Foundations of Abstract Interpretation

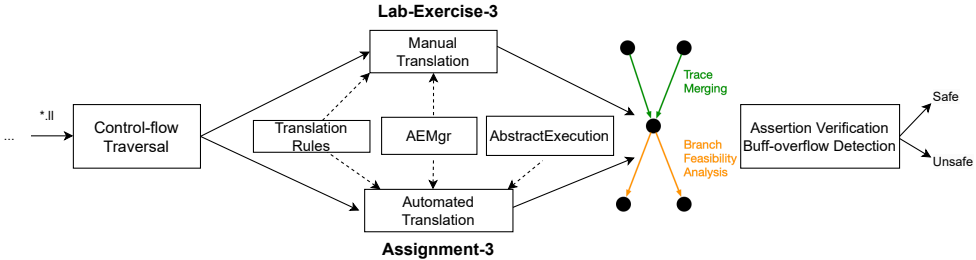
(Week 8)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Classes in the Next Three Weeks



Today's Outline

- Introduction to Abstract Interpretation: What and Why
- Abstract Interpretation vs. Symbolic Execution
- Definitions: abstract domains, abstract states, and abstract traces.
- Step-by-Step Motivating Examples.
- Widening and Narrowing: Improving Analysis Speed and Precision

Abstract Interpretation

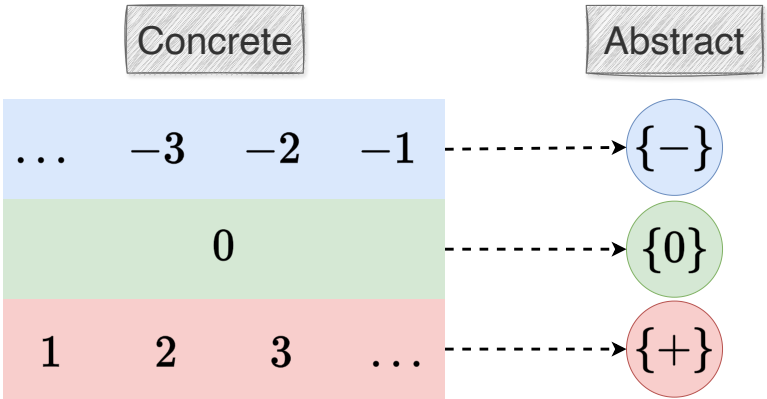
Abstract interpretation [Cousot & Cousot, POPL'77][1], a general framework for static analysis, aims to **soundly approximate** the potential concrete values that program variables may take during runtime, **based on monotonic functions over ordered sets, particularly lattices**.

Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.

Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.



Abstract Interpretation: Levels of Abstractions

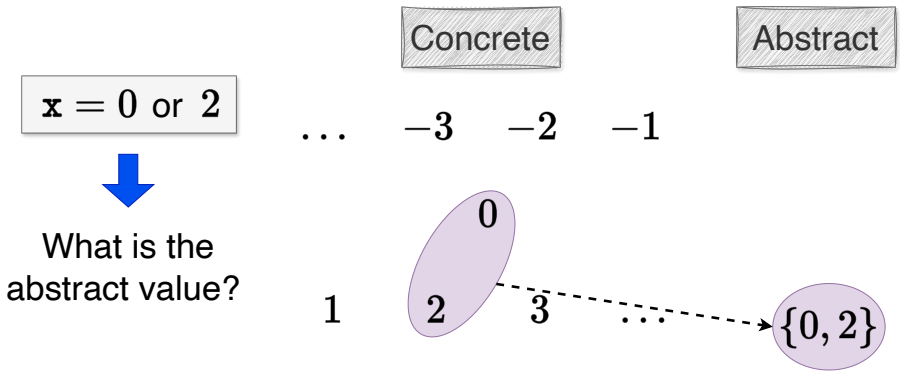
The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.

$$\mathbf{x} = 0 \text{ or } 2$$

What is the abstract value?

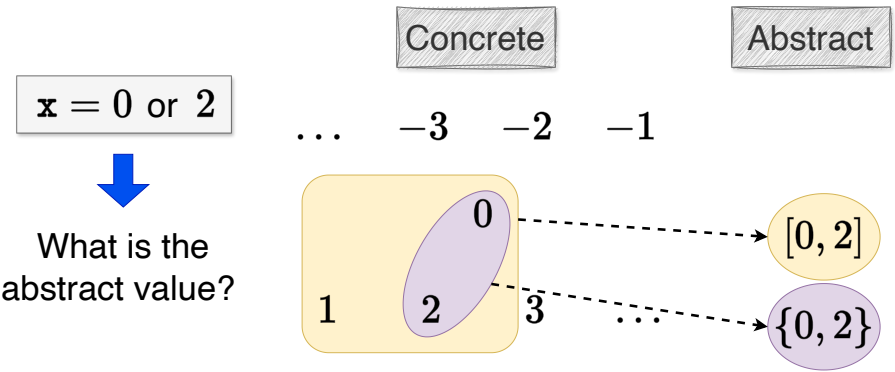
Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.



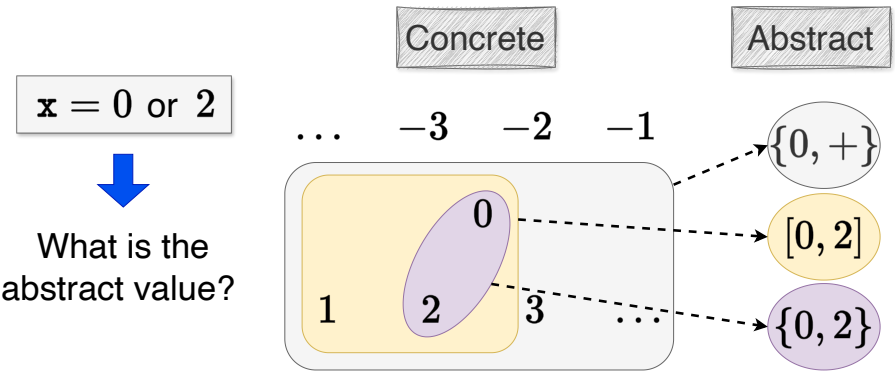
Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.



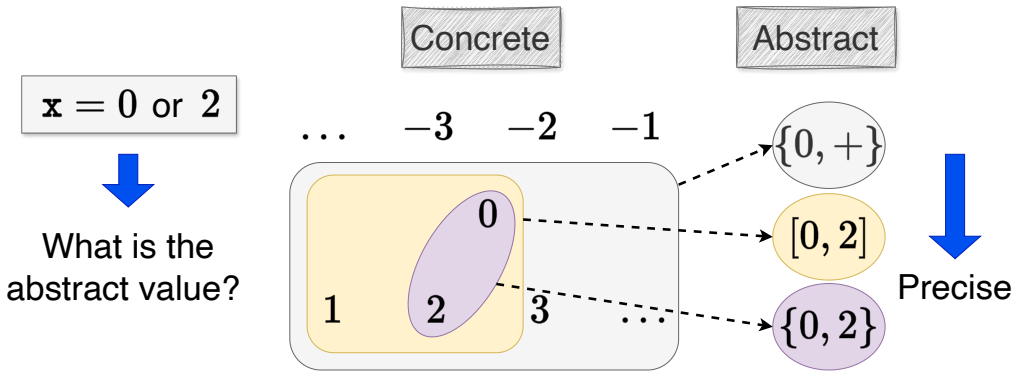
Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.



