/semantic properties* of programs **without** running them.[RY20]



What is static analysis?

Discover *syntactic/semantic properties* of programs **without** running them.[RY20]

Today, we learn how to…

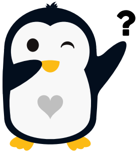


What is static analysis?

Discover *syntactic/semantic properties* of programs **without** running them.[RY20]

Today, we learn how to...

- describe semantic properties

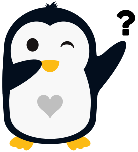


What is static analysis?

Discover *syntactic/semantic properties* of programs
without running them.[RY20]

Today, we learn how to…

- describe semantic properties
- compare, refine, and combine properties

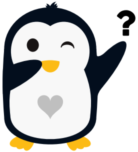


What is static analysis?

Discover *syntactic/semantic properties* of programs **without** running them.[RY20]

Today, we learn how to…

- describe semantic properties
- compare, refine, and combine properties
- describe and map language semantics to properties



What is static analysis?

Discover *syntactic/semantic properties* of programs **without** running them.[RY20]

Today, we learn how to...

- describe semantic properties
- compare, refine, and combine properties
- describe and map language semantics to properties
- deal with the cost of abstraction (and the fun)

The Why

```java
public static void main(String[] args) {
    int a = 1;
    double r = Math.random() * 10;
    if (r > 5) {
        a = 2;
    }
    System.out.println(1 / a);
}
```

The Why

java

```java
public static void main(String[] args) {
    int a = 1;
    double r = Math.random() * 10;
    if (r > 5) {
      a = 2;
    }
    System.out.println(1 / a);
}
```

The Why

```java
public static void main(String[] args) {
    int a = 1;                              ⦃ a ∈ {1} ⦄
    double r = Math.random() * 10;
    if (r > 5) {
      a = 2;
    }
    System.out.println(1 / a);
}
```

java

The Why

```java
public static void main(String[] args) {
    int a = 1;                              ⦃ a ∈ {1} ⦄
    double r = Math.random() * 10;          ⦃ r ∈ [0..10) ⦄
    if (r > 5) {
      a = 2;
    }
    System.out.println(1 / a);
}
```

java

The Why

```java
public static void main(String[] args) {
    int a = 1;                                  ⦃ a ∈ {1} ⦄
    double r = Math.random() * 10;              ⦃ r ∈ [0 .. 10) ⦄
    if (r > 5) {
      a = 2;                                    ⦃ a ∈ {2} ⦄
    }
    System.out.println(1 / a);
}
```

The Why

java

```java
public static void main(String[] args) {
    int a = 1;                            ⦃ a ∈ {1} ⦄
    double r = Math.random() * 10;        ⦃ r ∈ [0..10) ⦄
    if (r > 5) {
        a = 2;                            ⦃ a ∈ {2} ⦄
    }
    System.out.println(1 / a);            ⦃ a ∈ {1,2} ⦄
}
```

The Why

java

```java
public static void main(String[] args) {
    int a = 1;
    double r = Math.random() * 10;
    if (r > 5) {
      a = 2;
    }
    System.out.println(1 / a);
}
```

$\wr\, a \in \{1\} \,\int$

$\wr\, r \in [0\,..\,10) \,\int$

$\wr\, a \in \{2\} \,\int$

$\wr\, a \in \{1,2\} \,\int$ $\longrightarrow$ Valid? Ok? Safe?

The Why

```java
public static void main(String[] args) {
    int a = 1;
    double r = Math.random() * 10;
    if (r > 5) {
      a = 2;
    }
    System.out.println(1 / a);
}
```

java

$\wr\, a \in \{1\} \,\int$

$\wr\, r \in [0 .. 10) \,\int$

$\wr\, a \in \{2\} \,\int$

$\wr\, a \in \{1, 2\} \,\int \longrightarrow$ Valid? Ok? Safe?

- We want to proof, that a program satisfies certain properties

The Why

```java
public static void main(String[] args) {
    int a = 1;                              ⦃ a ∈ {1} ⦄
    double r = Math.random() * 10;          ⦃ r ∈ [0..10) ⦄
    if (r > 5) {
      a = 2;                                ⦃ a ∈ {2} ⦄
    }
    System.out.println(1 / a);              ⦃ a ∈ {1,2} ⦄  → Valid? Ok? Safe?
}
```

java

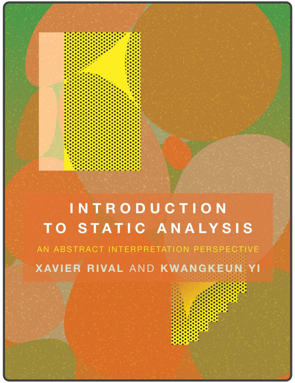
- We want to proof, that a program satisfies certain properties
- Abstract Interpretation is one (/the) technique to do so

Recommended Resources

And for an overview: “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17]

Recommended Resources

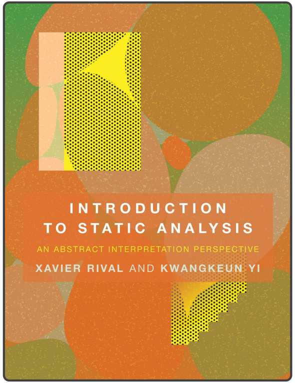
Using Analyses [RY20]



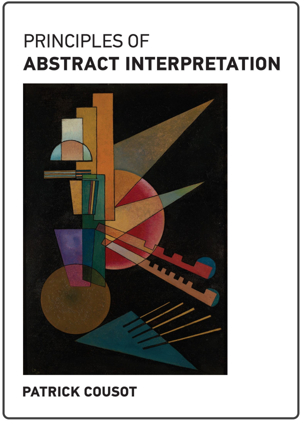
And for an overview: “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17]

Recommended Resources

Using Analyses [RY20]



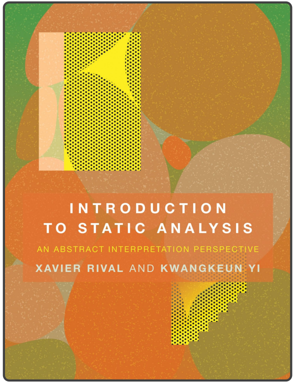
Formal Foundations [Cou21]



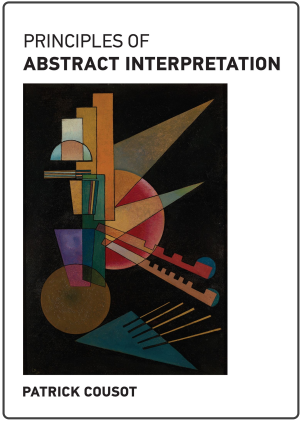
And for an overview: "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17]

Recommended Resources

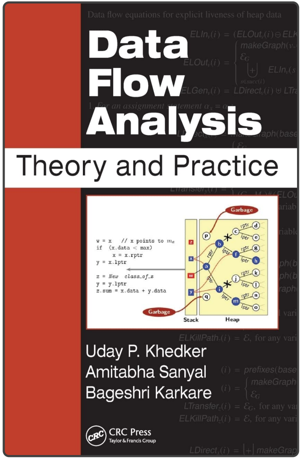
Using Analyses [RY20]



Formal Foundations [Cou21]



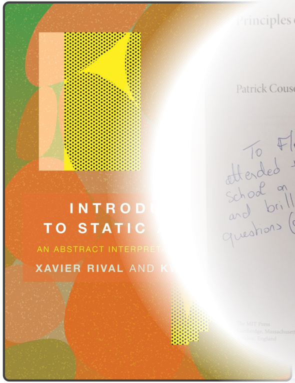
Dataflow Perspective [KSK09]



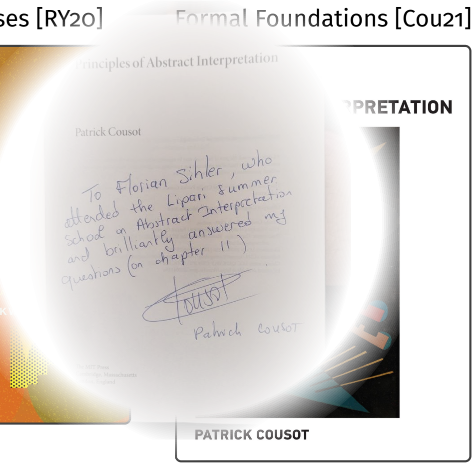
And for an overview: “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17]

Recommended Resources

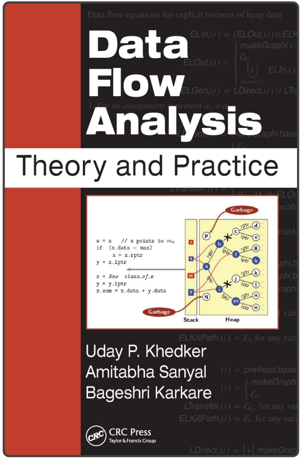
Using Analyses [RY20]



Formal Foundations [Cou21]



Dataflow Perspective [KSK09]



And for an overview: "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17]

Recommended Resources

Using Analyses [RY20] Formal Foundations [Cou21] Dataflow Perspective [KSK09]

And for an overview: “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17]

2. The How

Gimme properties, gimme abstractions!

Abstract Interpretation

See “A casual introduction to Abstract Interpretation” [Cou12]

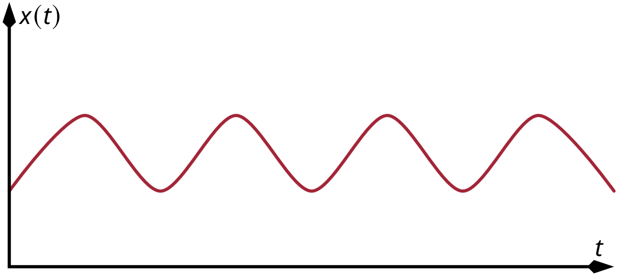
Concrete Interpretation

See “A casual introduction to Abstract Interpretation” [Cou12]

Concrete Interpretation

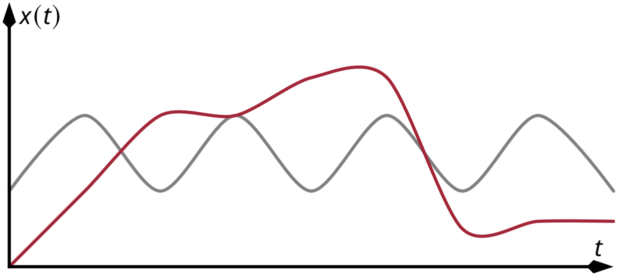
See "A casual introduction to Abstract Interpretation" [Cou12]

Concrete Interpretation



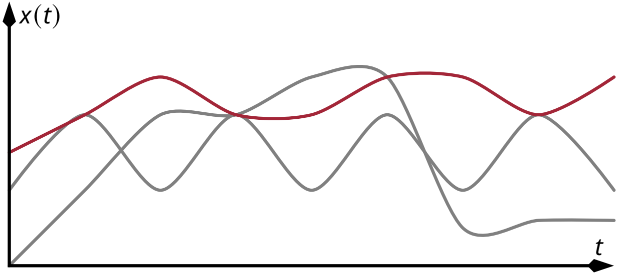
See "A casual introduction to Abstract Interpretation" [Cou12]

Concrete Interpretation



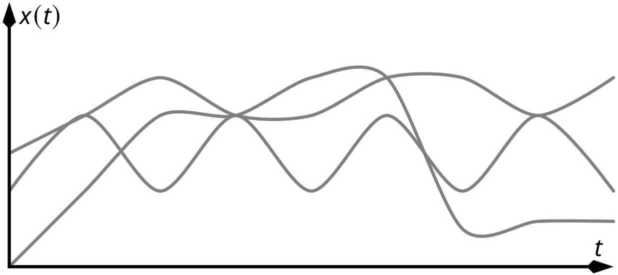
See “A casual introduction to Abstract Interpretation” [Cou12]

Concrete Interpretation



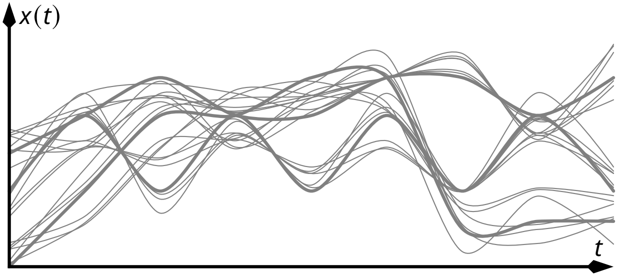
See “A casual introduction to Abstract Interpretation” [Cou12]

Concrete Interpretation



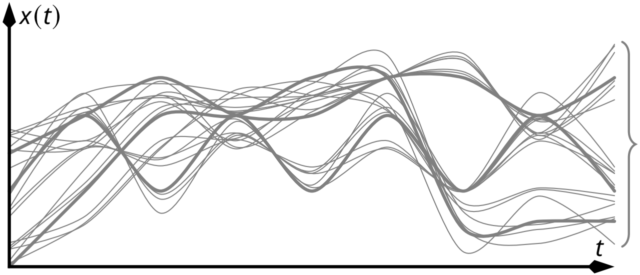
See “A casual introduction to Abstract Interpretation” [Cou12]

Concrete Interpretation



See “A casual introduction to Abstract Interpretation” [Cou12]

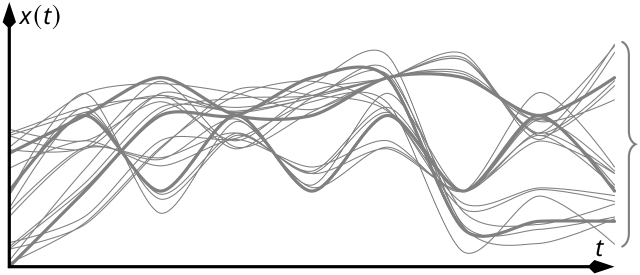
Concrete Interpretation



Collecting Semantics[Cou21, p. 91]

See "A casual introduction to Abstract Interpretation" [Cou12]

Concrete Interpretation

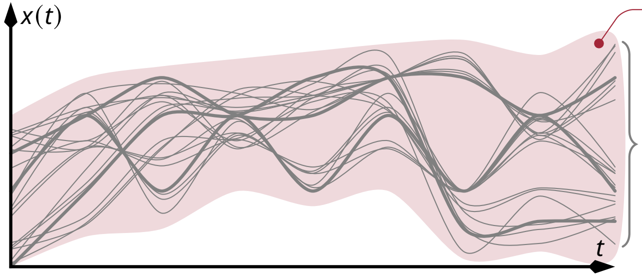


Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, …)
- … or very expensive (▸ *dynamic*)
- ▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

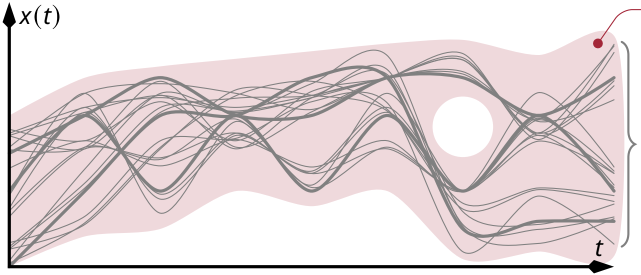
Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, …)
- … or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

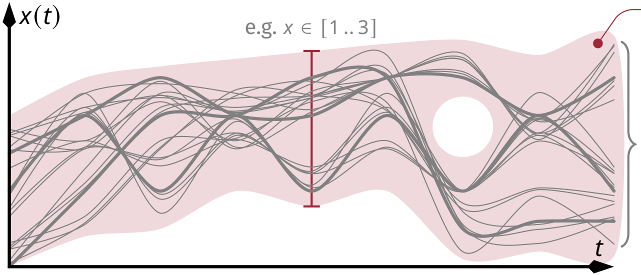
Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, …)
- … or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



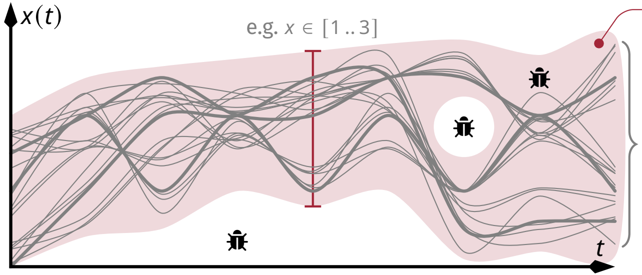
(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, ...)
- ... or very expensive (▸ *dynamic*)
- ▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

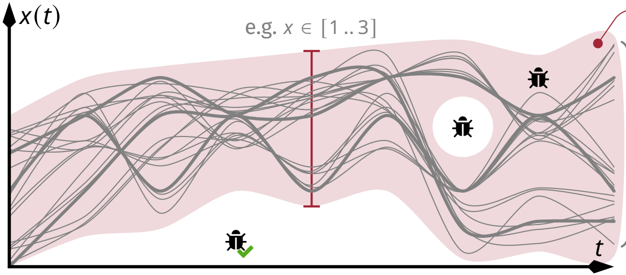
Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, ...)
- ... or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

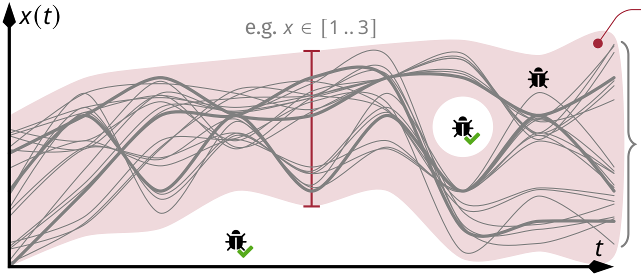
Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, …)
- … or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

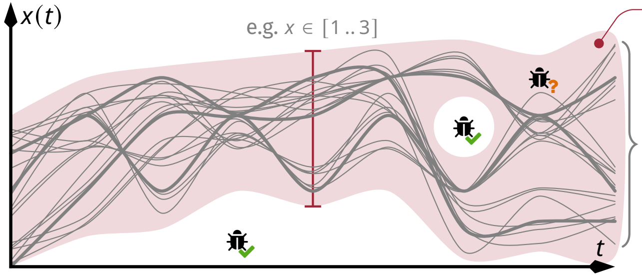
Collecting Semantics[Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, ...)
- ... or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation



(Trace) Abstraction[Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

Collecting Semantics[Cou21, p. 91]

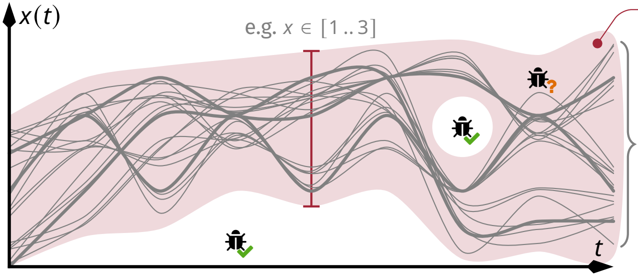
- Maybe impossible to compute statically
- Each trace is a single execution (test, …)
- … or very expensive (▸ *dynamic*)
- ▸ Abstract Interpretation to the rescue

See "A casual introduction to Abstract Interpretation" [Cou12]

Abstract Interpretation

Radhia Cousot (1947–2014)
Patrick Cousot

Patrick Cousot (1948)
Personal Website



(Trace) Abstraction [Cou21, p. 92]
just one of many
this **must** include all concrete traces!
(over-approximate)

Collecting Semantics [Cou21, p. 91]

- Maybe impossible to compute statically
- Each trace is a single execution (test, ...)
- ... or very expensive (▸ *dynamic*)

▸ Abstract Interpretation to the rescue

See “A casual introduction to Abstract Interpretation” [Cou12]

Terminology

- **Property**

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$
  universe ($\mathbb{U}$)

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $P = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \dots \} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\emptyset \subseteq P_1 \subseteq P_2 \subseteq \mathbb{U}$$