Abstract Interpretation: Levels of Abstractions

The key lies in representing potentially infinite sets of concrete states using abstract values that are compact and computationally manageable.



Abstract Interpretation: Applications

- **Program Optimization**: allows compilers to make safe assumptions about a program's behavior, leading to more efficient code generation.
  - **Range Analysis**: abstractly determines the loop's value range, aiding in memory optimization and eliminating redundant checks within this range.

Abstract Interpretation: Applications

- **Program Optimization**: allows compilers to make safe assumptions about a program's behavior, leading to more efficient code generation.
  - **Range Analysis**: abstractly determines the loop's value range, aiding in memory optimization and eliminating redundant checks within this range.
- **Hardware Design and Analysis:** used to verify that hardware designs meet certain specifications and to optimize the designs for better performance or lower power consumption.
  - **Analyzing Hardware Circuits:** By creating an abstract model of the circuit, it can predict how the circuit will behave under various input conditions.

Abstract Interpretation: Applications

- **Program Optimization**: allows compilers to make safe assumptions about a program's behavior, leading to more efficient code generation.
  - **Range Analysis**: abstractly determines the loop's value range, aiding in memory optimization and eliminating redundant checks within this range.
- **Hardware Design and Analysis:** used to verify that hardware designs meet certain specifications and to optimize the designs for better performance or lower power consumption.
  - **Analyzing Hardware Circuits:** By creating an abstract model of the circuit, it can predict how the circuit will behave under various input conditions.
- **Code Analysis (This Course)**: provides a systematic approach to approximate program behavior through value abstractions.
  - **Security Analysis:** crucial for early detection of bugs (e.g., assertion errors and buffer overflows), reducing debugging time and enhancing code reliability.

Abstract Interpretation: Tools

Widely used in safety-critical systems (e.g., aerospace industries) and commercial software products to enhance reliability, security, and performance.

Abstract Interpretation: Tools

Widely used in safety-critical systems (e.g., aerospace industries) and commercial software products to enhance reliability, security, and performance.

- **Astrée** is used to analyze and ensure the safety of software in modern aircraft, such as the Airbus A380.
- **Polyspace** is highly valued in the automotive and aerospace industries for ensuring software compliance with safety standards such as ISO 26262 for automotive software.
- **IKOS** is specialized in detecting run-time errors and numerical computation issues, making it ideal for space and aeronautics software.
- **SPARK** is used in the aerospace industry for writing and verifying safety-critical avionics software.
- **Infer** is a static analysis tool developed by Facebook to identify bugs in mobile and web applications.
- Other tools: **Frama-C, Julia static-analysis tools, BAP, Soot** and many more.

Abstract Interpretation vs. (Bounded) Symbolic Execution

Soundness

- **Abstract interpretation** aims for sound results. It can conservatively over-approximate all possible execution paths and runtime behaviors.

Abstract Interpretation vs. (Bounded) Symbolic Execution

Soundness

- **Abstract interpretation** aims for sound results. It can conservatively over-approximate all possible execution paths and runtime behaviors.
- **Symbolic execution** precisely explores feasible paths. However, bounding loop or path exploration may make it unsound, resulting in under-approximation because some feasible paths are omitted.

Assignment-2 vs. Assignment-3

Assignment-2

- **Delegate** constraint solving to the **Z3 SMT solver**.
- For each satisfiable path constraint, Z3 returns **one model with concrete values for the relevant variables**.
- Perform **per-path verification** without analyzing the inner iterations of a loop.

Assignment-3

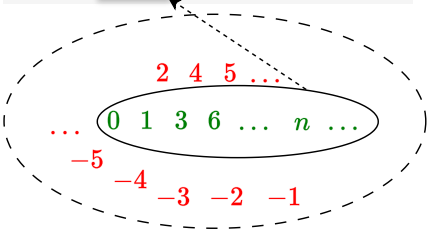
- Use **Abstract States** (`AEState`) and an **Abstract Trace** (a mapping from ICFG nodes to `AEState`s) to compute and maintain **abstract values** of variables.
- Represent the possible values of a variable using an **interval** for scalars or an **address set** for memory addresses.
- Perform verification with **path merging**, over-approximating loop behaviours using widening and narrowing.

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?

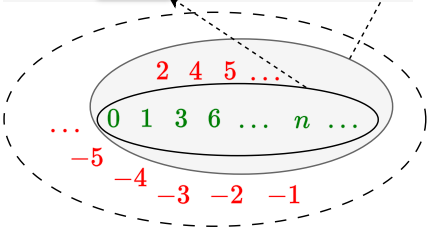


Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation

$\{0, +\}$

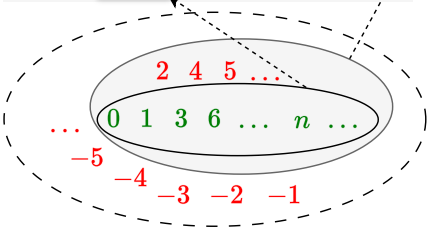
Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation

$\{0, +\}$

Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

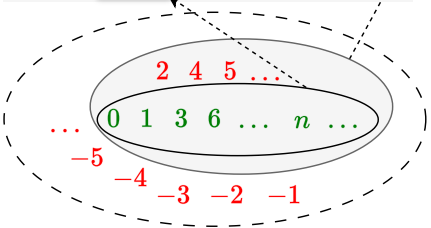
Path	Answer
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_6$	0

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation

$\{0, +\}$

Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

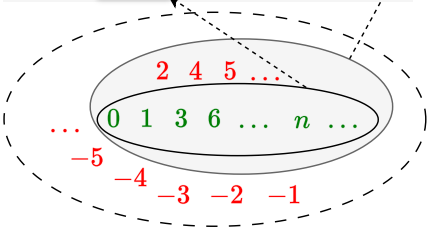
Path	Answer
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_6$	0
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	1

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation

$\{0, +\}$

Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

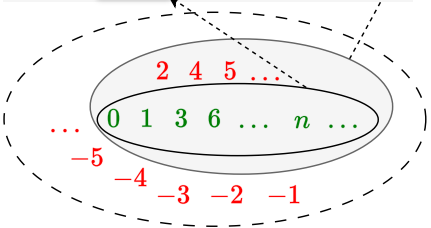
Path	Answer
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_6$	0
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	1
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	3

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation

$\{0, +\}$

Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

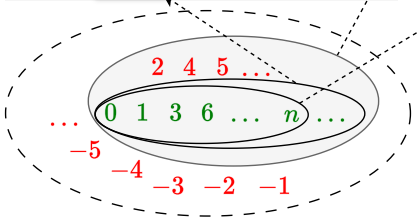
Path	Answer
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_6$	0
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	1
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	3
..... infinite paths!	...