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$

universe ($\mathbb{U}$)

$$\emptyset \subseteq \mathsf{P}_1 \subseteq \mathsf{P}_2 \subseteq \mathbb{U}$$

strongest

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$

  universe ($\mathbb{U}$)

$$\underbrace{\varnothing}_{\text{strongest}} \subseteq \underbrace{\mathrm{P}_1}_{\text{stronger}} \subseteq \mathrm{P}_2 \subseteq \mathbb{U}$$

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $P = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\underbrace{\varnothing}_{\text{strongest}} \subseteq \underbrace{P_1}_{\text{stronger}} \subseteq \underbrace{P_2}_{\text{weaker}} \subseteq \mathbb{U}$$

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{\, z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \,\} = \{\, 0, 2, -2, 4, -4, 6, \ldots \,\} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\underbrace{\varnothing}_{\text{strongest}} \subseteq \underbrace{\mathsf{P}_1}_{\text{stronger}} \subseteq \underbrace{\mathsf{P}_2}_{\text{weaker}} \subseteq \underbrace{\mathbb{U}}_{\text{weakest}}$$

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\underbrace{\varnothing}_{\text{strongest}} \subseteq \underbrace{\mathrm{P}_1}_{\text{stronger}} \subseteq \underbrace{\mathrm{P}_2}_{\text{weaker}} \subseteq \underbrace{\mathbb{U}}_{\text{weakest}}$$

- **Partial Order**

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property —** Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{\, z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \,\} = \{\, 0, 2, -2, 4, -4, 6, \ldots \,\} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\underbrace{\varnothing}_{\text{strongest}} \subseteq \underbrace{\mathrm{P}_1}_{\text{stronger}} \subseteq \underbrace{\mathrm{P}_2}_{\text{weaker}} \subseteq \underbrace{\mathbb{U}}_{\text{weakest}}$$

- **Partial Order —** A reflexive, transitive, antisymmetric relation on a set (“poset”)

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{\, z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \,\} = \{\, 0, 2, -2, 4, -4, 6, \ldots \,\} \subseteq \mathcal{P}(\mathbb{Z})$ universe ($\mathbb{U}$)

$$\underbrace{\emptyset}_{\text{strongest}} \subseteq \underbrace{\mathsf{P}_1}_{\text{stronger}} \subseteq \underbrace{\mathsf{P}_2}_{\text{weaker}} \subseteq \underbrace{\mathbb{U}}_{\text{weakest}}$$

$\forall x \in X : x \sqsubseteq x$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set ("poset")

"Principles of Abstract Interpretation" [Cou21, p. 15],"Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$
  universe ($\mathbb{U}$)

$$\varnothing \subseteq \mathrm{P}_1 \subseteq \mathrm{P}_2 \subseteq \mathbb{U}$$

strongest · stronger · weaker · weakest

$\forall x, y, z \in X : x \sqsubseteq y \land y \sqsubseteq z \implies x \sqsubseteq z$

$\forall x \in X : x \sqsubseteq x$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set (“poset”)

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{\, z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \,\} = \{\, 0, 2, -2, 4, -4, 6, \ldots \,\} \subseteq \mathcal{P}(\mathbb{Z})$

  universe ($\mathbb{U}$)

  $$\emptyset \subseteq \mathsf{P}_1 \subseteq \mathsf{P}_2 \subseteq \mathbb{U}$$

  strongest
  stronger
  weaker
  weakest

  $\forall x, y, z \in X : x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$

  $\forall x \in X : x \sqsubseteq x$

  $\forall x, y \in X : x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set (“poset”)

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{\, z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \,\} = \{\, 0, 2, -2, 4, -4, 6, \ldots \,\} \subseteq \mathcal{P}(\mathbb{Z})$

  universe ($\mathbb{U}$)

  $$\varnothing \subseteq \mathsf{P}_1 \subseteq \mathsf{P}_2 \subseteq \mathbb{U}$$

  strongest stronger weaker weakest

  $\forall x, y, z \in X : x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$

  $\forall x \in X : x \sqsubseteq x$

  $\forall x, y \in X : x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set (“poset”)
  $(\mathbb{Z}, \leq)$

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathsf{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \dots \} \subseteq \mathcal{P}(\mathbb{Z})$

  universe ($\mathbb{U}$)

  $$\varnothing \subseteq \mathsf{P}_1 \subseteq \mathsf{P}_2 \subseteq \mathbb{U}$$

  strongest

  stronger

  weaker

  weakest

$\forall x, y, z \in X : x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$

$\forall x \in X : x \sqsubseteq x$

$\forall x, y \in X : x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set (“poset”)
  $(\mathbb{Z}, \leq), \quad (\mathcal{P}(\mathbb{Z}), \subseteq), \quad \dots$

“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

Terminology

- **Property** — Set of states/traces that satisfy that property
  Even integers: $\mathrm{P} = \{ z \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : z = 2k \} = \{ 0, 2, -2, 4, -4, 6, \ldots \} \subseteq \mathcal{P}(\mathbb{Z})$
  universe ($\mathbb{U}$)

$$\varnothing \subseteq \mathrm{P}_1 \subseteq \mathrm{P}_2 \subseteq \mathbb{U}$$

strongest — stronger — weaker — weakest

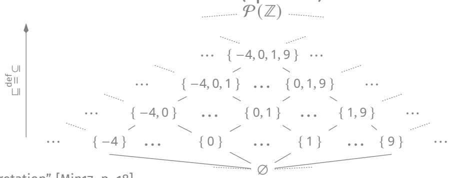
$\forall x, y, z \in X : x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$

$\forall x \in X : x \sqsubseteq x$

$\forall x, y \in X : x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$

- **Partial Order** — A reflexive, transitive, antisymmetric relation on a set (“poset”)
  $(\mathbb{Z}, \leq), \quad (\mathcal{P}(\mathbb{Z}), \subseteq), \quad \ldots$

Hasse Diagram of $(\mathcal{P}(\mathbb{Z}), \subseteq)$



“Principles of Abstract Interpretation” [Cou21, p. 15],“Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 18]

A Task In-Between

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

A Task In-Between

Helmut Hasse (1898–1979)
CC BY SA 2.0 Oberwolfach Photo Collection

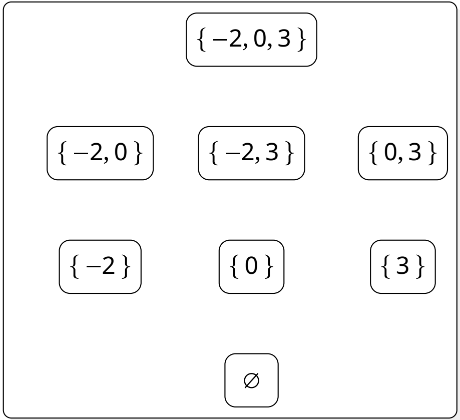
Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

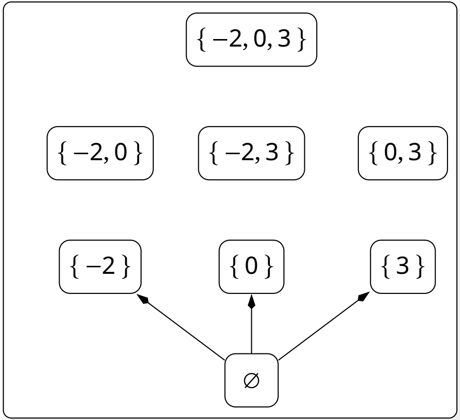


Hasse Diagram of $(\mathcal{P}(\{\mathbf{0}, \mathbf{3}, \mathbf{-2}\}), \subseteq)$

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

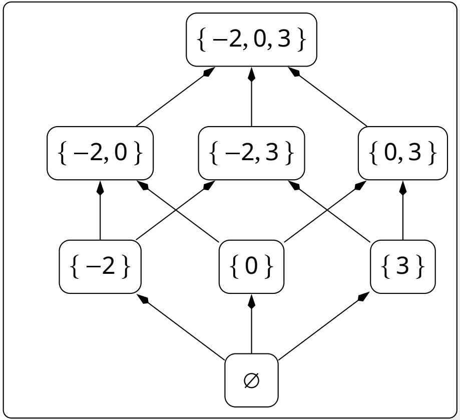


Hasse Diagram of $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

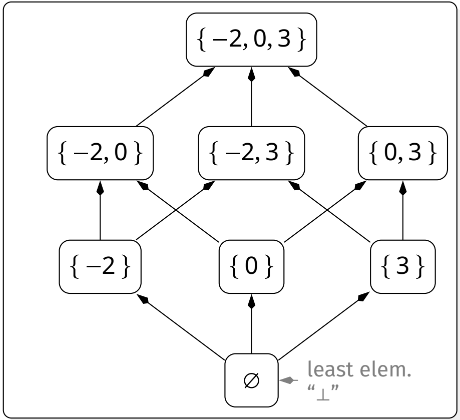


Hasse Diagram of $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$

A Task In-Between

Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.

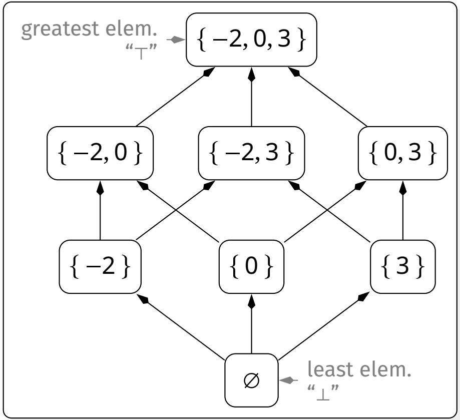


Hasse Diagram of $(\mathcal{P}(\{\mathbf{0}, \mathbf{3}, -\mathbf{2}\}), \subseteq)$

A Task In-Between

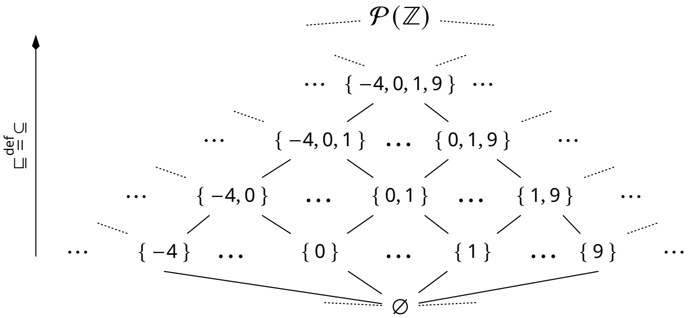
Task:
Consider the Poset $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$.
Draw its Hasse diagram.
Indicate the greatest and smallest element.

Hasse Diagram:
A directed graph, with edges from a to b ($a \neq b$) indicating that $a \sqsubseteq b$ without a c with $a \sqsubseteq c \sqsubseteq b$.



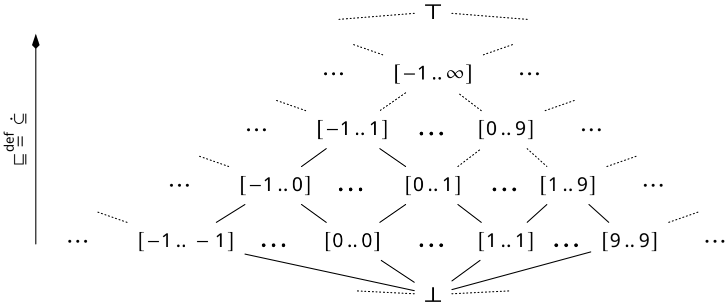
Hasse Diagram of $(\mathcal{P}(\{0, 3, -2\}), \subseteq)$

Posets, Lattices, and Chains



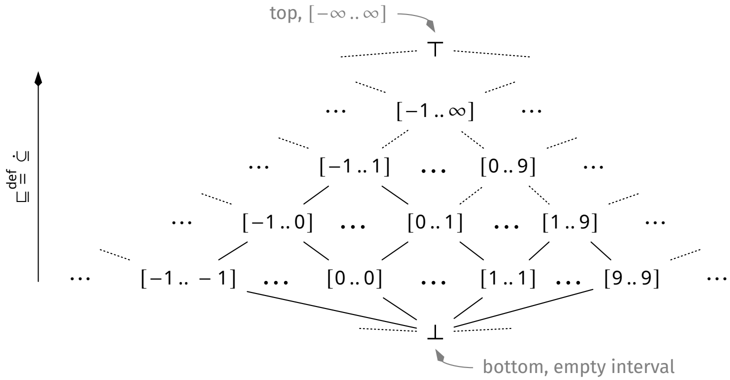
"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



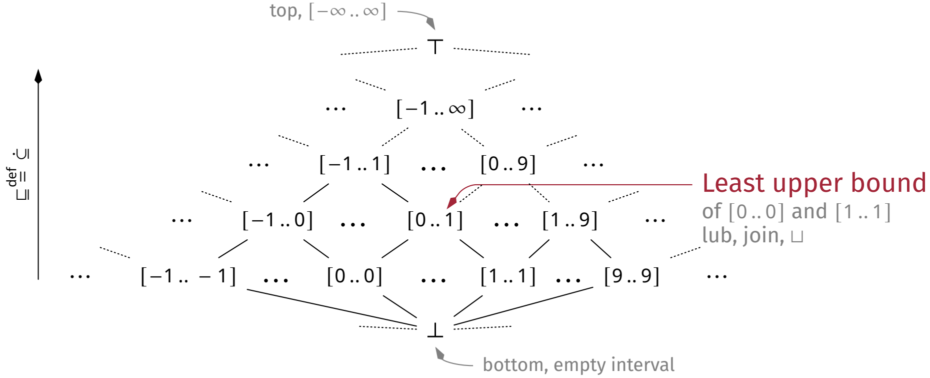
“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



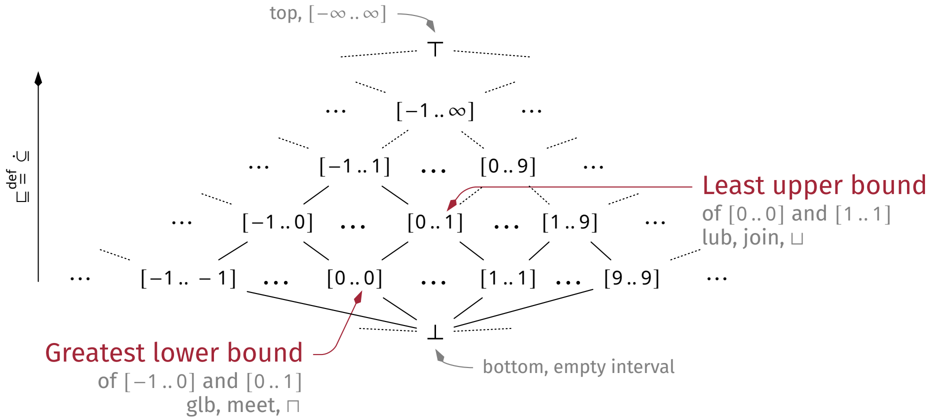
"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



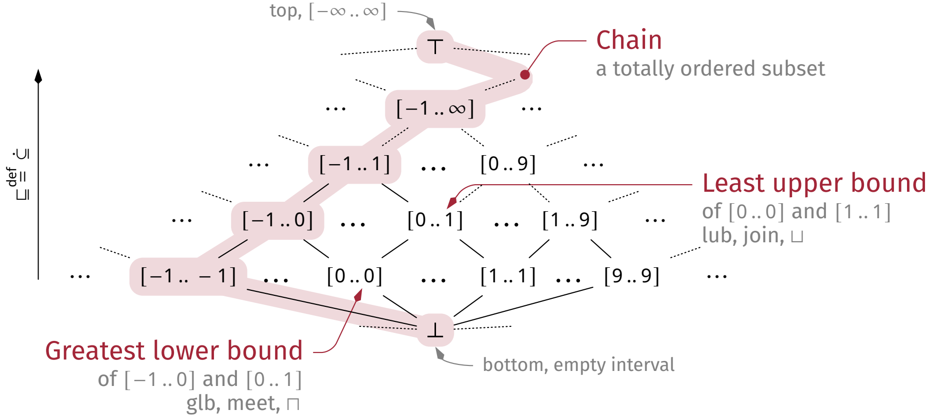
"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



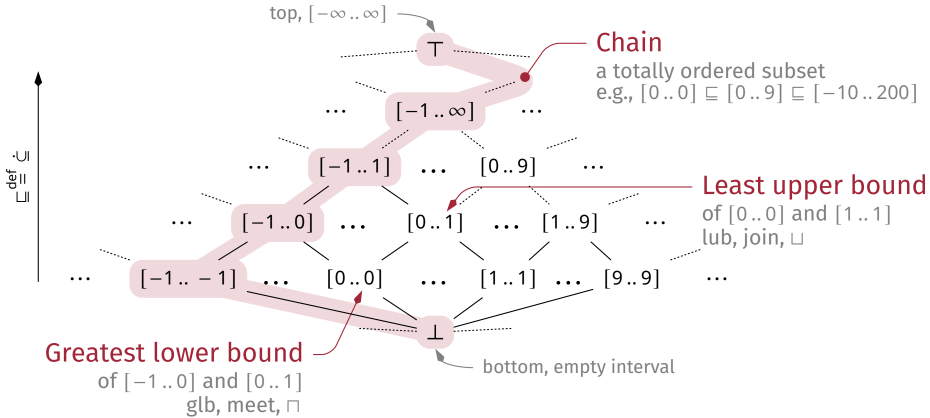
“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



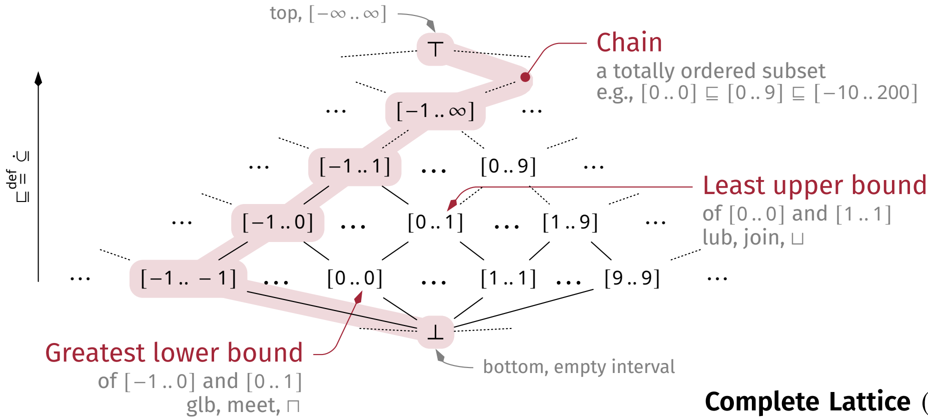
“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