Abstract Interpretation vs. (Bounded) Symbolic Execution

Over-Approximation vs. Under-Approximation

```
ℓ1 void analyzeThis(int x) {
ℓ2     int sum = 0;
ℓ3     for (int i = 0; i < x; ++i) {
ℓ4         sum += i;
ℓ5     }
ℓ6 }
```

sum = ?



Abstract Interpretation $\{0, +\}$

Sound (include all non-negative numbers)
imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

Precise (only include feasible numbers: 0, 1, 3, 6, ...)
unsound (cannot cover all possible numbers)

Path	Answer
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_6$	0
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	1
$\ell_1 \to \ell_2 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_3 \to \ell_4 \to \ell_5 \to \ell_6$	3
..... infinite paths!	...

Note: symbolic execution is exact for explored feasible paths; it becomes unsound only when exploration is bounded.

Importance of Soundness

- **Reliability:** Ensures comprehensive coverage of all possible program states, reducing unforeseen behavior in production.

Importance of Soundness

- **Reliability:** Ensures comprehensive coverage of all possible program states, reducing unforeseen behavior in production.
- **Quality Assurance:** Crucial for critical systems where failure can have serious consequences, ensuring software behaves as intended.

Importance of Soundness

- **Reliability:** Ensures comprehensive coverage of all possible program states, reducing unforeseen behavior in production.
- **Quality Assurance:** Crucial for critical systems where failure can have serious consequences, ensuring software behaves as intended.
- **Confidence in Maintenance:** Provides a safety net for code changes, reducing the risk of introducing new bugs.

Abstract Interpretation vs. (Bounded) Symbolic Execution

Termination

- **Abstract interpretation** terminates directly on finite-height domains. For infinite-height domains, widening is commonly used to force convergence after finitely many iterations.

Abstract Interpretation vs. (Bounded) Symbolic Execution

Termination

- **Abstract interpretation** terminates directly on finite-height domains. For infinite-height domains, widening is commonly used to force convergence after finitely many iterations.
- **Symbolic execution** may struggle with termination in complex or large-scale programs. The need to explore numerous paths in detail, especially in programs with loops and recursive calls, can lead to non-termination or impractical analysis times.

Importance of Termination

- **Deterministic:** Ensures consistent outcomes and predictable resource use for the same input.

Importance of Termination

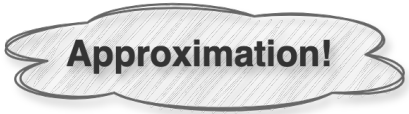
- **Deterministic:** Ensures consistent outcomes and predictable resource use for the same input.
- **Efficiency:** Reduces computational load by using abstracted state spaces, speeding up the analysis process.

Importance of Termination

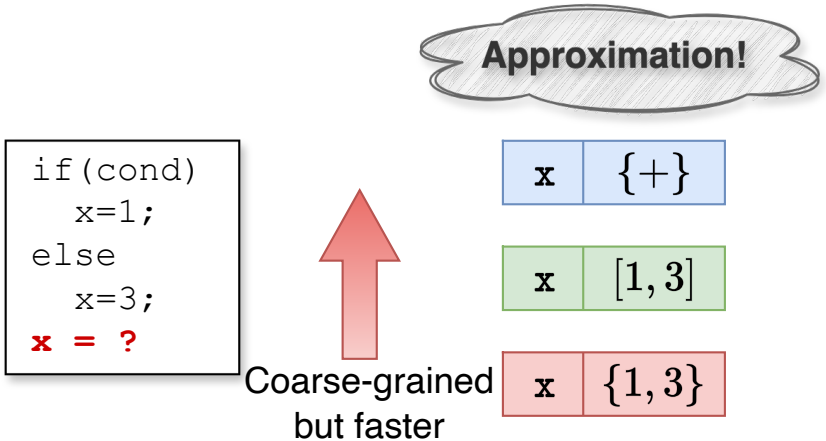
- **Deterministic:** Ensures consistent outcomes and predictable resource use for the same input.
- **Efficiency:** Reduces computational load by using abstracted state spaces, speeding up the analysis process.
- **Coverage:** Ensures that all parts of the code are analyzed, avoiding missed sections and ensuring thorough coverage for detecting issues.

Abstract Interpretation: A Code Example

```
if(cond)
   x=1;
else
   x=3;
x = ?
```

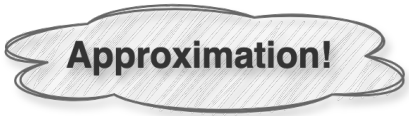


Abstract Interpretation: A Code Example



Abstract Interpretation: A Code Example

```
if(cond)
   x=1;
else
   x=3;
x = ?
```



Concrete Domain and Abstract Domain: Formal Definition

Concrete Domain

- $\mathbb{S}$ denotes the set of concrete values that a program variable can have.
  - E.g., $\mathbb{S} = \mathbb{Z}$ represents the concrete values that an integer variable can have.
- A **concrete domain** $\mathbb{C}$ is the *powerset* of $\mathbb{S}$, denoted as $\mathbb{C} = \mathcal{P}(\mathbb{S})$.
  - E.g. The *powerset integer domain* is a concrete domain for integer variables.

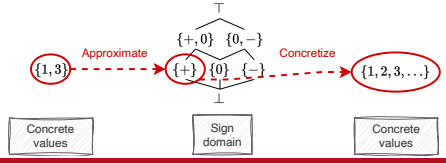
Abstract Domain

- An **abstract domain** $\mathbb{A}$ contains *abstract values* approximating a set of concrete values.
- An abstract domain is typically implemented using a **lattice** $\mathbb{L} = \langle \mathbb{A}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$ structure, a set of abstract values following a **partial order**, also equipped with two binary operations.
  - $\sqsubseteq$ is a partial order relation on $\mathbb{A}$ (e.g., $\sqsubseteq$ is the subset ($\subseteq$) on a power set).
  - $\sqcap$ and $\sqcup$ are the meet and join binary operations, and $\bot$ and $\top$ are unique least and greatest elements of $\mathbb{A}$.