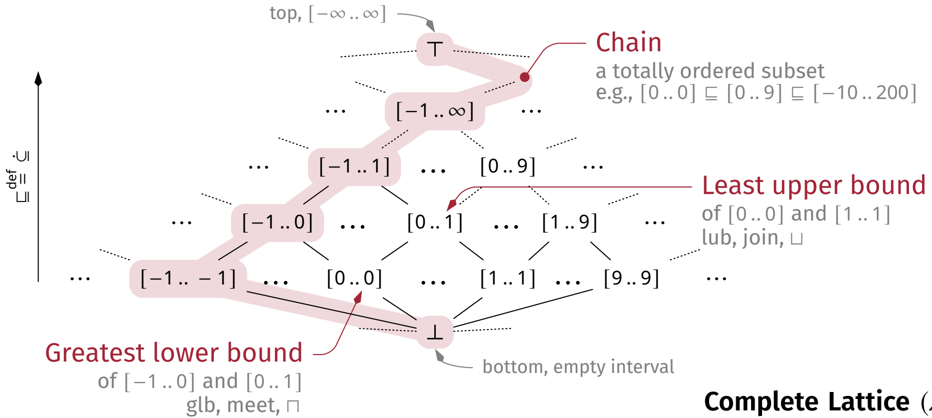
Posets, Lattices, and Chains



Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains

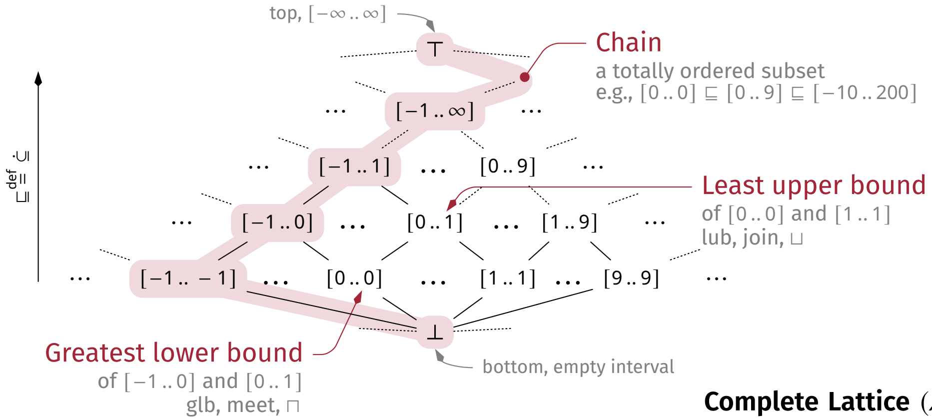


Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

- $(X, \sqsubseteq)$ is a partial order

“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains

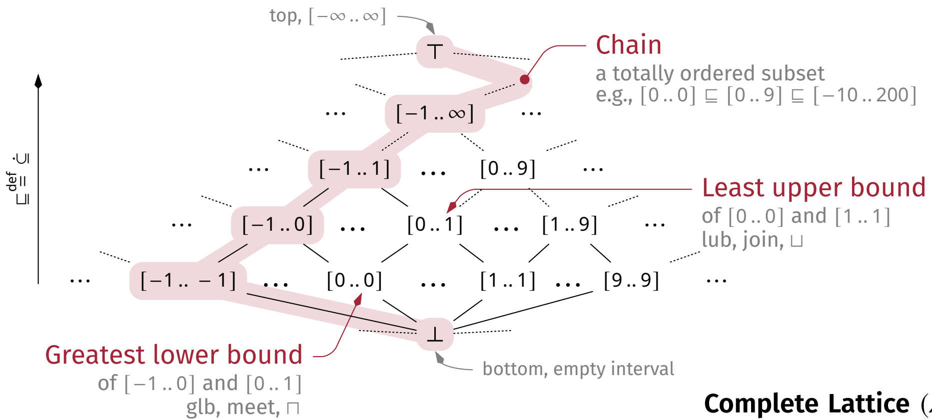


Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

- $(X, \sqsubseteq)$ is a partial order
- $\forall A \subseteq X : \sqcup A$ and $\sqcap A$ exist

“Lattice theory” [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains

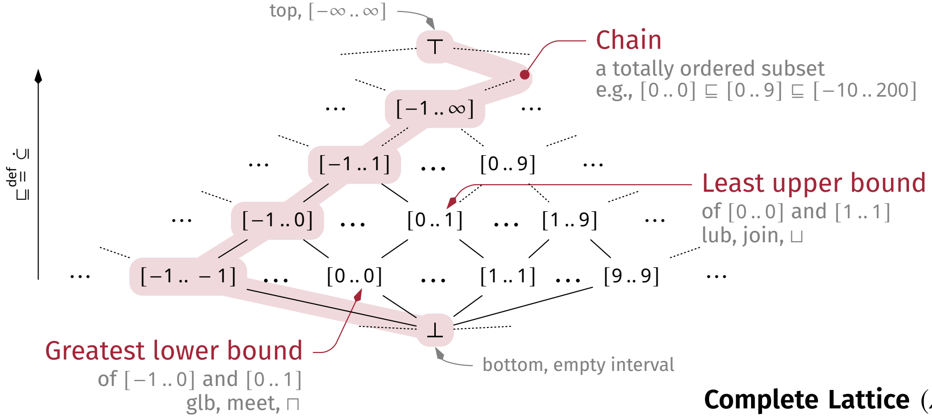


Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

- $(X, \sqsubseteq)$ is a partial order
- $\forall A \subseteq X : \sqcup A$ and $\sqcap A$ exist
- $\bot/\top$ as smallest/largest element

"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



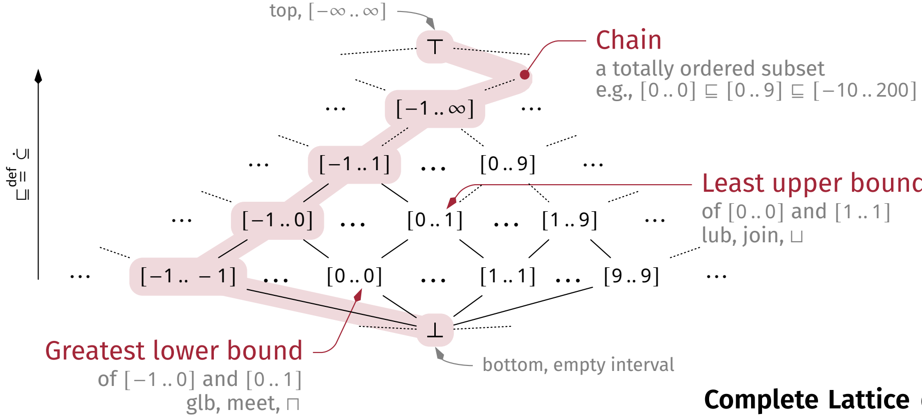
Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

- $(X, \sqsubseteq)$ is a partial order
- $\forall A \subseteq X : \sqcup A$ and $\sqcap A$ exist
- $\bot / \top$ as smallest/largest element

"Lattice elements (e.g. $[0\,..\,1]$) define — per variable — the abstract state (the abstraction) at a given time or program point!"

"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Posets, Lattices, and Chains



Garrett Birkhoff (1911–96)
Paul Halmos

"Lattice elements (e.g. $[0\,..\,1]$) define — per variable — the abstract state (the abstraction) at a given time or program point!"

Complete Lattice $(X, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

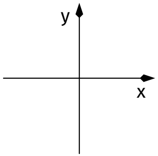
- $(X, \sqsubseteq)$ is a partial order
- $\forall A \subseteq X : \sqcup A$ and $\sqcap A$ exist
- $\bot / \top$ as smallest/largest element

"Lattice theory" [Bir67], see also sublattices [Min17, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

Numerical

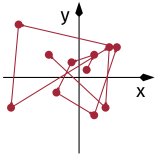


“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

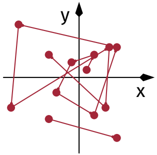
Numerical



“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

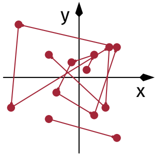
Pick Your Favorite Lattice!



"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

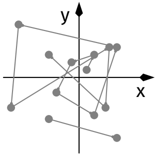


Collecting Semantics

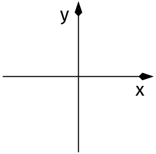
"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!



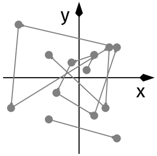
Collecting Semantics



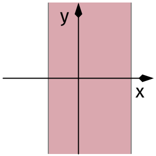
“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!



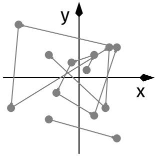
Collecting Semantics



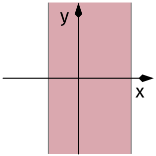
“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!



Collecting Semantics



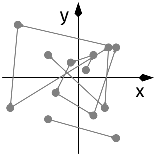
Intervals $x \in [a \,..\, b]$

$y \in [-\infty \,..\, \infty]$

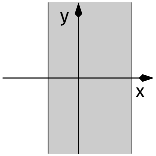
“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!



Collecting Semantics

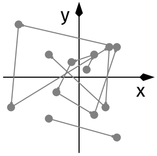


Intervals $x \in [a\,..\,b]$
$y \in [-\infty\,..\,\infty]$

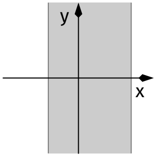
"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

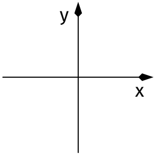


Collecting Semantics



Intervals $x \in [a \mathinner{..} b]$

$y \in [-\infty \mathinner{..} \infty]$

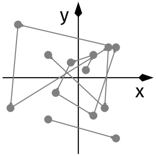


“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

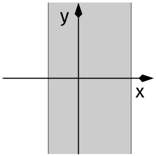
Abstract Domains

Pick Your Favorite Lattice!

Numerical

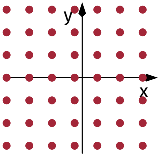


Collecting Semantics



Intervals $x \in [a\,..\,b]$

$y \in [-\infty\,..\,\infty]$

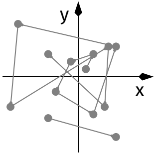


Simple Congruences

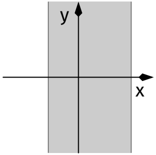
"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

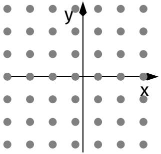


Collecting Semantics

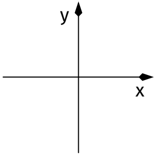


Intervals $x \in [a \,..\, b]$

$y \in [-\infty \,..\, \infty]$



Simple Congruences

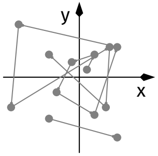


"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

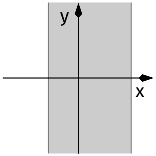
Abstract Domains

Pick Your Favorite Lattice!

Numerical

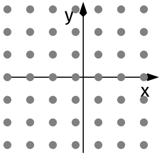


Collecting Semantics

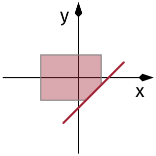


Intervals $x \in [a \mathinner{..} b]$

$y \in [-\infty \mathinner{..} \infty]$



Simple Congruences

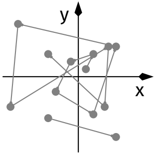


Pentagons

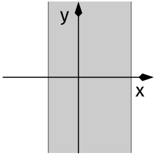
"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

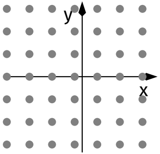
Pick Your Favorite Lattice!



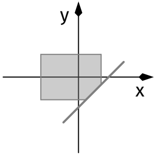
Collecting Semantics



Intervals $x \in [a \mathinner{..} b]$
$y \in [-\infty \mathinner{..} \infty]$



Simple Congruences

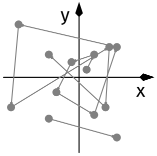


Pentagons

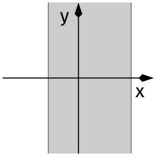
“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

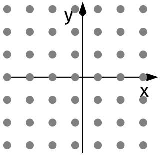
Pick Your Favorite Lattice!



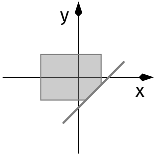
Collecting Semantics



Intervals $x \in [a \,..\, b]$
$y \in [-\infty \,..\, \infty]$



Simple Congruences



Pentagons

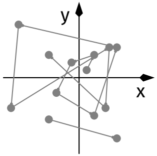
- Octagons

"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

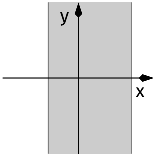
Abstract Domains

Pick Your Favorite Lattice!

Numerical

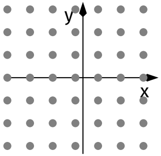


Collecting Semantics

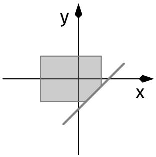


Intervals $x \in [a \mathinner{..} b]$

$y \in [-\infty \mathinner{..} \infty]$



Simple Congruences



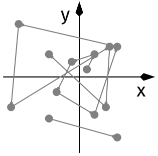
Pentagons

- Octagons
- Ellipses

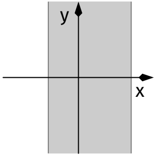
“Principles of Abstract Interpretation” [Cou21], “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses” [LF08, p. 25]

Abstract Domains

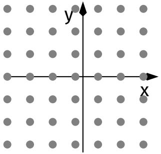
Pick Your Favorite Lattice!



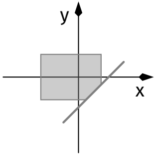
Collecting Semantics



Intervals $x \in [a \mathinner{..} b]$
$y \in [-\infty \mathinner{..} \infty]$



Simple Congruences



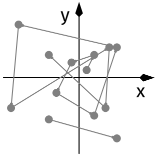
Pentagons

- Octagons
- Ellipses
- Exponentials

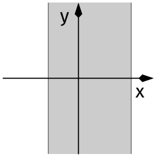
"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Abstract Domains

Pick Your Favorite Lattice!

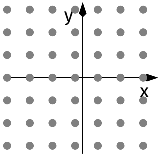


Collecting Semantics

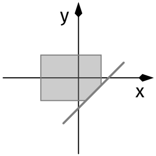


Intervals $x \in [a \,..\, b]$

$y \in [-\infty \,..\, \infty]$



Simple Congruences



Pentagons

- Octagons
- Ellipses
- Exponentials
- Signs

"Principles of Abstract Interpretation" [Cou21], "Pentagons: a weakly relational abstract domain for the efficient validation of array accesses" [LF08, p. 25]

Sign Analysis

Sign Analysis

Simple Sign Domain

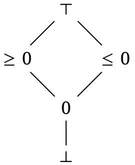
$$
\begin{tikzcd}[column sep=small]
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
\geq 0 \arrow[dr, dash] & & \leq 0 \arrow[dl, dash] \\
 & 0 \arrow[d, dash] & \\
 & \bot &
\end{tikzcd}
$$

Sign Analysis

Simple Sign Domain

- We still have no program semantics, but we can try…

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```



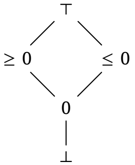
Sign Analysis

Simple Sign Domain

- We still have no program semantics, but we can try…

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

⦃ $a = 0$ ⦄



Simple Sign Domain

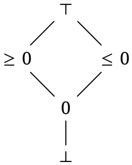
- We still have no program semantics, but we can try…

java

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

$\{ a = 0 \}$

$\{ b \geq 0 \}$



Sign Analysis

- We still have no program semantics, but we can try…

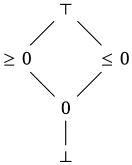
```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

java

$\{ a = 0 \}$

$\{ b \geq 0 \}$

$\{ c \geq 0 \quad (= 0\ +\ \geq 0) \}$



Sign Analysis

- We still have no program semantics, but we can try…

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

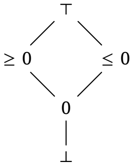
java

$\{\!\!\{\ a = 0\ \}\!\!\}$

$\{\!\!\{\ b \geq 0\ \}\!\!\}$

$\{\!\!\{\ c \geq 0 \quad (=0\ +\ \geq 0)\ \}\!\!\}$

$\{\!\!\{\ d = \top \quad (\geq 0\ -\ \geq 0)\ \}\!\!\}$



Sign Analysis

Simple Sign Domain

- We still have no program semantics, but we can try…

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

java

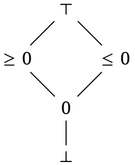
$\llparenthesis\, a = 0 \,\rrparenthesis$

$\llparenthesis\, b \geq 0 \,\rrparenthesis$

$\llparenthesis\, c \geq 0 \quad (= 0 \; + \; \geq 0) \,\rrparenthesis$

$\llparenthesis\, d = \top \quad (\geq 0 \; - \; \geq 0) \,\rrparenthesis$

- But, how do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?



Sign Analysis

Simple Sign Domain

- We still have no program semantics, but we can try...

java

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

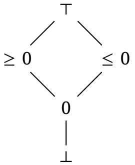
$\{\!\{\, a = 0 \,\}\!\}$

$\{\!\{\, b \geq 0 \,\}\!\}$

$\{\!\{\, c \geq 0 \quad (= 0 \;+\; \geq 0) \,\}\!\}$

$\{\!\{\, d = \top \quad (\geq 0 \;-\; \geq 0) \,\}\!\}$

- But, how do we know that `int a = 0;` implies $\{\!\{\, a = 0 \,\}\!\}$?
  - Language Semantics (we get to those later)



Sign Analysis

- We still have no program semantics, but we can try…

```java
int a = 0;
int b = 12;
int c = a + b;
int d = c - b;
```

java

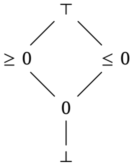
$\{\!\{\, a = 0 \,\}\!\}$

$\{\!\{\, b \geq 0 \,\}\!\}$

$\{\!\{\, c \geq 0 \quad (= 0 \;+\; \geq 0) \,\}\!\}$

$\{\!\{\, d = \top \quad (\geq 0 \;-\; \geq 0) \,\}\!\}$

- But, how do we know that `int a = 0;` implies $\{\!\{\, a = 0 \,\}\!\}$?
  - Language Semantics (we get to those later)
  - Galois Connections



Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\wr\, a = 0 \,\wr$?

A tale of two lattices…

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\wr\, a = 0 \,\wr$?

A tale of two lattices…

Concrete Domain
$(C, \sqsubseteq_C)$

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\wr\, a = 0 \,\wr$?

A tale of two lattices…

Concrete Domain
$(C, \sqsubseteq_C)$

This is the (usually infinite and unknown) ground truth, the reality

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\wr\, a = 0 \,\wr$?

A tale of two lattices…

Concrete Domain
$(C, \sqsubseteq_C)$

Abstract Domain
$(A, \sqsubseteq_A)$

This is the (usually infinite and unknown) ground truth, the reality

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?

A tale of two lattices…

Concrete Domain
$(C, \sqsubseteq_C)$

This is the (usually infinite and unknown) ground truth, the reality

Abstract Domain
$(A, \sqsubseteq_A)$

The abstract domain we choose, e.g., intervals

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llbracket a = 0 \rrbracket$?

A tale of two lattices…

α abstraction

Concrete Domain $(C, \sqsubseteq_C)$ $\xrightarrow{\alpha}$ **Abstract Domain** $(A, \sqsubseteq_A)$

This is the (usually infinite and unknown) ground truth, the reality

The abstract domain we choose, e.g., intervals

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?

A tale of two lattices…

$$\begin{tikzcd}[column sep=huge] \textbf{Concrete Domain } (C, \sqsubseteq_C) \arrow[r, bend left, "\alpha \text{ abstraction}"] & \textbf{Abstract Domain } (A, \sqsubseteq_A) \arrow[l, bend left, "\gamma \text{ concretization}"] \end{tikzcd}$$

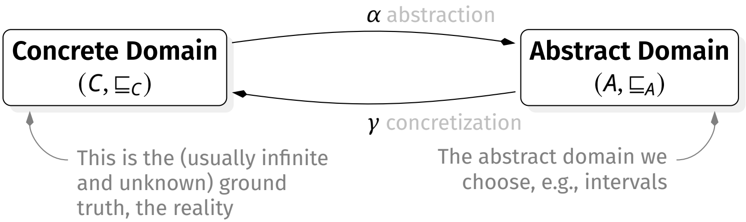
This is the (usually infinite and unknown) ground truth, the reality

The abstract domain we choose, e.g., intervals

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\wr\, a = 0 \,\wr$?

A tale of two lattices…



Beware, beware, conversions may be lossy!

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?

A tale of two lattices…

$$\begin{tikzcd}[column sep=huge] \boxed{\begin{array}{c}\textbf{Concrete Domain}\\ (C, \sqsubseteq_C)\end{array}} \arrow[r, bend left, "\alpha \text{ abstraction}"] & \boxed{\begin{array}{c}\textbf{Abstract Domain}\\ (A, \sqsubseteq_A)\end{array}} \arrow[l, bend left, "\gamma \text{ concretization}"] \end{tikzcd}$$

This is the (usually infinite and unknown) ground truth, the reality

The abstract domain we choose, e.g., intervals

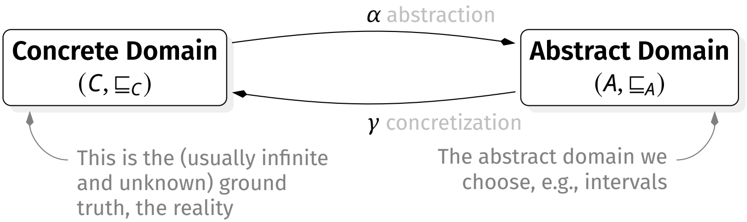
Beware, beware, conversions may be lossy!

- $\forall x \forall y : \alpha(x) \sqsubseteq_A y \iff x \sqsubseteq_C \gamma(y)$
- $\forall x \forall y : x \sqsubseteq_C \gamma(\alpha(x))$ and $\alpha(\gamma(y)) \sqsubseteq_A y$

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

How do we know that `int a = 0;` implies $\llparenthesis\, a = 0 \,\rrparenthesis$?

A tale of two lattices…



Beware, beware, conversions may be lossy!

- $\forall x \forall y : \alpha(x) \sqsubseteq_A y \iff x \sqsubseteq_C \gamma(y)$
- $\forall x \forall y : x \sqsubseteq_C \gamma(\alpha(x))$ and $\alpha(\gamma(y)) \sqsubseteq_A y$

"We can abstract $x \in \{1, 2\}$ with $\alpha(x) = [1 \mathinner{..} 2]$ or $\alpha(x) = [-5 \mathinner{..} 42]$, but *not* (e.g.) with $\alpha(x) = [1 \mathinner{..} 1]$! "

Galois Connection — Linking Posets [Cou21, p. 110] (small detour)

Évariste Galois (1811–32)
Public Domain
died in a duel for love

How do we know that `int a = 0;` implies $\llbracket a = 0 \rrbracket$?

A tale of two lattices…

α abstraction

Concrete Domain $(C, \sqsubseteq_C)$

Abstract Domain $(A, \sqsubseteq_A)$

γ concretization

This is the (usually infinite and unknown) ground truth, the reality

The abstract domain we choose, e.g., intervals

Beware, beware, conversions may be lossy!

- $\forall x \forall y : \alpha(x) \sqsubseteq_A y \iff x \sqsubseteq_C \gamma(y)$
- $\forall x \forall y : x \sqsubseteq_C \gamma(\alpha(x))$ and $\alpha(\gamma(y)) \sqsubseteq_A y$

“We can abstract $x \in \{1, 2\}$ with $\alpha(x) = [1 \mathinner{..} 2]$ or $\alpha(x) = [-5 \mathinner{..} 42]$, but *not* (e.g.) with $\alpha(x) = [1 \mathinner{..} 1]$! ”

Happy Abstractions

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if `u` is an unknown integer?
  - $\alpha($`u`$) =$
  - $\alpha($`u + 1`$) =$
  - $\alpha($`u * 0`$) =$
  - $\alpha($`sin(u)`$) =$

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\mathtt{u}) = [-\infty\,..\,\infty]$
  - $\alpha(\mathtt{u + 1}) =$
  - $\alpha(\mathtt{u * 0}) =$
  - $\alpha(\mathtt{sin(u)}) =$

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\texttt{u}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\texttt{u + 1}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\texttt{u * 0}) =$
  - $\alpha(\texttt{sin(u)}) =$

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\texttt{u}) = [-\infty \mathrel{..} \infty]$
  - $\alpha(\texttt{u + 1}) = [-\infty \mathrel{..} \infty]$
  - $\alpha(\texttt{u * 0}) = [0 \mathrel{..} 0]$
  - $\alpha(\texttt{sin(u)}) =$

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\mathtt{u}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\mathtt{u\ +\ 1}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\mathtt{u\ *\ 0}) = [0 \mathinner{..} 0]$
  - $\alpha(\mathtt{sin(u)}) = [-1 \mathinner{..} 1]$

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\texttt{u}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\texttt{u + 1}) = [-\infty \mathinner{..} \infty]$
  - $\alpha(\texttt{u * 0}) = [0 \mathinner{..} 0]$
  - $\alpha(\texttt{sin(u)}) = [-1 \mathinner{..} 1]$
- What is the best concretization of $[-5 \mathinner{..} 5]$? What do we loose?

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\mathtt{u}) = [-\infty\,..\,\infty]$
  - $\alpha(\mathtt{u\ +\ 1}) = [-\infty\,..\,\infty]$
  - $\alpha(\mathtt{u\ *\ 0}) = [0\,..\,0]$
  - $\alpha(\mathtt{sin(u)}) = [-1\,..\,1]$
- What is the best concretization of $[-5\,..\,5]$? What do we loose?
  $\gamma([-5\,..\,5]) = \{-5, -4, \ldots, 4, 5\}$. We loose…
  - Exact values and distribution information
  - Relational information (e.g., `x = u; y = x + 1;`)
  - Potential sequences (e.g., `for(i=0; i<n; i++) ...`)

Happy Abstractions

- Suppose, you use the interval domain, what are your best abstractions if u is an unknown integer?
  - $\alpha(\mathrm{u}) = [-\infty .. \infty]$
  - $\alpha(\texttt{u + 1}) = [-\infty .. \infty]$
  - $\alpha(\texttt{u * 0}) = [0 .. 0]$
  - $\alpha(\texttt{sin(u)}) = [-1 .. 1]$
- What is the best concretization of $[-5 .. 5]$? What do we loose?
  $\gamma([-5 .. 5]) = \{ -5, -4, \dots, 4, 5 \}$. We loose…
  - Exact values and distribution information
  - Relational information (e.g., `x = u; y = x + 1;`)
  - Potential sequences (e.g., `for(i=0; i<n; i++) ...`)
- But, how do we handle loops? Recursion?

Fixpoints

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$

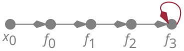
“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:

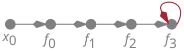
"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

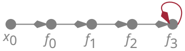
Fixpoints

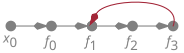
- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

    reach a fixpoint, $f^p = f(f^p)$

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

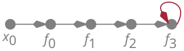
Fixpoints

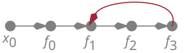
- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

 reach a fixpoint, $f^p = f(f^p)$
  - 

"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

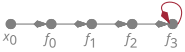
- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

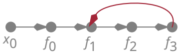
reach a fixpoint, $f^p = f(f^p)$
  - 

reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

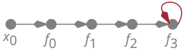
    reach a fixpoint, $f^p = f(f^p)$
  - 

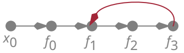
    reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

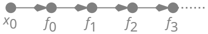
$f : \mathbb{N} \longrightarrow \mathbb{N}, f(x) = x + 1$

"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

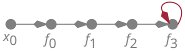
reach a fixpoint, $f^p = f(f^p)$
  - 

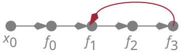
reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

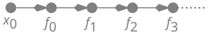
iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $\quad f : \mathbb{N} \longrightarrow \mathbb{N}, f(x) = x + 1$

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

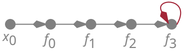
    reach a fixpoint, $f^p = f(f^p)$
  - 

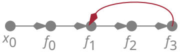
    reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

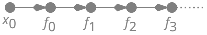
    iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $\quad f : \mathbb{N} \longrightarrow \mathbb{N}, f(x) = x + 1$
- If our function is monotonic, we can always find a fixpoint[Tar55]

“A lattice-theoretical fixpoint theorem and its applications.” [Tar55], “Introduction to metamathematics” [Kle52], “Principles of Abstract Interpretation” [Cou21, p. 165]

Fixpoints

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

reach a fixpoint, $f^p = f(f^p)$
  - 

reach a cycle, $f^{p+\ell} = f^p, \ell > 0$
  - 

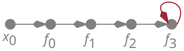
iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $f : \mathbb{N} \longrightarrow \mathbb{N}, f(x) = x + 1$
- If our function is monotonic, we can always find a fixpoint[Tar55]

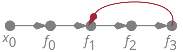
for complete, nonempty lattices
Tarski's Theorem

"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

Alfred Tarski (1901–83)
2.0 Oberwolfach Photo Collection

- For operators $f : X \to X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

    reach a fixpoint, $f^p = f(f^p)$
  - 

    reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

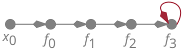
    iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $\quad f : \mathbb{N} \to \mathbb{N}, f(x) = x + 1$
- If our function is monotonic, we can always find a fixpoint[Tar55]

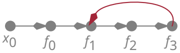
for complete, nonempty lattices
Tarski's Theorem

"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

Alfred Tarski (1901–83)
2.0 Oberwolfach Photo Collection

- For operators $f : X \longrightarrow X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

    reach a fixpoint, $f^p = f(f^p)$
  - 

    reach a cycle, $f^{p+\ell} = f^p$, $\ell > 0$
  - 

    iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$

$f : \mathbb{N} \longrightarrow \mathbb{N}, f(x) = x + 1$

- If our function is monotonic, we can always find a fixpoint[Tar55]

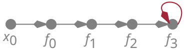
for complete, nonempty lattices
Tarski's Theorem

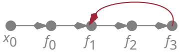
- Analyzing, e.g. loops, we "go up" the lattice until we reach a least fixpoint

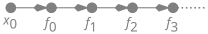
"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Fixpoints

Alfred Tarski (1901–83)
2.0 Oberwolfach Photo Collection

- For operators $f : X \to X$ a **fixpoint** is a $x \in X$ such that $f(x) = x$
- If we iterate f starting from some $x_0 \in X$:
  - 

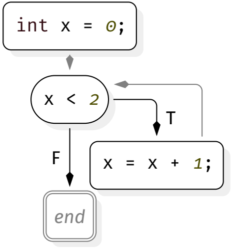
reach a fixpoint, $f^p = f(f^p)$
  - 

reach a cycle, $f^{p+\ell} = f^p, \ell > 0$
  - 

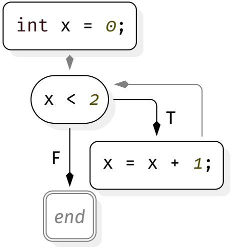
iterate forever, $\forall p \neq q \in \mathbb{N} : f^p \neq f^q$ $\quad f : \mathbb{N} \to \mathbb{N}, f(x) = x + 1$
- If our function is monotonic, we can always find a fixpoint[Tar55] for complete, nonempty lattices Tarski's Theorem
- Analyzing, e.g. loops, we "go up" the lattice until we reach a least fixpoint
- You may know fixpoints from tools like LaTeX or the λ-calculus

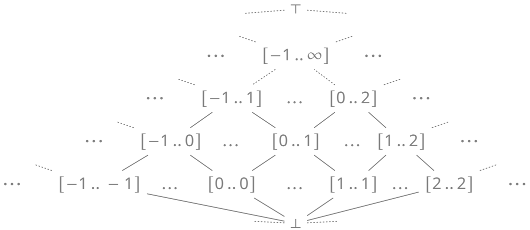
"A lattice-theoretical fixpoint theorem and its applications." [Tar55], "Introduction to metamathematics" [Kle52], "Principles of Abstract Interpretation" [Cou21, p. 165]

Interval Analysis, I (the intuitive approach)

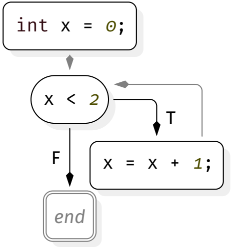


Interval Analysis, I (the intuitive approach)

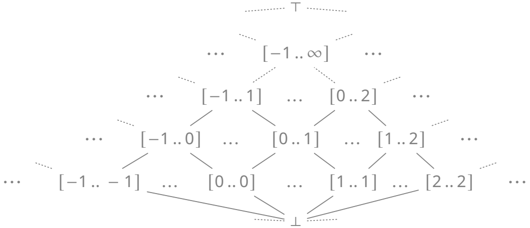




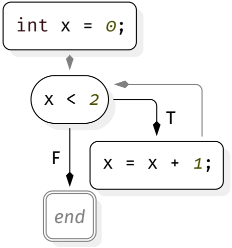
Interval Analysis, I (the intuitive approach)



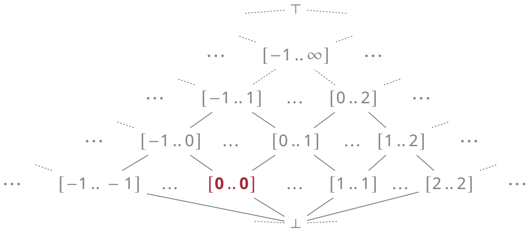
$\wr\, x_0 \in [0\,..\,0] \,\wr$



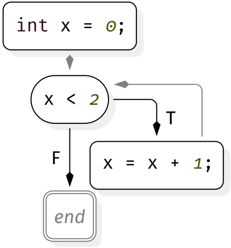
Interval Analysis, I (the intuitive approach)



$\wr\ x_0 \in [0\,..\,0]\ \wr$

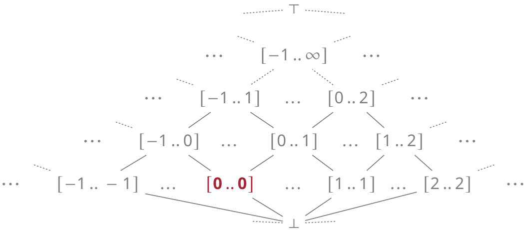


Interval Analysis, I (the intuitive approach)

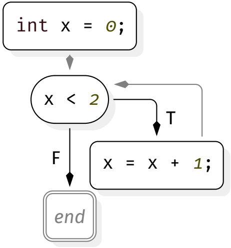


$$\wr\, x_0 \in [0\,..\,0]\, \int$$

$$\wr\, [\text{pre}]\; x_1 \in [0\,..\,0]\, \int$$



Interval Analysis, I (the intuitive approach)

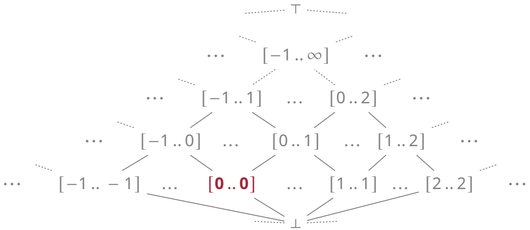


$\wr\, x_0 \in [0\,..\,0] \,\int$

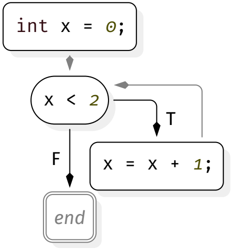
$\wr$ [pre] $x_1 \in [0\,..\,0] \,\int$

$\wr$ [in] $x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1]) \,\int$

if we are inside the loop we know that $x < 2$ holds!



Interval Analysis, I (the intuitive approach)



$\{\ x_0 \in [0\,..\,0]\ \}$

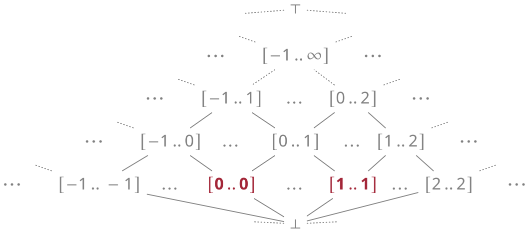
$\{\ [\text{pre}]\ x_1 \in [0\,..\,0]\ \}$

if we are inside the loop we know that $x < 2$ holds!

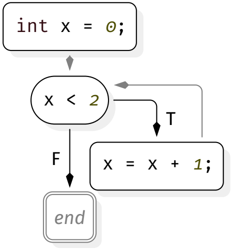
$\{\ [\text{in}]\ x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1])\ \}$

$\{\ x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1])\ \}$

we have to know how to add intervals here



Interval Analysis, I (the intuitive approach)



$\{\!\{\ x_0 \in [0\,..\,0]\ \}\!\}$

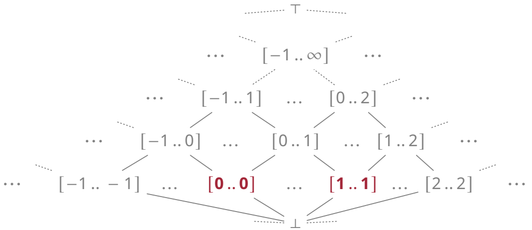
$\{\!\{$ [pre] $\boldsymbol{x_1 \in [0\,..\,0]}\ \}\!\}$

if we are inside the loop we know that $x < 2$ holds!

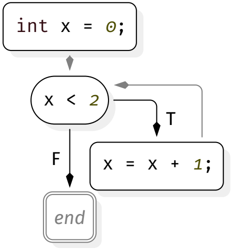
$\{\!\{$ [in] $x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1])\ \}\!\}$

$\{\!\{\ x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1])\ \}\!\}$

we have to know how to add intervals here



Interval Analysis, I (the intuitive approach)



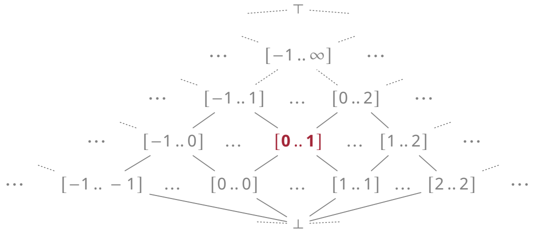
$\wr\; x_0 \in [0\,..\,0] \;\int$

we join on loops because, at this point x can have any of the values

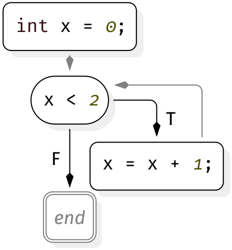
$\wr\;$ [pre] $\boldsymbol{x_1 \in [0\,..\,1]} \quad \boldsymbol{([0\,..\,0] \cup [1\,..\,1])} \;\int$

$\wr\;$ [in] $x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1]) \;\int$

$\wr\; x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1]) \;\int$



Interval Analysis, I (the intuitive approach)



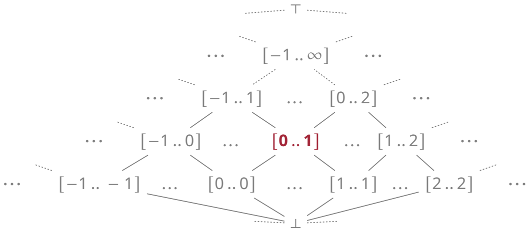
$\{ x_0 \in [0..0] \}$

we join on loops because, at this point x can have any of the values

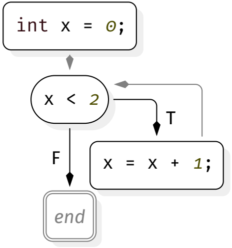
$\{ \text{[pre] } x_1 \in [0..1] \quad ([0..0] \cup [1..1]) \}$

$\{ \text{[in] } \boldsymbol{x_2 \in [0..1] \quad ([0..1] \cap (-\infty..1])} \}$

$\{ x_3 \in [1..1] \quad ([0..0] \oplus [1..1]) \}$



Interval Analysis, I (the intuitive approach)



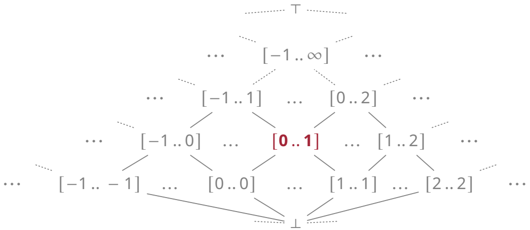
$\{ x_0 \in [0 \mathinner{..} 0] \}$

we join on loops because, at this point x can have any of the values

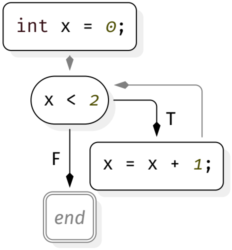
$\{ \text{[pre] } x_1 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 0] \cup [1 \mathinner{..} 1]) \}$

$\{ \text{[in] } x_2 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 1] \cap (-\infty \mathinner{..} 1]) \}$

$\boldsymbol{\{ x_3 \in [1 \mathinner{..} 2] \quad ([0 \mathinner{..} 1] \oplus [1 \mathinner{..} 1]) \}}$



Interval Analysis, I (the intuitive approach)

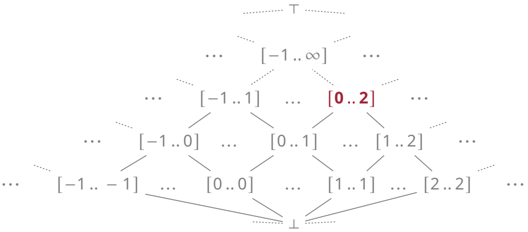


$$\{\!\{\ x_0 \in [0 \mathinner{..} 0]\ \}\!\}$$

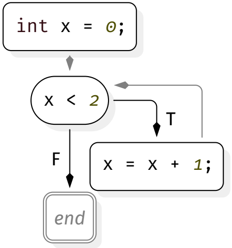
$$\{\!\{\ \text{[pre]}\ \boldsymbol{x_1 \in [0 \mathinner{..} 2]} \quad \mathbf{([0 \mathinner{..} 1] \cup [1 \mathinner{..} 2])}\ \}\!\}$$

$$\{\!\{\ \text{[in]}\ x_2 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 1] \cap (-\infty \mathinner{..} 1])\ \}\!\}$$

$$\{\!\{\ x_3 \in [1 \mathinner{..} 2] \quad ([0 \mathinner{..} 1] \oplus [1 \mathinner{..} 1])\ \}\!\}$$



Interval Analysis, I (the intuitive approach)



$$\{\!\mid x_0 \in [0 \mathinner{..} 0] \mid\!\}$$

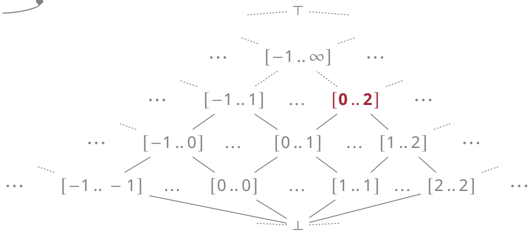
$$\{\!\mid \text{[pre]}\ x_1 \in [0 \mathinner{..} 2] \quad ([0 \mathinner{..} 1] \cup [1 \mathinner{..} 2]) \mid\!\}$$

$$\{\!\mid \text{[in]}\ x_2 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 1] \cap (-\infty \mathinner{..} 1]) \mid\!\}$$

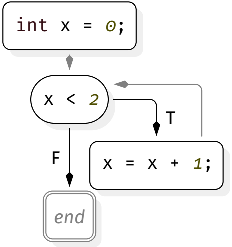
$$\{\!\mid x_3 \in [1 \mathinner{..} 2] \quad ([0 \mathinner{..} 1] \oplus [1 \mathinner{..} 1]) \mid\!\}$$

$$\{\!\mid \text{[post]}\ x_4 \in [2 \mathinner{..} 2] \quad ([0 \mathinner{..} 2] \cap [2 \mathinner{..} \infty)) \mid\!\}$$

after the loop we know that $\neg(x < 2) = x \geq 2$ holds!

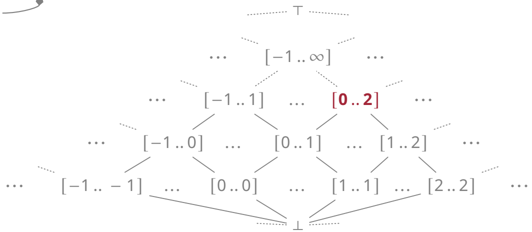


Interval Analysis, I (the intuitive approach)

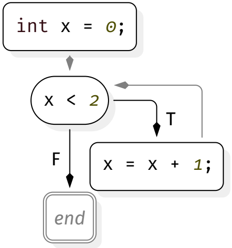


Intervals	Signs
$\{\!\{\ x_0 \in [0 \,..\, 0]\ \}\!\}$	$\{\!\{\ x_0 = 0\ \}\!\}$
$\{\!\{\ \text{[pre]}\ x_1 \in [0 \,..\, 2] \quad ([0 \,..\, 1] \cup [1 \,..\, 2])\ \}\!\}$	$\{\!\{\ \text{[pre]}\ x_1 \geq 0\ \}\!\}$
$\{\!\{\ \text{[in]}\ x_2 \in [0 \,..\, 1] \quad ([0 \,..\, 1] \cap (-\infty \,..\, 1])\ \}\!\}$	$\{\!\{\ \text{[in]}\ x_2 \geq 0\ \}\!\}$
$\{\!\{\ x_3 \in [1 \,..\, 2] \quad ([0 \,..\, 1] \oplus [1 \,..\, 1])\ \}\!\}$	$\{\!\{\ x_3 \geq 0\ \}\!\}$
$\{\!\{\ \text{[post]}\ x_4 \in [2 \,..\, 2] \quad ([0 \,..\, 2] \cap [2 \,..\, \infty))\ \}\!\}$	$\{\!\{\ \text{[post]}\ x_4 \geq 0\ \}\!\}$

after the loop we know that $\neg(x < 2) = x \geq 2$ holds!

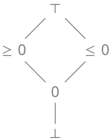


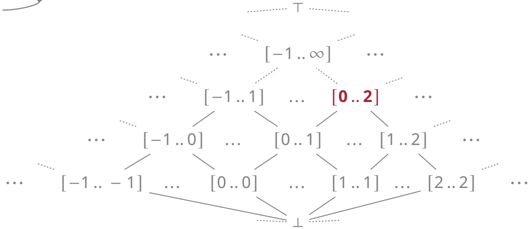
Interval Analysis, I (the intuitive approach)



Intervals	Signs
$\{\!\vert\; x_0 \in [0\,..\,0] \;\vert\!\}$	$\{\!\vert\; x_0 = 0 \;\vert\!\}$
$\{\!\vert\; \text{[pre]}\; x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2]) \;\vert\!\}$	$\{\!\vert\; \text{[pre]}\; x_1 \geq 0 \;\vert\!\}$
$\{\!\vert\; \text{[in]}\; x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1]) \;\vert\!\}$	$\{\!\vert\; \text{[in]}\; x_2 \geq 0 \;\vert\!\}$
$\{\!\vert\; x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1]) \;\vert\!\}$	$\{\!\vert\; x_3 \geq 0 \;\vert\!\}$
$\{\!\vert\; \text{[post]}\; x_4 \in [2\,..\,2] \quad ([0\,..\,2] \cap [2\,..\,\infty)) \;\vert\!\}$	$\{\!\vert\; \text{[post]}\; x_4 \geq 0 \;\vert\!\}$

after the loop we know that $\neg(x < 2) = x \geq 2$ holds!





3. Semantics

What does my program mean?

Semantics

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Semantics Program Syntax (simplified)

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$
Assignment

java