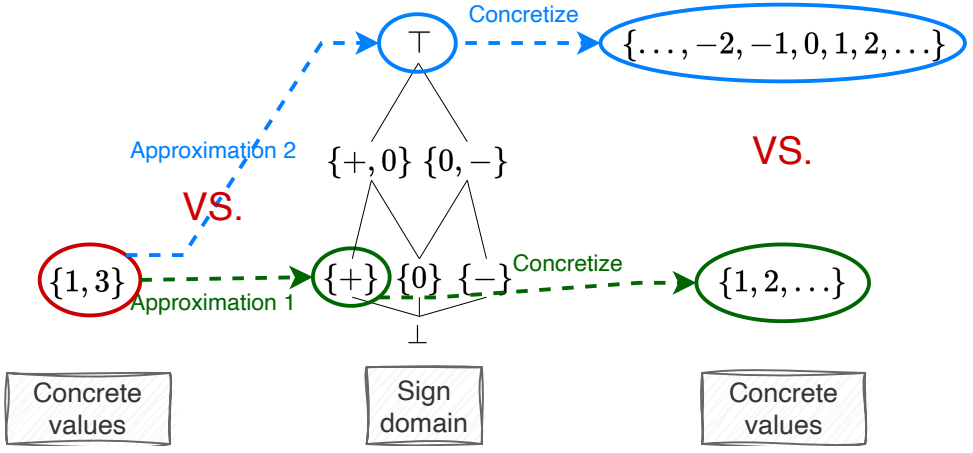
An Example: Abstract Sign Domain

An abstract domain that approximates a set of concrete values with their signs.

- Lattice is defined as $\mathbb{L} = \langle \mathcal{P}(\{-, 0, +\}), \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$.
- **Partial order**: $a \sqsubseteq b \Leftrightarrow a \subseteq b$. E.g., $\{+\} \sqsubseteq \{0, +\} \Leftrightarrow \{+\} \subseteq \{0, +\}$.
- **Meet operator** $a \sqcap b$: returns the **greatest lower bound (GLB)** that is less than or equal to both a and b (**move downwards along the lattice**)
  - $\{+\} \sqcap \{0\} = \bot$
- **Join operator** $a \sqcup b$: returns the **least upper bound (LUB)** that is greater than or equal to both a and b (**move upwards along the lattice**)
  - $\{+\} \sqcup \{0\} = \{+, 0\}$
- **Approximation**: concrete value set $\{1, 3\}$ is **over-approximated** as $\{+\}$. After **concretization**, it is restored as $\{x \in \mathbb{Z} | x > 0\}$, a **super set** of $\{1, 3\}$.



An Example, the Best Abstraction using Sign Domain



Approximation 1 (more precise than Approximation 2) is the best abstraction!

Galois Connection

A **Galois connection** provides a consistent bridge between the concrete and abstract domains: abstraction maps concrete properties to their **most precise** abstract representation, while concretization explains which concrete properties an abstract value represents.

- Abstraction $\alpha : \mathbb{C} \to \mathbb{A}$ maps a set of concrete values to an abstract value.
- Concretization $\gamma : \mathbb{A} \to \mathbb{C}$ maps an abstract value to a set of concrete values.

Galois Connection

A **Galois connection** provides a consistent bridge between the concrete and abstract domains: abstraction maps concrete properties to their **most precise** abstract representation, while concretization explains which concrete properties an abstract value represents.

- Abstraction $\alpha : \mathbb{C} \to \mathbb{A}$ maps a set of concrete values to an abstract value.
- Concretization $\gamma : \mathbb{A} \to \mathbb{C}$ maps an abstract value to a set of concrete values.

Example: Sign domain (Abstraction, Concretization and Galois Connection)

$$\alpha_{Sign}(S) = \{s \in \{-, 0, +\} \mid \exists x \in S : \text{sign}(x) = s\},$$
$$\alpha_{Sign}(\{-2, 0, 3\}) = \{-, 0, +\},$$
$$\alpha_{Sign}(\{1, 3\}) = \{+\}.$$

$$\gamma_{Sign}(a) = \bigcup_{s \in a} \gamma_{Sign}(\{s\}),$$
$$\gamma_{Sign}(\{-\}) = \{x \in \mathbb{Z} \mid x < 0\},$$
$$\gamma_{Sign}(\{0, +\}) = \{x \in \mathbb{Z} \mid x \geq 0\},$$
$$\gamma_{Sign}(\top) = \mathbb{Z}.$$

$$\boxed{\alpha(S) \sqsubseteq a \iff S \subseteq \gamma(a)}$$

Thus, $\alpha(S)$ is the **most precise abstract value** that safely represents all concrete values in S.

Best Abstraction vs. Practical Abstract Interpreter

Best abstraction under a Galois connection

For (α, γ),

$$\alpha(c) \sqsubseteq a \iff c \sqsubseteq \gamma(a).$$

Hence, $\alpha(c)$ is the **most precise value in** $\mathbb{A}$ that soundly represents c.

Best Abstraction vs. Practical Abstract Interpreter

Best abstraction under a Galois connection

For (α, γ),

$$\alpha(c) \sqsubseteq a \iff c \sqsubseteq \gamma(a).$$

Hence, $\alpha(c)$ is the **most precise value in** $\mathbb{A}$ that soundly represents c.

Practical abstract interpretation

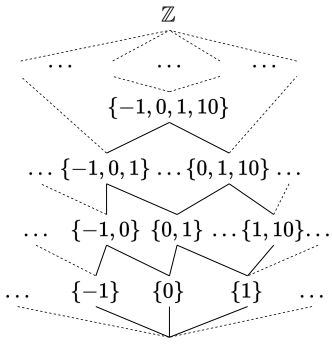
Let $\mathsf{lfp}(F)$ denote the **least fixed point** of the concrete transfer function F, representing the reachable concrete states. An abstract interpreter may compute $a^\sharp$ such that

$$\alpha(\mathsf{lfp}(F)) \sqsubseteq a^\sharp.$$

Thus, $a^\sharp$ is sound but may be less precise due to approximate transfer functions, joins, or widening.

Galois connection: best abstraction in the chosen domain.
Analysis algorithm: best abstraction or possibly less precise yet sound result.

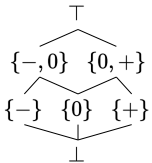
Concrete Domain vs. Abstract Sign Domain



(a) Powerset integer domain

$$\gamma_{Sign}(\top) = \mathbb{Z}$$
$$\gamma_{Sign}(\{-\}) = \{x \mid x < 0\}$$
$$\gamma_{Sign}(\{+\}) = \{x \mid x > 0\}$$
$$\gamma_{Sign}(\{-,0\}) = \{x \mid x \leq 0\}$$
$$\gamma_{Sign}(\{+,0\}) = \{x \mid x \geq 0\}$$
$$\gamma_{Sign}(\{\bot\}) = \varnothing$$

$$\alpha_{Sign}(c) = \begin{cases} \{+\} \ if\ c \in \mathbb{Z}_{>0} \\ \{-\} \ if\ c \in \mathbb{Z}_{<0} \\ \{+,0\} \ if\ c \in \mathbb{Z}_{\geq 0} \\ \{-,0\} \ if\ c \in \mathbb{Z}_{\leq 0} \\ \bot \ \ if\ c = \varnothing \\ \top \ otherwise \end{cases}$$



(b) Sign domain

Interval Domain, Meet ($\sqcap$) and Join ($\sqcup$) Operators

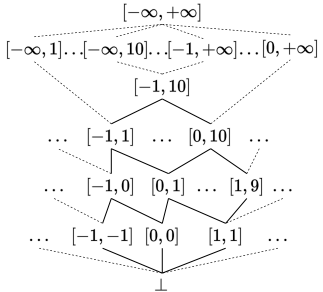
The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\mathbb{I} = \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\},\ a \leq b\}$.
- Partial order: $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$.
  - E.g., $[0, 0], [0, 1] \in \mathbb{A}_{interval}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.