```java
int x = 0;

while(x < 2) {

  x = x + 1;

}
```

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Semantics   Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

```java
int x = 0;

while(x < 2) {

  x = x + 1;

}
```

Numeric Constant $c \in \mathbb{I}$

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Semantics Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

Sequence

Numeric Constant $c \in \mathbb{I}$

java

```java
int x = 0;

while(x < 2) {

   x = x + 1;

}
```

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

Sequence

Numeric Constant $c \in \mathbb{I}$

Loop

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

```java
int x = 0;
```

Sequence

Numeric Constant $c \in \mathbb{I}$

Loop

```java
while(x < 2) {
```

Comparison $\bowtie \in \{\leq, <, \ldots\}$

```java
  x = x + 1;

}
```

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

```java
int x = 0;
```

Sequence

Numeric Constant $c \in \mathbb{I}$

Loop

```java
while(x < 2) {
```

Comparison $\bowtie \in \{\leq, <, \ldots\}$

```java
  x = x + 1;
```

Binary Expression

```java
}
```

java

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Program Syntax (simplified)

Variable $v \in \mathbb{V}$

Assignment

```
int x = 0;
```

Sequence

Numeric Constant $c \in \mathbb{I}$

Loop

```
while(x < 2) {
```

Comparison $\bowtie \in \{\leq, <, \ldots\}$

```
   x = x + 1;
```

Binary Expression

```
}
```

$$
\begin{array}{llll}
\mathit{stm} & ::= & V \leftarrow \mathit{expr} & \text{(assignment, } V \in \mathbb{V}\text{)} \quad \text{java} \\
 & | & \mathit{stm}_1;\ \mathit{stm}_2 & \text{(sequence)} \\
 & | & \textbf{while}(\mathit{cond})\ \{\ \mathit{stm}\ \} & \text{(loop)} \\
\mathit{expr} & ::= & V & \text{(variable, } V \in \mathbb{V}\text{)} \\
 & | & c & \text{(constant, } c \in \mathbb{I}\text{)} \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & \text{(bin. expr., } \diamond \in \{+, -, \ldots\}\text{)} \\
\mathit{cond} & ::= & b & \text{(boolean, } b \in \mathbb{B}\text{)} \\
 & | & \mathit{expr}_1 \bowtie \mathit{expr}_2 & \text{(comparison, } \bowtie \in \{\leq, <, \ldots\}\text{)}
\end{array}
$$

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{llll}
\mathit{expr} & ::= & V & \text{(variable, } V \in \mathbb{V}\text{)} \\
 & | & c & \text{(constant, } c \in \mathbb{I}\text{)} \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & \text{(bin. expr., } \diamond \in \{+, -, \ldots\}\text{)}
\end{array}
$$

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{I}) \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{I}) \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

Variable

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```
java

$$\begin{array}{llll} expr & ::= & V & (\text{variable}, V \in \mathbb{V}) \\ & \mid & c & (\text{constant}, c \in \mathbb{I}) \\ & \mid & expr_1 \diamond expr_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\}) \end{array}$$

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \underbrace{\mathbb{V}}_{\text{Variable}} \to \underbrace{\mathbb{I}}_{\text{Integer Values}}$ to represent the current program state

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{Z}) \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \underbrace{\mathbb{V}}_{\text{Variable}} \to \underbrace{\mathbb{Z}}_{\text{Integer Values}}$ to represent the current program state

$\mathbb{V}$	$\mathbb{Z}$
x	0
c	5

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

expr	::=	V	(variable, $V \in \mathbb{V}$)
	|	c	(constant, $c \in \mathbb{I}$)
	|	$expr_1 \diamond expr_2$	(bin. expr., $\diamond \in \{+, -, \ldots\}$)

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \underbrace{\mathbb{V}}_{\text{Variable}} \rightarrow \underbrace{\mathbb{I}}_{\text{Integer Values}}$ to represent the current program state
- Now we can define evalExpr(*expr*, *env*) for an environment $env \in \mathcal{E}$

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{lcll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{I}) \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \underbrace{\mathbb{V}}_{\text{Variable}} \rightarrow \underbrace{\mathbb{I}}_{\text{Integer Values}}$ to represent the current program state
- Now we can define $\text{evalExpr}(\mathit{expr}, \mathit{env})$ for an environment $\mathit{env} \in \mathcal{E}$

Usually written as $\mathbb{E}[\![\, \mathit{expr} \,]\!]\rho$

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Shortened form of "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
expr & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & | & c & (\text{constant}, c \in \mathbb{I}) \\
 & | & expr_1 \diamond expr_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

Variable

Integer Values

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, expr \,]\!]\rho$

- Now we can define evalExpr($expr$, env) for an environment $env \in \mathcal{E}$

$$\text{evalExpr}(V,\ env) \stackrel{\text{def}}{=} env(V)$$

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{llll}
expr & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & \mid & c & (\text{constant}, c \in \mathbb{I}) \\
 & \mid & expr_1 \diamond expr_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

Variable — Integer Values

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, expr \,]\!]\rho$

- Now we can define evalExpr($expr$, env) for an environment $env \in \mathcal{E}$

$$\text{evalExpr}(V,\ env) \stackrel{\text{def}}{=} env(V)$$

Value of $V \in \mathbb{V}$ in Environment env

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & \mid & c & (\text{constant}, c \in \mathbb{I}) \\
 & \mid & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \underbrace{\mathbb{V}}_{\text{Variable}} \rightarrow \underbrace{\mathbb{I}}_{\text{Integer Values}}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

- Now we can define evalExpr(expr, env) for an environment $\mathit{env} \in \mathcal{E}$

Usually written as $\mathbb{E}[\![\mathit{expr}]\!]\rho$

$$
\begin{aligned}
\text{evalExpr}(V, \mathit{env}) &\stackrel{\text{def}}{=} \mathit{env}(V) \\
\text{evalExpr}(c, \mathit{env}) &\stackrel{\text{def}}{=} c
\end{aligned}
$$

Value of $V \in \mathbb{V}$ in Environment env

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$\begin{array}{rcll} expr & ::= & V & (\text{variable}, V \in \mathbb{V}) \\ & | & c & (\text{constant}, c \in \mathbb{I}) \\ & | & expr_1 \diamond expr_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\}) \end{array}$$

Variable
Integer Values

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, expr \,]\!]\rho$

- Now we can define $\text{evalExpr}(expr, env)$ for an environment $env \in \mathcal{E}$

$$\begin{aligned} \text{evalExpr}(V,\ env) &\overset{\text{def}}{=} env(V) \\ \text{evalExpr}(c,\ env) &\overset{\text{def}}{=} c \\ \text{evalExpr}(expr_1 + expr_2,\ env) &\overset{\text{def}}{=} \text{evalExpr}(expr_1,\ env) + \text{evalExpr}(expr_2,\ env) \end{aligned}$$

Value of $V \in \mathbb{V}$ in Environment env

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
\mathit{expr} & ::= & V & \text{(variable, } V \in \mathbb{V}\text{)} \\
 & | & c & \text{(constant, } c \in \mathbb{I}\text{)} \\
 & | & \mathit{expr}_1 \diamond \mathit{expr}_2 & \text{(bin. expr., } \diamond \in \{+, -, \ldots\}\text{)}
\end{array}
$$

Variable

Integer Values

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\ \mathit{expr}\ ]\!]\rho$

- Now we can define evalExpr(expr, env) for an environment $\mathit{env} \in \mathcal{E}$

Value of $V \in \mathbb{V}$ in Environment env

$$
\begin{array}{rcl}
\text{evalExpr}(V,\ \mathit{env}) & \stackrel{\text{def}}{=} & \mathit{env}(V) \\
\text{evalExpr}(c,\ \mathit{env}) & \stackrel{\text{def}}{=} & c \\
\text{evalExpr}(\mathit{expr}_1 + \mathit{expr}_2,\ \mathit{env}) & \stackrel{\text{def}}{=} & \text{evalExpr}(\mathit{expr}_1,\ \mathit{env}) + \text{evalExpr}(\mathit{expr}_2,\ \mathit{env}) \\
 & \vdots &
\end{array}
$$

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{aligned}
\textcolor{red}{expr} \quad ::= \quad & V && \text{(variable, } V \in \mathbb{V}\text{)} \\
| \quad & c && \text{(constant, } c \in \mathbb{I}\text{)} \\
| \quad & expr_1 \diamond expr_2 && \text{(bin. expr., } \diamond \in \{+, -, \ldots\}\text{)}
\end{aligned}
$$

Variable   Integer Values

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, expr \,]\!]\rho$

- Now we can define evalExpr($expr$, env) for an environment $env \in \mathcal{E}$

$$
\begin{aligned}
\text{evalExpr}(V,\ env) &\stackrel{\text{def}}{=} env(V) \\
\text{evalExpr}(c,\ env) &\stackrel{\text{def}}{=} c \\
\text{evalExpr}(expr_1 + expr_2,\ env) &\stackrel{\text{def}}{=} \text{evalExpr}(expr_1,\ env) + \text{evalExpr}(expr_2,\ env) \\
&\ \ \vdots
\end{aligned}
$$

Value of $V \in \mathbb{V}$ in Environment env

- Additionally, we can define evalCond($cond$, $envs$) and evalStm(stm, $envs$)

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{rcll}
expr & ::= & V & \text{(variable, } V \in \mathbb{V}\text{)} \\
 & | & c & \text{(constant, } c \in \mathbb{I}\text{)} \\
 & | & expr_1 \diamond expr_2 & \text{(bin. expr., } \diamond \in \{+, -, \dots\}\text{)}
\end{array}
$$

Variable — Integer Values

- We use an environment $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, expr \,]\!]\rho$

- Now we can define evalExpr($expr$, env) for an environment $env \in \mathcal{E}$

$$
\begin{aligned}
\text{evalExpr}(V,\ env) &\stackrel{\text{def}}{=} env(V) \\
\text{evalExpr}(c,\ env) &\stackrel{\text{def}}{=} c \\
\text{evalExpr}(expr_1 + expr_2,\ env) &\stackrel{\text{def}}{=} \text{evalExpr}(expr_1,\ env) + \text{evalExpr}(expr_2,\ env) \\
&\ \ \vdots
\end{aligned}
$$

Value of $V \in \mathbb{V}$ in Environment env

$\mathbb{C}[\![\, cond \,]\!]\mathcal{D}$

- Additionally, we can define evalCond($cond$, $envs$) and evalStm(stm, $envs$)

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Atomic Expression Semantics

```java
int x = 0;
while(x < 2) {
  x = x + 1;
}
```

java

$$
\begin{array}{llll}
\mathit{expr} & ::= & V & (\text{variable}, V \in \mathbb{V}) \\
 & \mid & c & (\text{constant}, c \in \mathbb{I}) \\
 & \mid & \mathit{expr}_1 \diamond \mathit{expr}_2 & (\text{bin. expr.}, \diamond \in \{+, -, \ldots\})
\end{array}
$$

Variable Integer Values

- We use an environment $\mathcal{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ to represent the current program state

$\mathbb{V}$	$\mathbb{I}$
x	0
c	5

Usually written as $\mathbb{E}[\![\, \mathit{expr} \,]\!]\rho$

- Now we can define evalExpr(expr, env) for an environment $\mathit{env} \in \mathcal{E}$

$$
\begin{aligned}
\text{evalExpr}(V,\ \mathit{env}) &\overset{\text{def}}{=} \mathit{env}(V) \\
\text{evalExpr}(c,\ \mathit{env}) &\overset{\text{def}}{=} c \\
\text{evalExpr}(\mathit{expr}_1 + \mathit{expr}_2,\ \mathit{env}) &\overset{\text{def}}{=} \text{evalExpr}(\mathit{expr}_1,\ \mathit{env}) + \text{evalExpr}(\mathit{expr}_2,\ \mathit{env}) \\
&\ \ \vdots
\end{aligned}
$$

Value of $V \in \mathbb{V}$ in Environment env

$\mathbb{C}[\![\, \mathit{cond} \,]\!]\mathcal{D}$ $\mathbb{S}[\![\, \mathit{stm} \,]\!]\mathcal{D}$

- Additionally, we can define evalCond(cond, envs) and evalStm(stm, envs)

Shortened form of “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” [Min17, p. 46], craftinginterpreters.com/representing-code.html

Denotational Semantics while loops

Denotational Semantics while loops

$$\textbf{while}(\mathit{cond})\ \{\ \mathit{stm}\ \}$$

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$$\textbf{while}(\textcolor{purple}{\mathit{cond}})\ \{\ \textcolor{purple}{\mathit{stm}}\ \}$$

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$$\textbf{while}(\textcolor{red}{cond})\ \{\ \textcolor{red}{stm}\ \}\quad F(X) \overset{\text{def}}{=} \mathit{Start} \cup \mathit{evalStm}(\textcolor{red}{stm},\ \mathit{evalCond}(\textcolor{red}{cond},\ X))$$

iterate to find the least fixpoint [Min17, p. 52]

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$$\textbf{while}(cond)\ \{\ stm\ \} \quad F(X) \stackrel{\text{def}}{=} Start \cup evalStm(stm,\ evalCond(cond,\ X))$$

iterate to find the least fixpoint [Min17, p. 52]

Try to map this to the *Interval Analysis, I* example!

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$\textbf{while}(cond)\ \{\ stm\ \}$ $\quad F(X) \stackrel{\text{def}}{=} Start \cup evalStm(stm,\ evalCond(cond,\ X))$

iterate to find the least fixpoint [Min17, p. 52]

Try to map this to the *Interval Analysis, I* example!

Keep only states S with $evalCond(\neg cond,\ S)$

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$\textbf{while}(\mathit{cond})\ \{\ \mathit{stm}\ \}$ $\quad F(X) \stackrel{\text{def}}{=} \mathit{Start} \cup \mathit{evalStm}(\mathit{stm},\ \mathit{evalCond}(\mathit{cond},\ X))$

iterate to find the least fixpoint [Min17, p. 52]

Try to map this to the *Interval Analysis, I* example!

Keep only states S with $\mathit{evalCond}(\neg\mathit{cond},\ S)$

There are alternatives (e.g., equation systems, [Cou21, part 7])

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$$\textbf{while}(\textit{cond})\ \{\ \textit{stm}\ \} \qquad F(X) \stackrel{\text{def}}{=} \textit{Start} \cup \textit{evalStm}(\textit{stm},\ \textit{evalCond}(\textit{cond},\ X))$$

iterate to find the least fixpoint [Min17, p. 52]

Try to map this to the *Interval Analysis*, *I* example!

Keep only states S with $\textit{evalCond}(\neg \textit{cond},\ S)$

There are alternatives (e.g., equation systems, [Cou21, part 7])

We achieve their abstract counterpart using
the same principles but for abstract domains!

Denotational Semantics while loops

Suppose we start the loop with states *Start*

$\textbf{while}(cond)\ \{\ stm\ \}$ $F(X) \stackrel{\text{def}}{=} Start \cup evalStm(stm,\ evalCond(cond,\ X))$

iterate to find the least fixpoint [Min17, p. 52]

Try to map this to the *Interval Analysis, I* example!

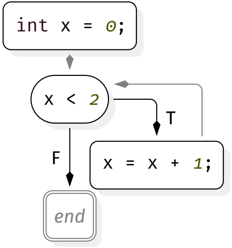
Keep only states S with $evalCond(\neg cond,\ S)$

There are alternatives (e.g., equation systems, [Cou21, part 7])

We achieve their abstract counterpart using
the same principles but for abstract domains!

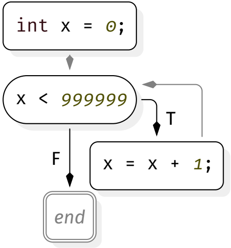
Usually written as $\mathbb{S}^\#, \mathbb{C}^\#, \mathbb{E}^\#, \ldots$

Interval Analysis, II



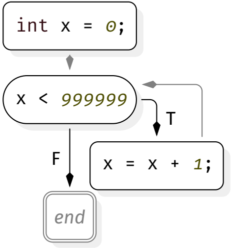
“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

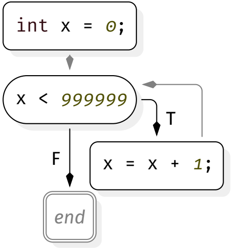
Interval Analysis, II



$\wr\, x_0 \in [0\,..\,0]\, \wr$

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II

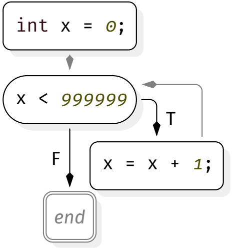


$\wr\, x_0 \in [0\,..\,0] \,\int$

$\wr$ [pre] $x_1 \in [0\,..\,0] \,\int$

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



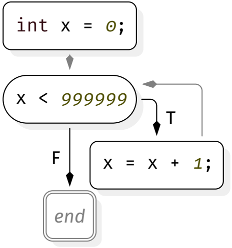
$\{\!\!\{\; x_0 \in [0\,..\,0] \;\}\!\!\}$

$\{\!\!\{\; \text{[pre]}\; x_1 \in [0\,..\,0] \;\}\!\!\}$

$\{\!\!\{\; \text{[in]}\; x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1]) \;\}\!\!\}$

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



$$\lbag\, x_0 \in [0 \mathinner{..} 0] \,\rbag$$

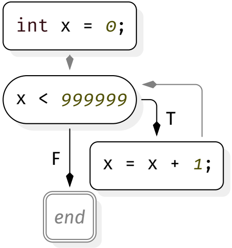
$$\lbag\, \text{[pre]}\ x_1 \in [0 \mathinner{..} 0] \,\rbag$$

$$\lbag\, \text{[in]}\ x_2 \in [0 \mathinner{..} 0] \quad ([0 \mathinner{..} 0] \cap (-\infty \mathinner{..} 1])) \,\rbag$$

$$\lbag\, x_3 \in [1 \mathinner{..} 1] \quad ([0 \mathinner{..} 0] \oplus [1 \mathinner{..} 1]) \,\rbag$$

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II

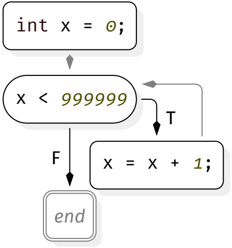


$$\wr\ x_0 \in [0\,..\,0]\ \int$$

$$\wr\ [\text{pre}]\ \boldsymbol{x_1 \in [0\,..\,0]}\ \int$$
$$\wr\ [\text{in}]\ x_2 \in [0\,..\,0] \quad ([0\,..\,0] \cap (-\infty\,..\,1])\ \int$$
$$\wr\ x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1])\ \int$$

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



$\wr\, x_0 \in [0 \,..\, 0] \,\int$

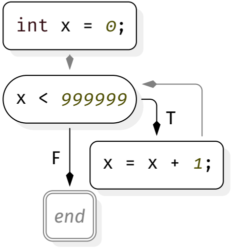
$\wr$ [pre] $\boldsymbol{x_1 \in [0 \,..\, 1] \quad ([0 \,..\, 0] \cup [1 \,..\, 1])} \,\int$

$\wr$ [in] $x_2 \in [0 \,..\, 0] \quad ([0 \,..\, 0] \cap (-\infty \,..\, 1]) \,\int$

$\wr\, x_3 \in [1 \,..\, 1] \quad ([0 \,..\, 0] \oplus [1 \,..\, 1]) \,\int$

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



$$x_0 \in [0\,..\,0]$$

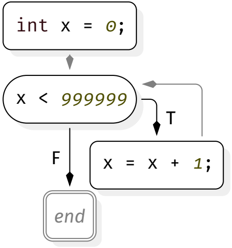
$$\text{[pre]}\ x_1 \in [0\,..\,1] \quad ([0\,..\,0] \cup [1\,..\,1])$$

$$\text{[in]}\ \boldsymbol{x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1])}$$

$$x_3 \in [1\,..\,1] \quad ([0\,..\,0] \oplus [1\,..\,1])$$

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



$$\{\!\{\ x_0 \in [0 \mathinner{..} 0]\ \}\!\}$$

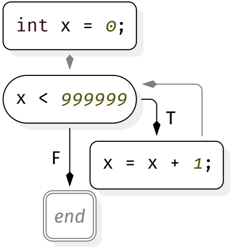
$$\{\!\{\ \text{[pre]}\ x_1 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 0] \cup [1 \mathinner{..} 1])\ \}\!\}$$

$$\{\!\{\ \text{[in]}\ x_2 \in [0 \mathinner{..} 1] \quad ([0 \mathinner{..} 1] \cap (-\infty \mathinner{..} 1])\ \}\!\}$$

$$\{\!\{\ \boldsymbol{x_3 \in [1 \mathinner{..} 2] \quad ([0 \mathinner{..} 1] \oplus [1 \mathinner{..} 1])}\ \}\!\}$$

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



$\wr\, x_0 \in [0\,..\,0] \,\int$

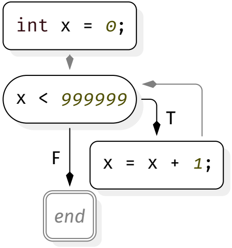
$\wr$ [pre] $\boldsymbol{x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2])}\,\int$

$\wr$ [in] $x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1])\,\int$

$\wr\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1])\,\int$

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



$\wr\, x_0 \in [0\,..\,0]\, \int$

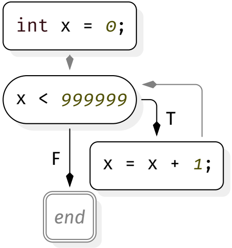
$\wr$ [pre] $\boldsymbol{x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2])}\, \int \quad \cdots$

$\wr$ [in] $x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1])\, \int$

$\wr\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1])\, \int$

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



$$\lbag\, x_0 \in [0 \,..\, 0] \,\rbag$$

$$\lbag\, \text{[pre]}\ \boldsymbol{x_1 \in [0 \,..\, 2] \quad ([0 \,..\, 1] \cup [1 \,..\, 2])} \,\rbag \quad \cdots$$

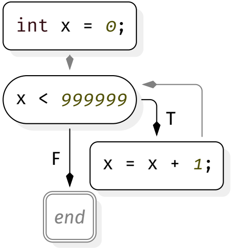
$$\lbag\, \text{[in]}\ x_2 \in [0 \,..\, 1] \quad ([0 \,..\, 1] \cap (-\infty \,..\, 1]) \,\rbag$$

$$\lbag\, x_3 \in [1 \,..\, 2] \quad ([0 \,..\, 1] \oplus [1 \,..\, 1]) \,\rbag$$

- Fixpoint iteration can be *very* expensive, and may not stabilize

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



$\{\, x_0 \in [0\,..\,0] \,\}$

$\{$ [pre] $\boldsymbol{x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2])}\,\} \quad \cdots$

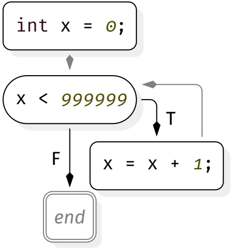
$\{$ [in] $x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1])\,\}$

$\{\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1])\,\}$

- Fixpoint iteration can be *very* expensive, and may not stabilize
- *Widening* (∇) is crucial, computing an upper bound
  (We only need widening if the lattice has an infinite ascending chain!)

“Principles of Abstract Interpretation” [Cou21, p. 393], “Widening and narrowing operators for abstract interpretation” [CZ11]

Interval Analysis, II



$\wr\, x_0 \in [0\,..\,0] \,\int$

$\wr\, \text{[pre]}\ \boldsymbol{x_1 \in [0\,..\,2] \quad ([0\,..\,1] \cup [1\,..\,2])} \,\int \quad \nabla \implies x_1 \in [0\,..\,\infty)$

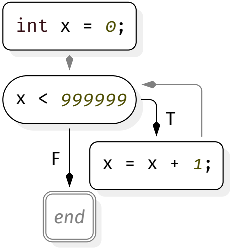
$\wr\, \text{[in]}\ x_2 \in [0\,..\,1] \quad ([0\,..\,1] \cap (-\infty\,..\,1]) \,\int$

$\wr\, x_3 \in [1\,..\,2] \quad ([0\,..\,1] \oplus [1\,..\,1]) \,\int$

- Fixpoint iteration can be *very* expensive, and may not stabilize
- *Widening* (∇) is crucial, computing an upper bound
  (We only need widening if the lattice has an infinite ascending chain!)

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Interval Analysis, II



$$\{\, x_0 \in [0 \,..\, 0] \,\}$$

$$\{\, \text{[pre]}\; x_1 \in [0 \,..\, 2] \quad ([0 \,..\, 1] \cup [1 \,..\, 2]) \,\} \quad \nabla \quad \Longrightarrow \quad x_1 \in [0 \,..\, \infty)$$

$$\{\, \text{[in]}\; x_2 \in [0 \,..\, 1] \quad ([0 \,..\, 1] \cap (-\infty \,..\, 1]) \,\}$$

$$\{\, x_3 \in [1 \,..\, 2] \quad ([0 \,..\, 1] \oplus [1 \,..\, 1]) \,\}$$

$$\{\, \text{[post]}\; x_4 \in [999999 \,..\, \infty) \quad ([0 \,..\, \infty) \cap [999999 \,..\, \infty)) \,\}$$

- Fixpoint iteration can be *very* expensive, and may not stabilize
- *Widening* (∇) is crucial, computing an upper bound
  (We only need widening if the lattice has an infinite ascending chain!)

"Principles of Abstract Interpretation" [Cou21, p. 393], "Widening and narrowing operators for abstract interpretation" [CZ11]

Let's Bring it All Together Sign Analysis

Let's Bring it All Together Sign Analysis

I want to make a sign analysis! (amazing!)

Let's Bring it All Together Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

Let's Bring it All Together — Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

2. Define the abstract semantics on the following language!

$$\mathit{expr} \quad ::= \quad c \qquad \text{(constant, } c \in \mathbb{R}\text{)}$$
$$\quad | \quad \mathit{expr}_1 + \mathit{expr}_2 \qquad \text{(addition)}$$

Let's Bring it All Together   Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

2. Define the abstract semantics on the following language!

$$
\begin{array}{lcll}
\mathit{expr} & ::= & c & \text{(constant, } c \in \mathbb{R}\text{)} \\
 & | & \mathit{expr}_1 + \mathit{expr}_2 & \text{(addition)}
\end{array}
$$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

Let's Bring it All Together

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

2. Define the abstract semantics on the following language!

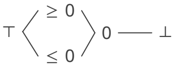
$$\begin{array}{llll} \mathit{expr} & ::= & c & \text{(constant, } c \in \mathbb{R}\text{)} \\ & \mid & \mathit{expr}_1 + \mathit{expr}_2 & \text{(addition)} \end{array}$$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

4. Do we need widening? If so, define ∇!

Let's Bring it All Together — Sign Analysis

I want to make a sign analysis! (amazing!)



1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here
   $\mathcal{L} = (x = \{\bot, 0, \leq 0, \geq 0, \top\},\ \leq = \{(\bot, 0), (\bot, \leq 0), \ldots\}, \sqcup = \{(\bot, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \ldots\},$
   $\sqcap = \{(\top, x) \mapsto x, (0, \leq 0) \mapsto 0, \ldots\}, \bot, \top)$