Interval Domain, Meet ($\sqcap$) and Join ($\sqcup$) Operators

The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\mathbb{I} = \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\},\ a \leq b\}$.
- Partial order: $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$.
  - E.g., $[0, 0], [0, 1] \in \mathbb{A}_{interval}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.



(b) Interval domain

Given $a_1 = [3, 8]$ and $a_2 = [7, 12]$.

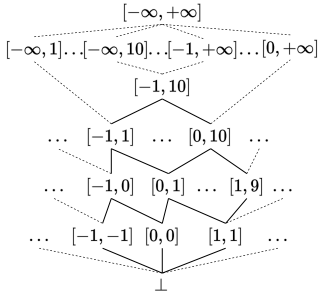
Meet operation $a_1 \sqcap a_2$ returns the **greatest Lower Bound** (GLB):

- The largest range shared by both a_1 and a_2.

Interval Domain, Meet ($\sqcap$) and Join ($\sqcup$) Operators

The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\mathbb{I} = \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\},\ a \leq b\}$.
- Partial order: $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$.
  - E.g., $[0, 0], [0, 1] \in \mathbb{A}_{interval}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.



(b) Interval domain

Given $a_1 = [3, 8]$ and $a_2 = [7, 12]$.

Meet operation $a_1 \sqcap a_2$ returns the **greatest Lower Bound** (GLB):

- The largest range shared by both a_1 and a_2. **GLB = [7, 8]**

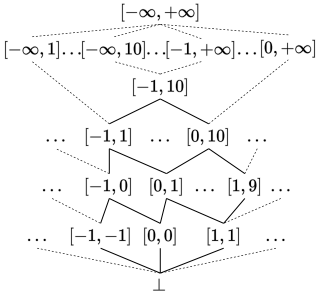
Join operation $a_1 \sqcup a_2$ returns the **Least Upper Bound** (LUB):

- The smallest range includes both a_1 and a_2.

Interval Domain, Meet ($\sqcap$) and Join ($\sqcup$) Operators

The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\mathbb{I} = \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\},\ a \leq b\}$.
- Partial order: $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$.
  - E.g., $[0, 0], [0, 1] \in \mathbb{A}_{interval}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.



(b) Interval domain

Given $a_1 = [3, 8]$ and $a_2 = [7, 12]$.

Meet operation $a_1 \sqcap a_2$ returns the **greatest Lower Bound** (GLB):

- The largest range shared by both a_1 and a_2. **GLB = [7, 8]**

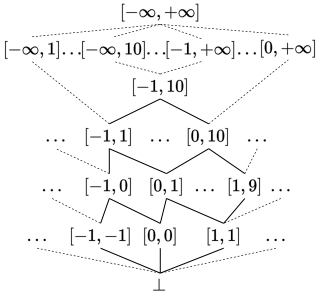
Join operation $a_1 \sqcup a_2$ returns the **Least Upper Bound** (LUB):

- The smallest range includes both a_1 and a_2. **LUB = [3,12]**

Interval Domain, Meet ($\sqcap$) and Join ($\sqcup$) Operators

The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \bot, \top \rangle$, where $\mathbb{I} = \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\},\ a \leq b\}$.
- Partial order: $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$.
  - E.g., $[0, 0], [0, 1] \in \mathbb{A}_{interval}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.



(b) Interval domain

Given $a_1 = [3, 8]$ and $a_2 = [7, 12]$.

Meet operation $a_1 \sqcap a_2$ returns the **greatest Lower Bound** (GLB):

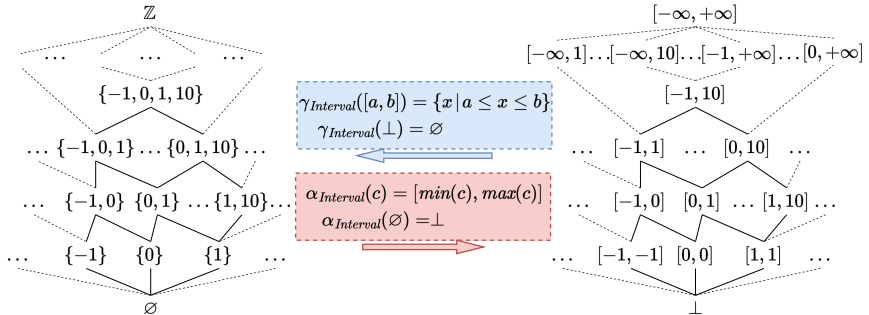
- The largest range shared by both a_1 and a_2. **GLB = [7, 8]**

Join operation $a_1 \sqcup a_2$ returns the **Least Upper Bound** (LUB):

- The smallest range includes both a_1 and a_2. **LUB = [3,12]**

The greatest and least elements of $\mathbb{L}_{interval}$ are $\top = [-\infty, +\infty]$ and $\bot$, respectively.

Galois Connection between $\mathbb{C}$ and $\mathbb{A}_{interval}$



(a) Powerset integer domain (b) Interval domain

Figure: Powerset integer domain $\mathbb{C}$ and its abstraction as the interval domain $\mathbb{A}_{interval}$.

Abstract State and Abstract Trace

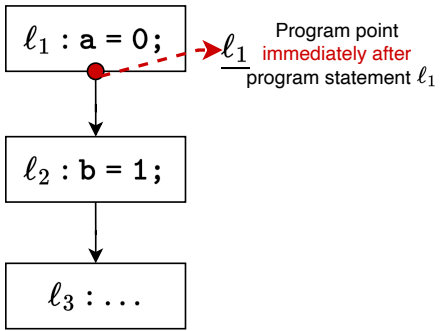
- An **abstract state** (`AEState` in Lab-3 and Assignment-3) is defined as a map $AS : \mathcal{V} \to \mathbb{A}$ associating program variables $\mathcal{V}$ with an abstract value in $\mathbb{A}$, approximating the runtime states of program variables.

Abstract State and Abstract Trace

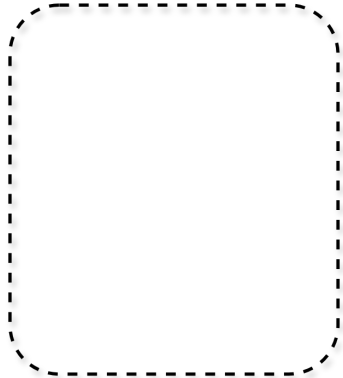
- An **abstract state** (`AEState` in Lab-3 and Assignment-3) is defined as a map $AS : \mathcal{V} \to \mathbb{A}$ associating program variables $\mathcal{V}$ with an abstract value in $\mathbb{A}$, approximating the runtime states of program variables.
- An **abstract trace** is a mapping from program points to abstract states, $\sigma : \mathbb{L} \to (\mathcal{V} \to \mathbb{A})$. In Assignment-3, `preAbsTrace` and `postAbsTrace` maintain the abstract states before ($\overline{\ell}$) and after ($\underline{\ell}$) each program statement ℓ.