2. Define the abstract semantics on the following language!

expr	::=	c	(constant, $c \in \mathbb{R}$)
	|	$expr_1 + expr_2$	(addition)

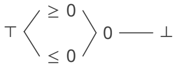
3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

4. Do we need widening? If so, define ∇!

Let's Bring it All Together Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here



$$\mathcal{L} = (x = \{\bot, 0, \leq 0, \geq 0, \top\},\ \leq = \{(\bot, 0), (\bot, \leq 0), \dots\}, \sqcup = \{(\bot, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \dots\},$$
$$\sqcap = \{(\top, x) \mapsto x, (0, \leq 0) \mapsto 0, \dots\}, \bot, \top)$$

2. Define the abstract semantics on the following language!

expr	::=	c	(constant, $c \in \mathbb{R}$)
	|	$expr_1 + expr_2$	(addition)

$$\mathbb{E}[\![\, \mathrm{c} \,]\!]\rho \overset{\text{def}}{=} \begin{cases} \geq 0 & \text{if } \mathrm{c} > 0 \\ 0 & \text{if } \mathrm{c} = 0 \\ \leq 0 & \text{if } \mathrm{c} < 0 \end{cases}$$

$$\mathbb{E}[\![\, \mathrm{a} + \mathrm{b} \,]\!]\rho \overset{\text{def}}{=} \mathbb{E}[\![\, \mathrm{a} \,]\!]\rho \oplus \mathbb{E}[\![\, \mathrm{b} \,]\!]\rho$$

$\oplus$	$\bot$	0	$\leq$	$\geq$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0		0	$\leq$	$\geq$	$\top$
$\leq$			$\leq$	$\top$	$\top$
$\geq$				$\geq$	$\top$
$\top$					$\top$

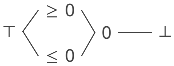
$x \oplus y = y \oplus x$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

4. Do we need widening? If so, define ∇!

Let's Bring it All Together Sign Analysis

I want to make a sign analysis! (amazing!)



1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here
   $\mathcal{L} = (x = \{\bot, 0, \leq 0, \geq 0, \top\},\ \leq = \{(\bot, 0), (\bot, \leq 0), \ldots\}, \sqcup = \{(\bot, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \ldots\},$
   $\sqcap = \{(\top, x) \mapsto x, (0, \leq 0) \mapsto 0, \ldots\}, \bot, \top)$

2. Define the abstract semantics on the following language!

expr	::=	c	(constant, $c \in \mathbb{R}$)
	|	$expr_1 + expr_2$	(addition)

short for evalExpr(*expr*, $\rho = env$)

$$\mathbb{E}[\![\, c \,]\!]\rho \overset{\text{def}}{=} \begin{cases} \geq 0 & \text{if } c > 0 \\ 0 & \text{if } c = 0 \\ \leq 0 & \text{if } c < 0 \end{cases}$$

$$\mathbb{E}[\![\, a + b \,]\!]\rho \overset{\text{def}}{=} \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$$

$\oplus$	$\bot$	0	$\leq$	$\geq$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0		0	$\leq$	$\geq$	$\top$
$\leq$			$\leq$	$\top$	$\top$
$\geq$				$\geq$	$\top$
$\top$					$\top$

$x \oplus y = y \oplus x$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!

4. Do we need widening? If so, define ∇!

Let's Bring it All Together   Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

$$
\begin{tikzcd}[column sep=small, row sep=tiny]
 & \geq 0 \arrow[dl, dash] \arrow[dr, dash] & & \\
\top & & 0 \arrow[r, dash] & \bot \\
 & \leq 0 \arrow[ul, dash] \arrow[ur, dash] & &
\end{tikzcd}
$$

$$\mathcal{L} = (x = \{\bot, 0, \leq 0, \geq 0, \top\},\ \leq = \{(\bot, 0), (\bot, \leq 0), \ldots\}, \sqcup = \{(\bot, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \ldots\},$$
$$\sqcap = \{(\top, x) \mapsto x, (0, \leq 0) \mapsto 0, \ldots\}, \bot, \top)$$

2. Define the abstract semantics on the following language!

$$
\begin{array}{lcll}
\mathit{expr} & ::= & c & \text{(constant, } c \in \mathbb{R}) \\
 & \mid & \mathit{expr}_1 + \mathit{expr}_2 & \text{(addition)}
\end{array}
$$

$$
\mathbb{E}[\![\, \mathrm{c} \,]\!]\rho \overset{\text{def}}{=} \begin{cases} \geq 0 & \text{if } \mathrm{c} > 0 \\ 0 & \text{if } \mathrm{c} = 0 \\ \leq 0 & \text{if } \mathrm{c} < 0 \end{cases}
$$

short for evalExpr(expr, $\rho = \mathit{env}$)

$$\mathbb{E}[\![\, \mathrm{a} + \mathrm{b} \,]\!]\rho \overset{\text{def}}{=} \mathbb{E}[\![\, \mathrm{a} \,]\!]\rho \oplus \mathbb{E}[\![\, \mathrm{b} \,]\!]\rho$$

$\oplus$	$\bot$	0	$\leq$	$\geq$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0		0	$\leq$	$\geq$	$\top$
$\leq$			$\leq$	$\top$	$\top$
$\geq$				$\geq$	$\top$
$\top$					$\top$

$x \oplus y = y \oplus x$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!
   $\gamma(0) = \{0\}, \quad \gamma(\leq 0) = \{n \in \mathbb{Z} \mid n \leq 0\}$

4. Do we need widening? If so, define ∇!

Let's Bring it All Together — Sign Analysis

I want to make a sign analysis! (amazing!)

1. Define the lattice! (there are many solutions)
   drawing the hasse diagram is enough for us here

$$\begin{tikzcd}[column sep=small] & \geq 0 \arrow[dr, dash] & & \\ \top \arrow[ur, dash] \arrow[dr, dash] & & 0 \arrow[r, dash] & \bot \\ & \leq 0 \arrow[ur, dash] & & \end{tikzcd}$$

   $\mathcal{L} = (x = \{\bot, 0, \leq 0, \geq 0, \top\},\ \leq = \{(\bot, 0), (\bot, \leq 0), \ldots\}, \sqcup = \{(\bot, x) \mapsto x, (0, \leq 0) \mapsto \leq 0, \ldots\},$
   $\sqcap = \{(\top, x) \mapsto x, (0, \leq 0) \mapsto 0, \ldots\}, \bot, \top)$

2. Define the abstract semantics on the following language!

expr	::=	c	(constant, $c \in \mathbb{R}$)
	|	$expr_1 + expr_2$	(addition)

$$\mathbb{E}[\![\ c\ ]\!]\rho \overset{\text{def}}{=} \begin{cases} \geq 0 & \text{if } c > 0 \\ 0 & \text{if } c = 0 \\ \leq 0 & \text{if } c < 0 \end{cases}$$

short for evalExpr(*expr*, $\rho = env$)

$$\mathbb{E}[\![\ a + b\ ]\!]\rho \overset{\text{def}}{=} \mathbb{E}[\![\ a\ ]\!]\rho \oplus \mathbb{E}[\![\ b\ ]\!]\rho$$

$\oplus$	$\bot$	0	$\leq$	$\geq$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0		0	$\leq$	$\geq$	$\top$
$\leq$			$\leq$	$\top$	$\top$
$\geq$				$\geq$	$\top$
$\top$					$\top$

$x \oplus y = y \oplus x$

3. Provide an integer concretization for $\gamma(0)$ and $\gamma(\leq 0)$!
   $\gamma(0) = \{0\}, \quad \gamma(\leq 0) = \{n \in \mathbb{Z} \mid n \leq 0\}$

4. Do we need widening? If so, define ∇!
   We do not need widening here as our lattice is finite and our semantics do not introduce new elements. (We will concretize this next week.)

4. Outlook and Comments

This is incredible, I need more!

The Domains We Built…

The Domains We Built…

- We know a handful of important concepts for our domains:

The Domains We Built…

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)

The Domains We Built...

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)

The Domains We Built…

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\perp$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \dots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)

The Domains We Built. . .

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \ldots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)
  - Fixpoint Iterations (to interpret loops, recursion, . . .)

The Domains We Built…

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \ldots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)
  - Fixpoint Iterations (to interpret loops, recursion, …)
  - Widening (to ensure termination with infinite chains)

The Domains We Built…

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\perp$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \dots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)
  - Fixpoint Iterations (to interpret loops, recursion, …)
  - Widening (to ensure termination with infinite chains)
  - Galois Connections (to relate concrete and abstract domains)

The Domains We Built...

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \ldots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)
  - Fixpoint Iterations (to interpret loops, recursion, ...)
  - Widening (to ensure termination with infinite chains)
  - Galois Connections (to relate concrete and abstract domains)

- With abstract interpretation we interpret programs *over* these domains!

The Domains We Built...

- We know a handful of important concepts for our domains:
  - Lattices (poset, with join $\sqcup$, meet $\sqcap$, bottom $\bot$, and top $\top$)
  - Program Semantics ($\mathbb{S}[\![\, stm \,]\!]\mathcal{D}, \ldots$, e.g., $\mathbb{E}[\![\, a + b \,]\!]\rho = \mathbb{E}[\![\, a \,]\!]\rho \oplus \mathbb{E}[\![\, b \,]\!]\rho$)
  - Fixpoint Iterations (to interpret loops, recursion, ...)
  - Widening (to ensure termination with infinite chains)
  - Galois Connections (to relate concrete and abstract domains)
- With abstract interpretation we interpret programs *over* these domains!
- However, we have only looked at single variables and single domains so far!

Combining Domains: Products

Combining Domains: Products

- Let's assume we have two domains D_1 and D_2

Combining Domains: Products

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

Combining Domains: Products

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

Saunders Mac Lane (1909–2005)
CC BY SA 2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
CC BY SA 2.0 Oberwolfach Photo Collection

Combining Domains: Products

Saunders Mac Lane (1909–2005)
CC BY-SA 2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
CC BY-SA 2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables!

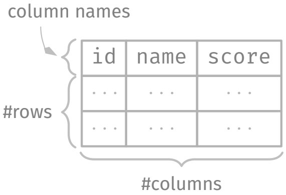
Combining Domains: Products

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

Saunders Mac Lane (1909–2005)
2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
2.0 Oberwolfach Photo Collection

- For example, let's suppose we want to track the shape of tables!



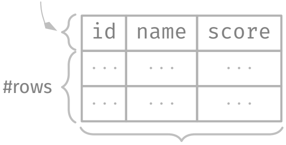
Combining Domains: Products

Saunders Mac Lane (1909–2005)
2.0 Oberwolfach
Photo Collection

Samuel Eilenberg (1913–1998)
2.0 Oberwolfach
Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables!
  We define three domains!



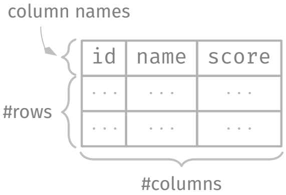
Combining Domains: Products

Saunders Mac Lane (1909–2005)
2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables! We define three domains!
  - $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)



Combining Domains: Products

Saunders Mac Lane (1909–2005)
CC BY-SA 2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
CC BY-SA 2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables! We define three domains!
  - $\mathcal{N} \overset{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)
  - $\mathcal{R} = \mathcal{C} \overset{\text{def}}{=} \{\, [a\,..\,b] \mid a, b \in \mathbb{N}_0 \cup \{\infty\}, a \leq b \,\}$ (# of rows/columns)

column names

#rows

id	name	score
...	...	...
...	...	...

#columns

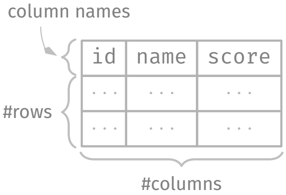
Combining Domains: Products

Saunders Mac Lane (1909–2005)
cc by sa 2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
cc by sa 2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables! We define three domains!
  - $\mathcal{N} \overset{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)
  - $\mathcal{R} = \mathcal{C} \overset{\text{def}}{=} \{\, [a\,..\,b] \mid a, b \in \mathbb{N}_0 \cup \{\infty\}, a \leq b \,\}$ (# of rows/columns)
  - Then we can define the domain: $\mathcal{DF} \overset{\text{def}}{=} \mathcal{N} \times \mathcal{R} \times \mathcal{C}$



Combining Domains: Products

Saunders Mac Lane (1909–2005)
cc by sa 2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
cc by sa 2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables! We define three domains!
  - $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)
  - $\mathcal{R} = \mathcal{C} \stackrel{\text{def}}{=} \{\, [a\,..\,b] \mid a, b \in \mathbb{N}_0 \cup \{\infty\}, a \leq b \,\}$ (# of rows/columns)
  - Then we can define the domain: $\mathcal{DF} \stackrel{\text{def}}{=} \mathcal{N} \times \mathcal{R} \times \mathcal{C}$

column names

id	name	score
...	...	...
...	...	...

#rows

#columns

Category theory is amazing!

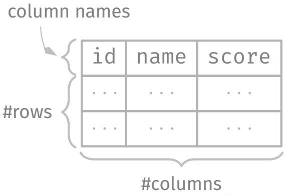
Combining Domains: Products

Saunders Mac Lane (1909–2005)
2.0 Oberwolfach Photo Collection

Samuel Eilenberg (1913–1998)
2.0 Oberwolfach Photo Collection

- Let's assume we have two domains D_1 and D_2
- We can combine them into a *product domain*: $D_1 \times D_2$

- For example, let's suppose we want to track the shape of tables! We define three domains!
  - $\mathcal{N} \overset{\text{def}}{=} \mathcal{P}(\Sigma^*)$ (set of column names)
  - $\mathcal{R} = \mathcal{C} \overset{\text{def}}{=} \{\, [a\mathinner{..}b] \mid a, b \in \mathbb{N}_0 \cup \{\infty\}, a \leq b \,\}$ (# of rows/columns)
  - Then we can define the domain: $\mathcal{DF} \overset{\text{def}}{=} \mathcal{N} \times \mathcal{R} \times \mathcal{C}$