	Notation	Domain
Abstract trace	σ	$\mathbb{L} \to (\mathcal{V} \to \mathbb{A}_{Interval})$
Abstract state at program point $L \in \mathbb{L}$	σ_L	$\mathcal{V} \to \mathbb{A}_{Interval}$
Abstract value of x **at program point** $L \in \mathbb{L}$	$\sigma_L(x)$	$\mathbb{A}_{Interval}$

Abstract Trace : A Simple Example

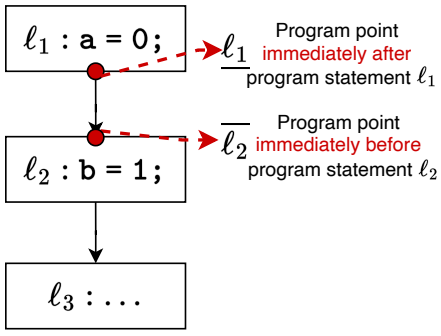


Control Flow Graph



Abstract Trace σ

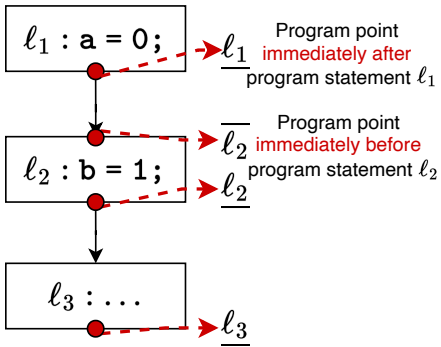
Abstract Trace : A Simple Example



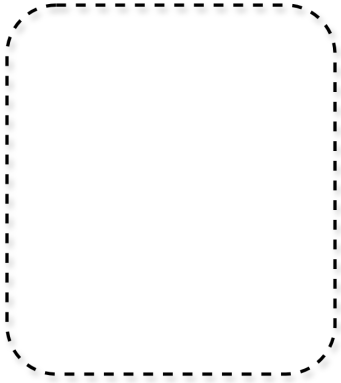
Control Flow Graph

Abstract Trace σ

Abstract Trace : A Simple Example

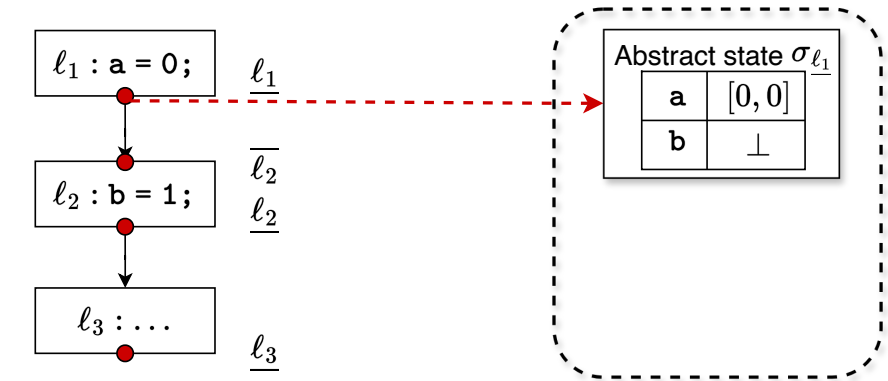


Control Flow Graph

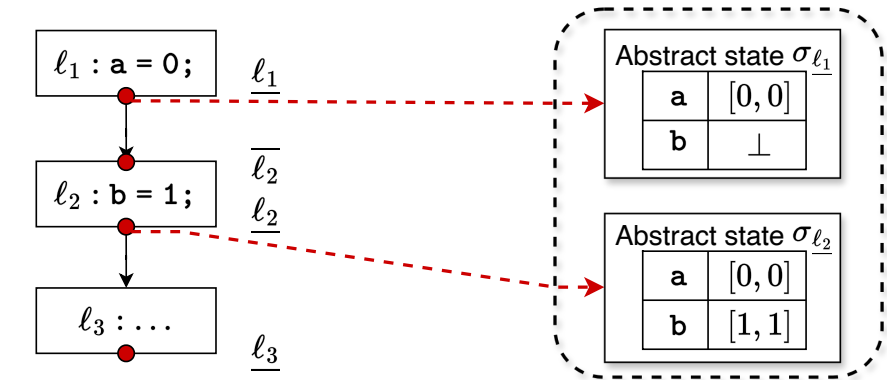


Abstract Trace σ

Abstract Trace : A Simple Example

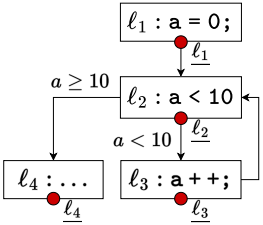


Abstract Trace : A Simple Example



Abstract Trace: Naive Fixed-Point Computation for Loops

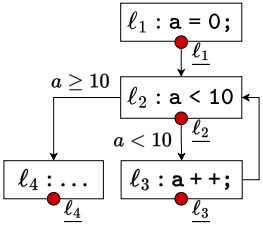
Abstract trace												
$\sigma_{\ell_1}(a)$												
$\sigma_{\ell_2}(a)$												
$\sigma_{\ell_3}(a)$												
$\sigma_{\ell_4}(a)$												



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace												
$\sigma_{\ell_1}(a)$												
$\sigma_{\ell_2}(a)$												
$\sigma_{\ell_3}(a)$												
$\sigma_{\ell_4}(a)$												

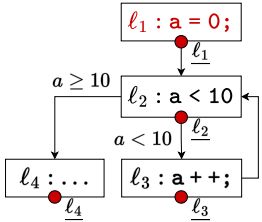


Control Flow Graph

What is the abstract state after analyzing each statement?

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace												
$\sigma_{\underline{\ell_1}}(a)$												
$\sigma_{\underline{\ell_2}}(a)$												
$\sigma_{\underline{\ell_3}}(a)$												
$\sigma_{\underline{\ell_4}}(a)$												



Control Flow Graph

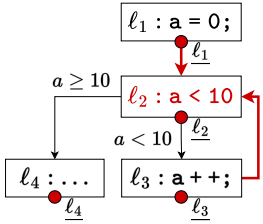
What is the abstract state after analyzing each statement?

$\sigma_{\underline{\ell_1}}(a) := F_1() = [0, 0]$

$F_1, \ldots, F_4$ are t**ransfer functions** which indicate how abstract states are updated

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace												
$\sigma_{\underline{\ell_1}}(a)$												
$\sigma_{\underline{\ell_2}}(a)$												
$\sigma_{\underline{\ell_3}}(a)$												
$\sigma_{\underline{\ell_4}}(a)$												



Control Flow Graph

What is the abstract state after analyzing each statement?

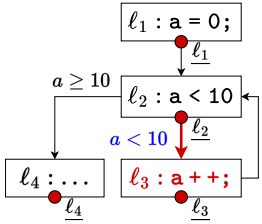
$$\sigma_{\underline{\ell_1}}(a) := F_1() = [0, 0]$$

$$\sigma_{\underline{\ell_2}}(a) := F_2(\sigma_{\underline{\ell_1}}, \sigma_{\underline{\ell_3}}) = \sigma_{\underline{\ell_1}}(a) \sqcup \sigma_{\underline{\ell_3}}(a)$$

$F_1, \ldots, F_4$ are t**ransfer functions** which indicate how abstract states are updated

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace												
$\sigma_{\underline{\ell_1}}(a)$												
$\sigma_{\underline{\ell_2}}(a)$												
$\sigma_{\underline{\ell_3}}(a)$												
$\sigma_{\underline{\ell_4}}(a)$												



Control Flow Graph

What is the abstract state after analyzing each statement?

$$\sigma_{\underline{\ell_1}}(a) := F_1() = [0, 0]$$

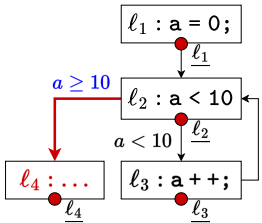
$$\sigma_{\underline{\ell_2}}(a) := F_2(\sigma_{\underline{\ell_1}}, \sigma_{\underline{\ell_3}}) = \sigma_{\underline{\ell_1}}(a) \sqcup \sigma_{\underline{\ell_3}}(a)$$

$$\sigma_{\underline{\ell_3}}(a) := F_3(\sigma_{\underline{\ell_2}}) = ([-\infty, 9] \sqcap \sigma_{\underline{\ell_2}}(a)) + [1, 1]$$

$F_1, \ldots, F_4$ are **transfer functions** which indicate how abstract states are updated

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace												
$\sigma_{\underline{\ell_1}}(a)$												
$\sigma_{\underline{\ell_2}}(a)$												
$\sigma_{\underline{\ell_3}}(a)$												
$\sigma_{\underline{\ell_4}}(a)$												



Control Flow Graph

What is the abstract state after analyzing each statement?

$$\sigma_{\underline{\ell_1}}(a) := F_1() = [0,0]$$

$$\sigma_{\underline{\ell_2}}(a) := F_2(\sigma_{\underline{\ell_1}}, \sigma_{\underline{\ell_3}}) = \sigma_{\underline{\ell_1}}(a) \sqcup \sigma_{\underline{\ell_3}}(a)$$

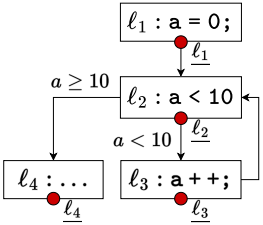
$$\sigma_{\underline{\ell_3}}(a) := F_3(\sigma_{\underline{\ell_2}}) = ([-\infty, 9] \sqcap \sigma_{\underline{\ell_2}}(a)) + [1,1]$$

$$\sigma_{\underline{\ell_4}}(a) := F_4(\sigma_{\underline{\ell_2}}) = ([10, \infty] \sqcap \sigma_{\underline{\ell_2}}(a))$$

$F_1, \ldots, F_4$ are t**ransfer functions** which indicate how abstract states are updated

Abstract Trace: Naive Fixed-Point Computation for Loops

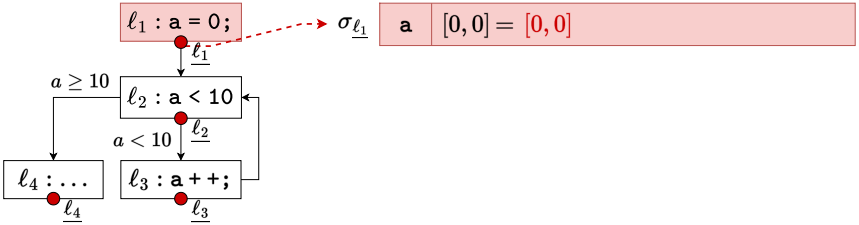
Abstract trace	Init											
$\sigma_{\ell_1}(a)$	$\bot$											
$\sigma_{\ell_2}(a)$	$\bot$											
$\sigma_{\ell_3}(a)$	$\bot$											
$\sigma_{\ell_4}(a)$	$\bot$											



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

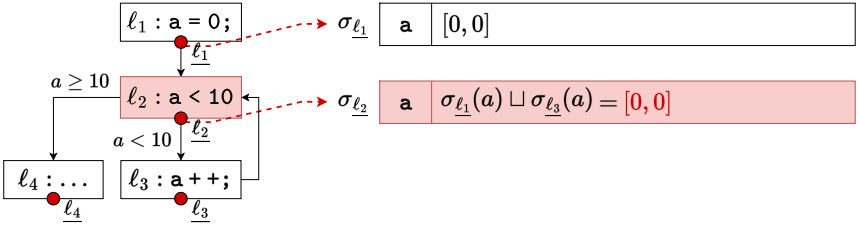
Abstract trace	Init	After analyzing ℓ_1								
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$								
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$								
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$								
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$								



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

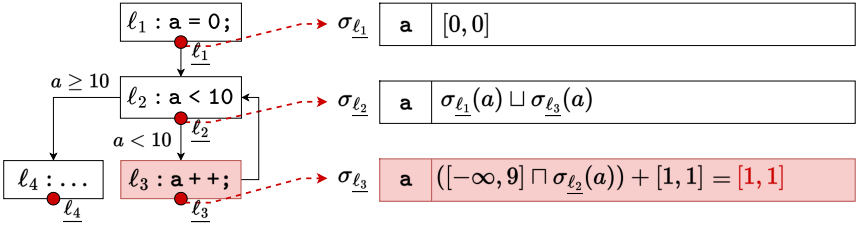
Abstract trace	Init	After analyzing ℓ_1	1st loop iter: After ℓ_2									
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$									
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$									
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$									
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$									



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

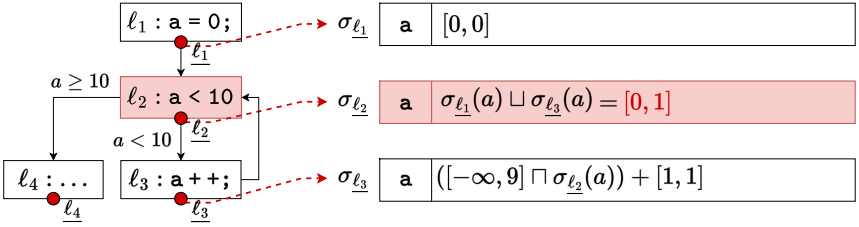
Abstract trace	Init	After analyzing ℓ_1	1st loop iter								
			After ℓ_2	After ℓ_3							
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$							
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$							
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$							
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$							