[Ger25] Oliver Gerstl. Tracking the shape of data frames in R programs using abstract interpretation (Ulm University, 2025)

Category theory is amazing!

Outlook

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]

- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations [Cou21, p. 395]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations [Cou21, p. 395]
- Function calls
  require special handling [MJ12]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations [Cou21, p. 395]
- Function calls
  require special handling [MJ12]
- Existing libraries allow for easy implementation
  LiSA [Fer+21], MOPSA [Jou+19], Apron [JM09]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations [Cou21, p. 395]
- Function calls
  require special handling [MJ12]
- Existing libraries allow for easy implementation
  LiSA [Fer+21], MOPSA [Jou+19], Apron [JM09]
- There are other ways to define semantics
  e.g., small-step, big-step [Cio13]

Outlook

- Domain transformers
  combine abstract domains [Min17, p. 149]
- Galois connections (offer so much more)
  define the relationship between concrete and abstract domains [Cou21, p. 110]
- Corresponding to widening, narrowing
  refines approximations [Cou21, p. 395]
- Function calls
  require special handling [MJ12]
- Existing libraries allow for easy implementation
  LiSA [Fer+21], MOPSA [Jou+19], Apron [JM09]
- There are other ways to define semantics
  e.g., small-step, big-step [Cio13]

Domains can also capture relations between variables (e.g. *polyhedra*), their provenance, and much more!

However, this implies trade-offs which we discuss next time.

Back to the Questions

Back to the Questions

1. How would you capture what a *property* is?
2. How would you phrase that one property is “better” than another?
3. For what operations would you *not* use a control-flow graph?
4. Why can’t there be a fully automatic, sound, and complete static analyzer for general programs?
5. What (big) additional challenges do you see in the real-world?

Back to the Questions

1. How would you capture what a *property* is? ✔
2. How would you phrase that one property is “better” than another?
3. For what operations would you *not* use a control-flow graph?
4. Why can’t there be a fully automatic, sound, and complete static analyzer for general programs?
5. What (big) additional challenges do you see in the real-world?

Back to the Questions

1. How would you capture what a *property* is? ✔
2. How would you phrase that one property is “better” than another? ✔
3. For what operations would you *not* use a control-flow graph?
4. Why can’t there be a fully automatic, sound, and complete static analyzer for general programs?
5. What (big) additional challenges do you see in the real-world?

And… Back to the Real World?

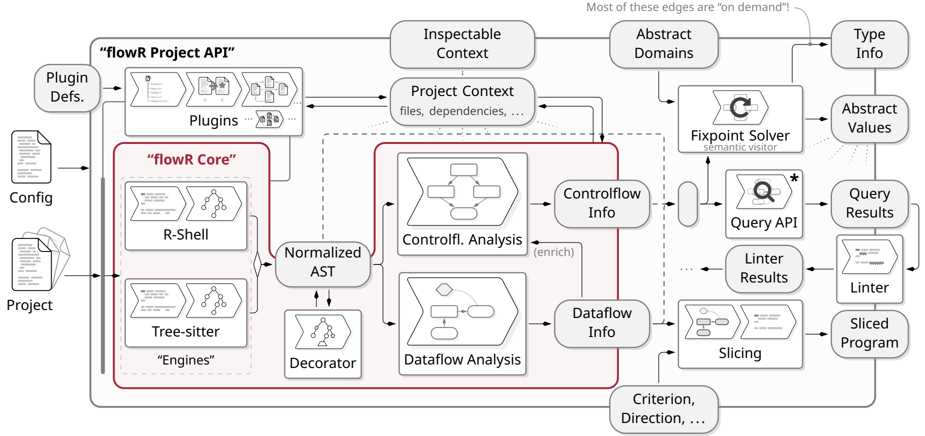
And… Back to the Real World?

In case you like/are intrigued by what you see, join the horde: **florian.sihler@uni-ulm.de**

And… Back to the Real World?

In case you like/are intrigued by what you see, join the horde: **florian.sihler@uni-ulm.de**

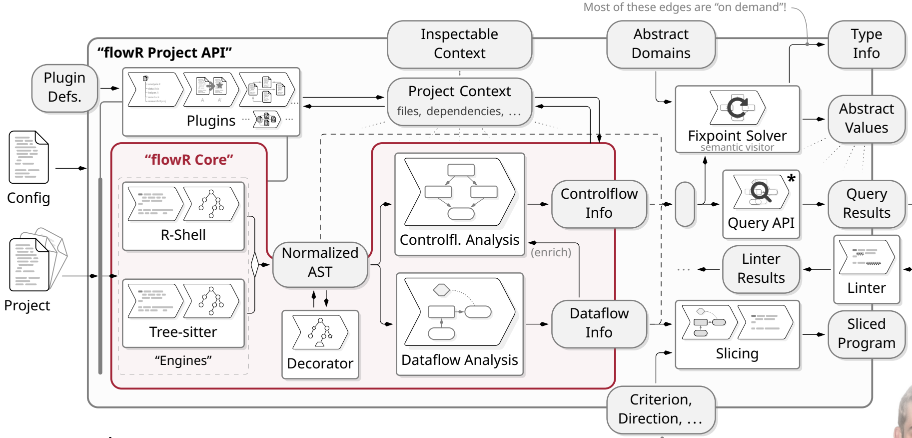
And... Back to the Real World?



* Many Queries: Clusters, Dependencies, Call-Contexts, Code Search, ...

In case you like/are intrigued by what you see, join the horde: **florian.sihler@uni-ulm.de**

And... Back to the Real World?



* Many Queries: Clusters, Dependencies, Call-Contexts, Code Search, ...

In case you like/are intrigued by what you see, join the horde: **florian.sihler@uni-ulm.de**

Bibliography

References I

[Bal+18] Roberto Baldoni et al. “A Survey of Symbolic Execution Techniques”. In: *ACM Comput. Surv.* 51.3 (2018), 50:1–50:39. DOI: 10.1145/3182657. URL: https://doi.org/10.1145/3182657.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq’Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN: 978-3-642-05880-6. DOI: 10.1007/978-3-662-07964-5. URL: https://doi.org/10.1007/978-3-662-07964-5.

[BEL75] Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. “SELECT - a formal system for testing and debugging programs by symbolic execution”. In: *Proceedings of the International Conference on Reliable Software 1975, Los Angeles, California, USA, April 21-23, 1975*. Ed. by Martin L. Shooman and Raymond T. Yeh. ACM, 1975, pp. 234–245. DOI: 10.1145/800027.808445. URL: https://doi.org/10.1145/800027.808445.

[Bir67] Garrett Birkhoff. “Lattice theory”. In: *Publications of AMS* (1967).

[CC77] Patrick Cousot and Radhia Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*. Ed. by Robert M. Graham, Michael A. Harrison, and Ravi Sethi. ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973. URL: https://doi.org/10.1145/512950.512973.

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”. In: *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*. Ed. by Richard Draves and Robbert van Renesse. USENIX Association, 2008, pp. 209–224. URL: http://www.usenix.org/events/osdi08/tech/full%5C_papers/cadar/cadar.pdf.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications”. In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986), pp. 244–263. DOI: 10.1145/5397.5399. URL: https://doi.org/10.1145/5397.5399.

[Cio13] Ştefan Ciobâcă. “From Small-Step Semantics to Big-Step Semantics, Automatically”. In: *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*. Ed. by Einar Broch Johnsen and Luigia Petre. Vol. 7940. Lecture Notes in Computer Science. Springer, 2013, pp. 347–361. DOI: 10.1007/978-3-642-38613-8_24. URL: https://doi.org/10.1007/978-3-642-38613-8%5C_24.

[CKL04] Edmund Clarke, Daniel Kroening, and Flavio Lerda. “A tool for checking ANSI-C programs”. In: *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29-April 2, 2004. Proceedings 10*. Springer. 2004, pp. 168–176.

References II

[Cou12] Patrick Cousot. “A casual introduction to Abstract Interpretation”. In: *CMACS Workshop on Systems Biology and Formals Methods (SBFM’12)* (2012). URL: https://pcousot.github.io/talks/PCousot-SBFM-2012-1-1.pdf (visited on 12/09/2024).

[Cou21] Patrick Cousot. “Principles of Abstract Interpretation”. In: (2021).

[CZ11] Agostino Cortesi and Matteo Zanioli. “Widening and narrowing operators for abstract interpretation”. In: *Comput. Lang. Syst. Struct.* 37.1 (2011), pp. 24–42. DOI: 10.1016/J.CL.2010.09.001. URL: https://doi.org/10.1016/j.cl.2010.09.001.

[Fer+21] Pietro Ferrara et al. “Static analysis for dummies: experiencing LiSA”. In: *SOAP@PLDI 2021: Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, Virtual Event, Canada, 22 June, 2021*. Ed. by Lisa Nguyen Quang Do and Caterina Urban. ACM, 2021, pp. 1–6. DOI: 10.1145/3460946.3464316. URL: https://doi.org/10.1145/3460946.3464316.

[Flo67] Robert W. Floyd. “Assigning Meanings to Programs”. In: *Proc. of the American Mathematical Society Symposia on Applied Mathematics*. Vol. 19. 1967, pp. 19–32.

[Ger25] Oliver Gerstl. *Tracking the shape of data frames in R programs using abstract interpretation*. en. Master’s Thesis. 2025. DOI: 10.18725/OPARU-58621. URL: https://oparu.uni-ulm.de//handle/123456789/58696.

[GR22] Roberto Giacobazzi and Francesco Ranzato. “History of Abstract Interpretation”. In: *IEEE Ann. Hist. Comput.* 44.2 (2022), pp. 33–43. DOI: 10.1109/MAHC.2021.3133136. URL: https://doi.org/10.1109/MAHC.2021.3133136.

[Hoa69] C. A. R. Hoare. “An Axiomatic Basis for Computer Programming”. In: *Commun. ACM* 12.10 (1969), pp. 576–580. DOI: 10.1145/363235.363259. URL: https://doi.org/10.1145/363235.363259.

[JM09] Bertrand Jeannet and Antoine Miné. “Apron: A Library of Numerical Abstract Domains for Static Analysis”. In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. Ed. by Ahmed Bouajjani and Oded Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 661–667. DOI: 10.1007/978-3-642-02658-4_52. URL: https://doi.org/10.1007/978-3-642-02658-4%5C_52.

[Jou+19] Matthieu Journault et al. “Combinations of Reusable Abstract Domains for a Multilingual Static Analyzer”. In: *Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers*. Ed. by Supratik Chakraborty and Jorge A. Navas. Vol. 12031. Lecture Notes in Computer Science. Springer, 2019, pp. 1–18. DOI: 10.1007/978-3-030-41600-3_1. URL: https://doi.org/10.1007/978-3-030-41600-3%5C_1.

[Kin74] James C. King. “A New Approach to Program Testing”. In: *Programming Methodology, 4th Informatik Symposium, IBM Germany, Wildbad, September 25-27, 1974*. Ed. by Clemens Hackl. Vol. 23. Lecture Notes in Computer Science. Springer, 1974, pp. 278–290. DOI: 10.1007/3-540-07131-8_30. URL: https://doi.org/10.1007/3-540-07131-8%5C_30.

References III

[Kle52] Stephen Cole Kleene. “Introduction to metamathematics”. In: (1952).

[KSK09] Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. 1st. USA: CRC Press, Inc., 2009. ISBN: 0849328802.

[LF08] Francesco Logozzo and Manuel Fähndrich. “Pentagons: a weakly relational abstract domain for the efficient validation of array accesses”. In: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*. Ed. by Roger L. Wainwright and Hisham Haddad. ACM, 2008, pp. 184–188. DOI: 10.1145/1363686.1363736. URL: https://doi.org/10.1145/1363686.1363736.

[Mau04] Laurent Mauborgne. “Astrée: verification of absence of run-time error”. In: *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*. Ed. by René Jacquart. Vol. 156. IFIP. Kluwer/Springer, 2004, pp. 385–392. DOI: 10.1007/978-1-4020-8157-6_30. URL: https://doi.org/10.1007/978-1-4020-8157-6%5C_30.

[Min17] Antoine Miné. “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation”. In: *Found. Trends Program. Lang.* 4.3-4 (2017), pp. 120–372. DOI: 10.1561/2500000034. URL: https://doi.org/10.1561/2500000034.

[MJ12] Jan Midtgaard and Thomas P. Jensen. “Control-flow analysis of function calls and returns by abstract interpretation”. In: *Inf. Comput.* 211 (2012), pp. 49–76. DOI: 10.1016/J.IC.2011.11.005. URL: https://doi.org/10.1016/j.ic.2011.11.005.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. “PVS: A Prototype Verification System”. In: *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*. Ed. by Deepak Kapur. Vol. 607. Lecture Notes in Computer Science. Springer, 1992, pp. 748–752. DOI: 10.1007/3-540-55602-8_217. URL: https://doi.org/10.1007/3-540-55602-8%5C_217.

[Ric53] Henry Gordon Rice. “Classes of recursively enumerable sets and their decision problems”. In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.

[RY20] Xavier Rival and Kwangkeun Yi. “Introduction to Static Analysis: An Abstract Interpretation Perspective”. In: (2020).

[Tar55] Alfred Tarski. “A lattice-theoretical fixpoint theorem and its applications.”. In: (1955).

[Tur49] Alan Turing. “Checking a large routine”. In: *Report of a Conference on High Speed Automatic Calculating Machines*. 1949, pp. 67–69.

A Little Bit of History

Static Analysis

Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]

Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]

1949: First Checks
Turing [Tur49]

1953: Rice Theorem
Non-trivial properties are undecidable [Ric53]

Static Analysis

Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]

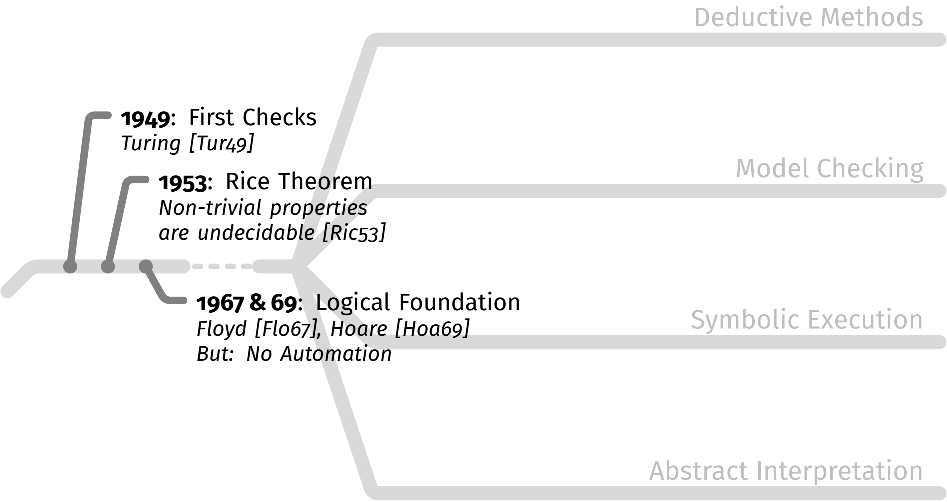
1949: First Checks
Turing [Tur49]

1953: Rice Theorem
Non-trivial properties
are undecidable [Ric53]

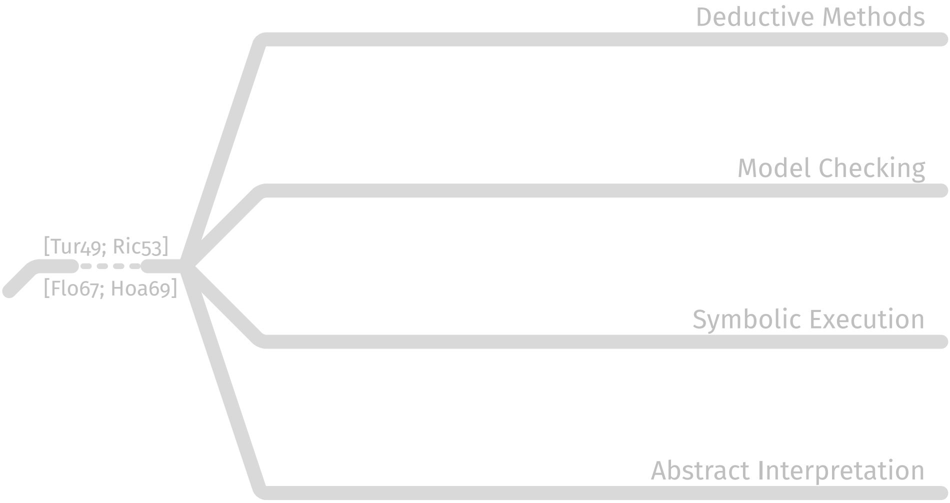
Static Analysis

1967 & 69: Logical Foundation
Floyd [Flo67], Hoare [Hoa69]
But: No Automation

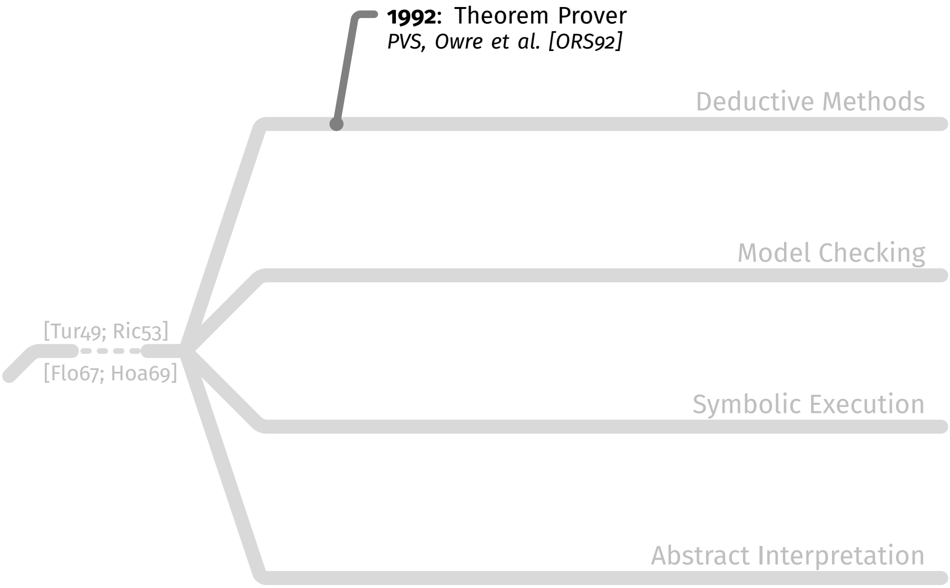
Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



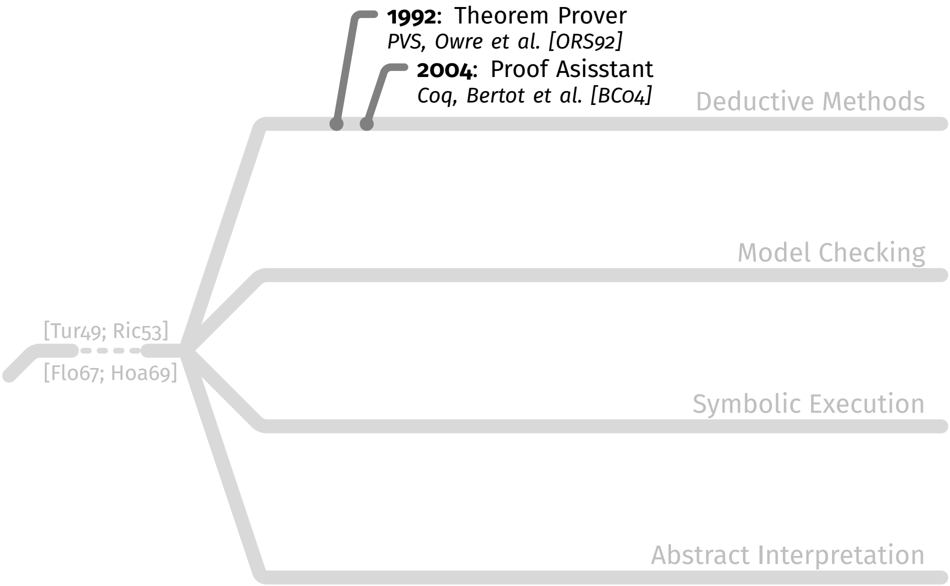
Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



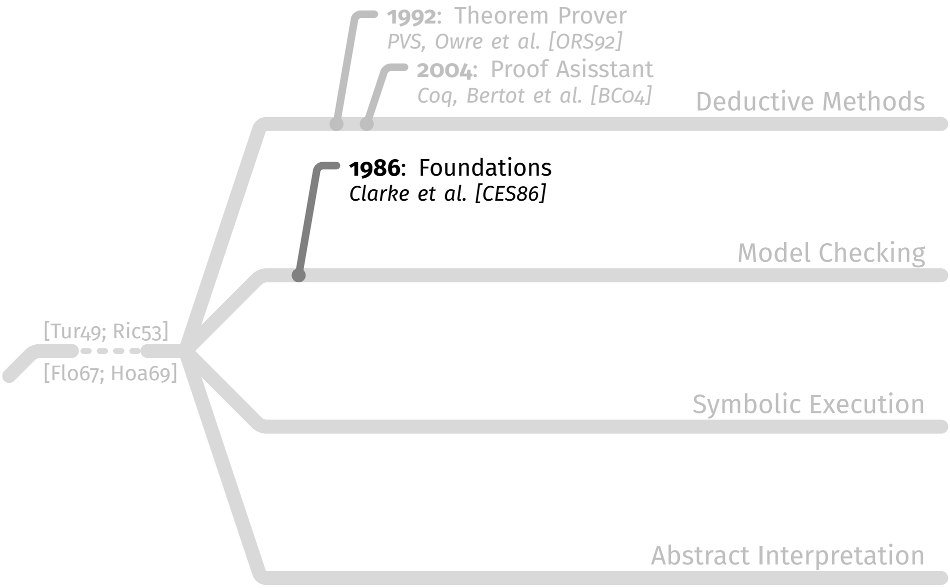
Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



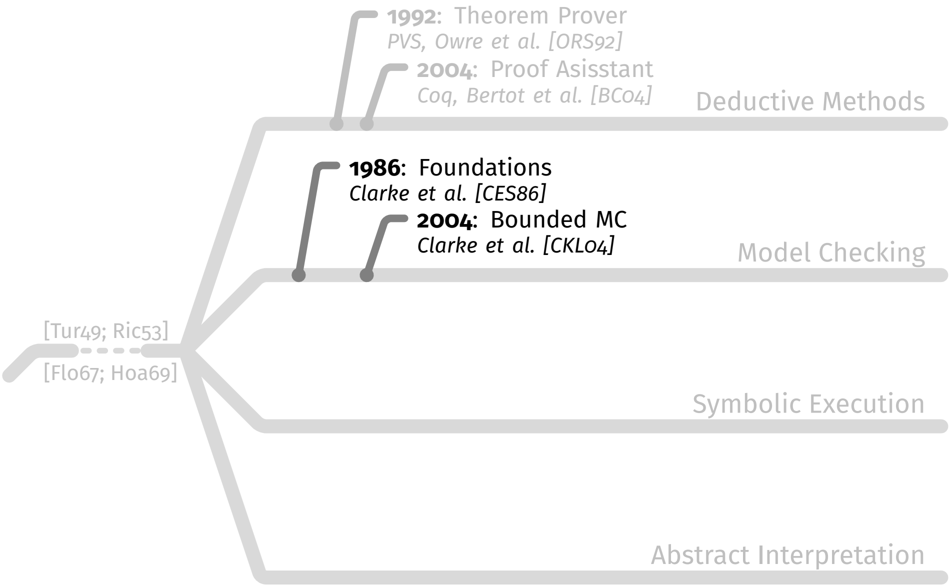
Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



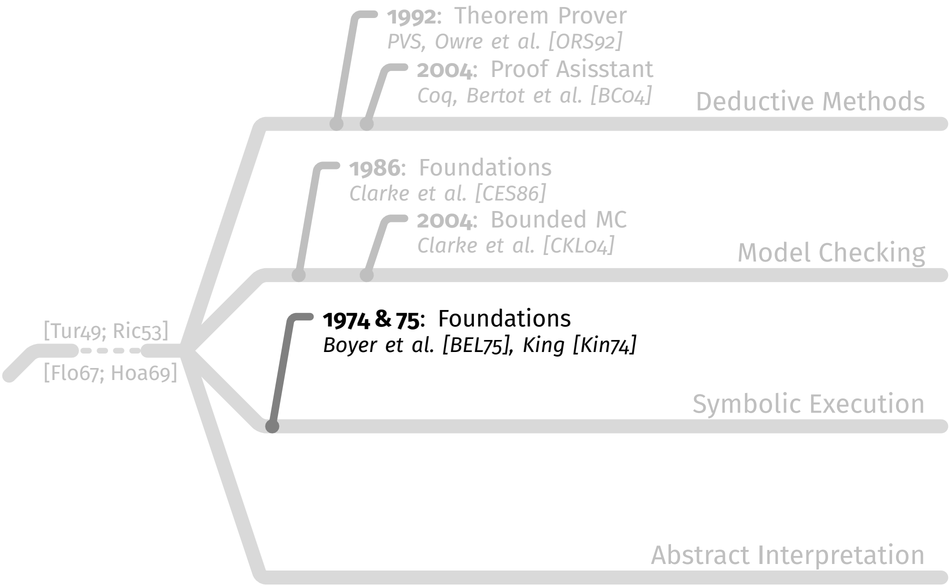
Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



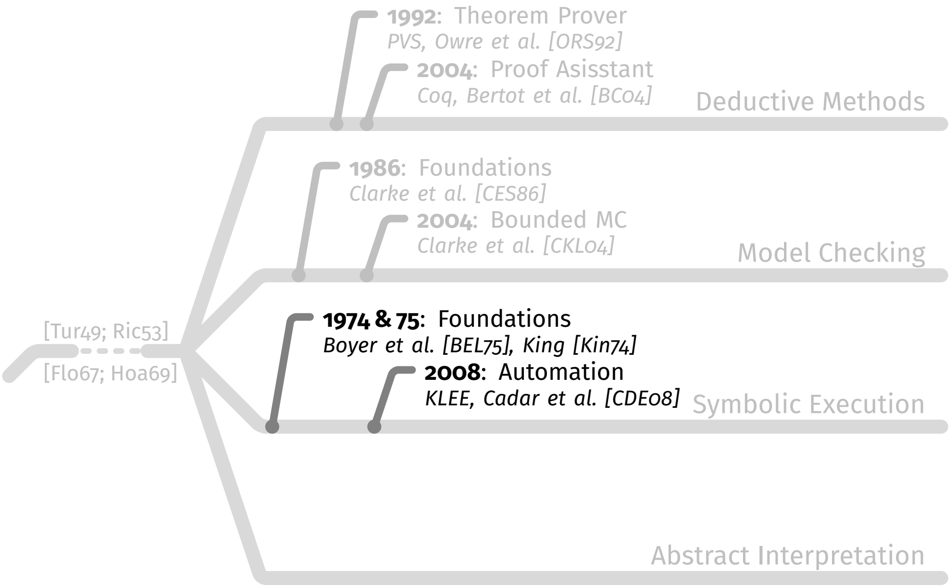
Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



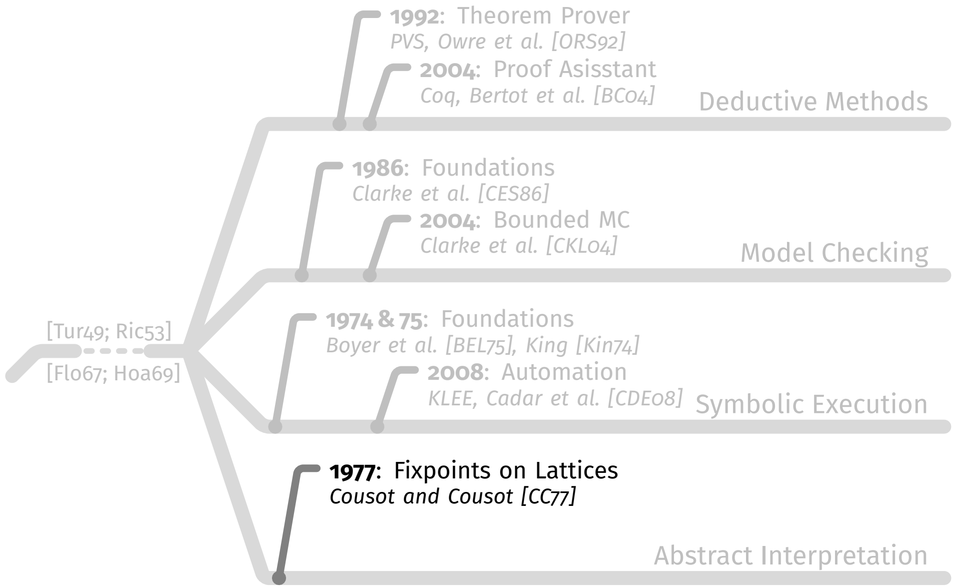
Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



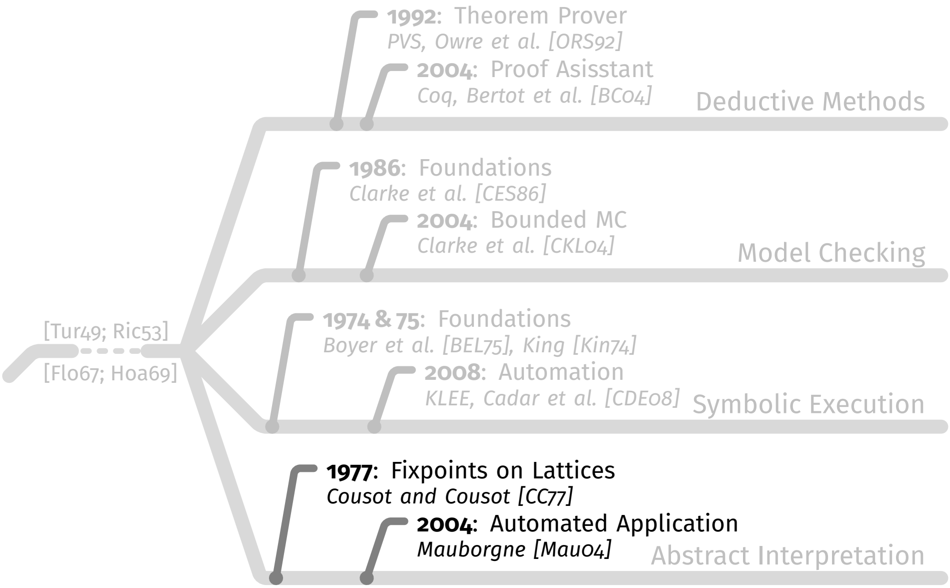
Based on the amazing "Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation" by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]



Based on the amazing “Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation” by Miné [Min17], https://www.di.ens.fr/~cousot/AI/, and [Bal+18; GR22]