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

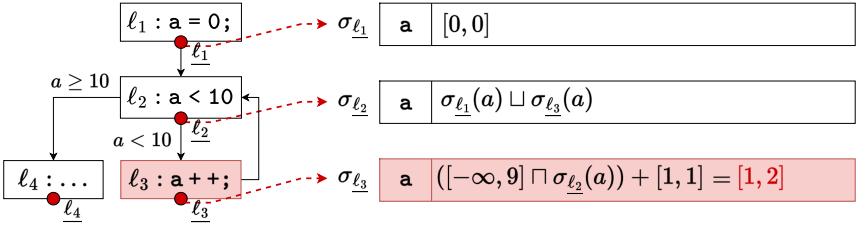
Abstract trace	Init	After analyzing ℓ_1	1st loop iter: After ℓ_2	1st loop iter: After ℓ_3	2nd loop iter: After ℓ_2							
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$							
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$							
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$							
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$							



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

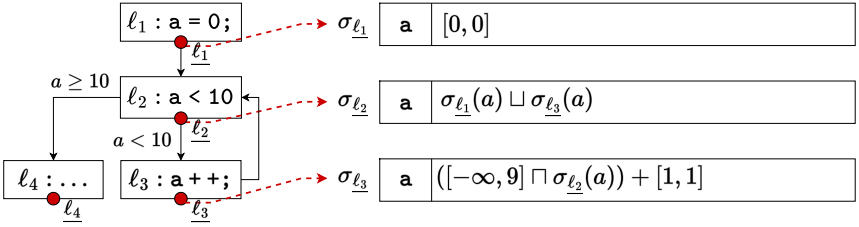
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter							
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3						
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$						
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$						
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$						
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$						



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

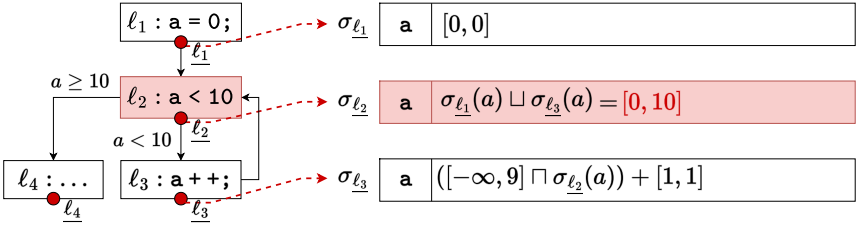
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3			
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$			
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$			
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$			
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$			



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

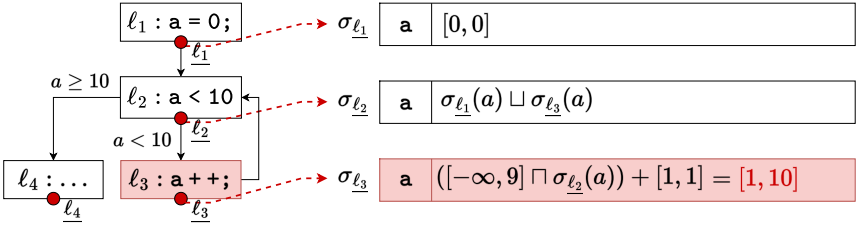
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter		12nd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2		
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$		
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$		
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$		
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$		



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter		12nd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$	

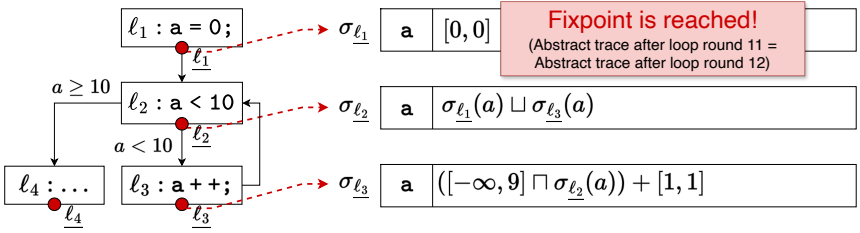


Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter		12nd loop iter	
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$

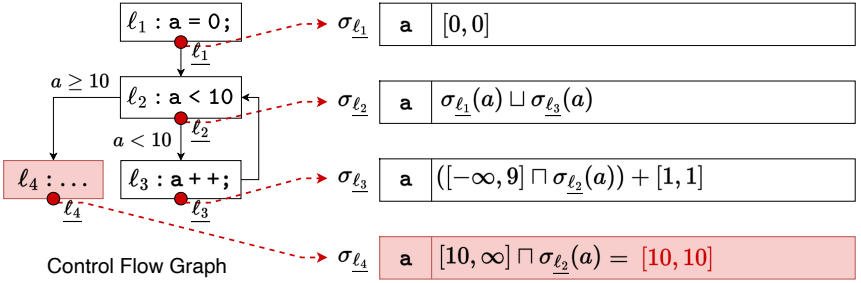
Fixpoint is reached!
(Abstract trace after loop round 11 = Abstract trace after loop round 12)



Control Flow Graph

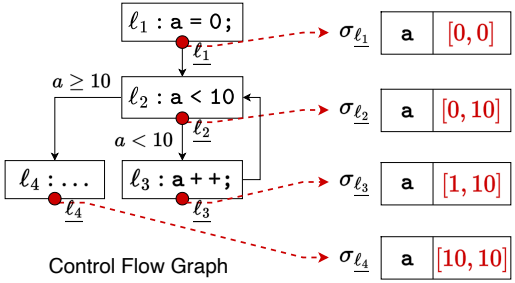
Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter		12nd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



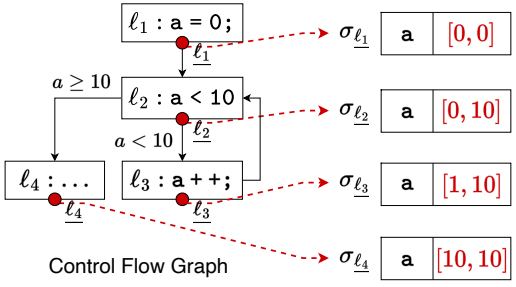
Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		...	11st loop iter		12nd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter			11st loop iter		12nd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$

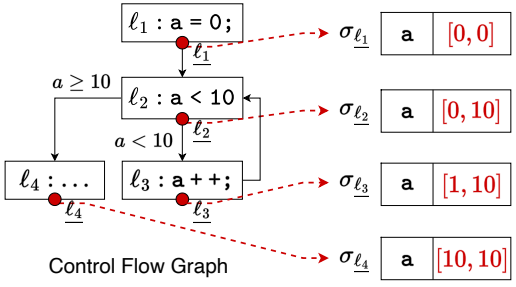


$[0,0] \Rightarrow [0,1] \Rightarrow [0,2] \Rightarrow \ldots \Rightarrow [0,10] \Rightarrow [0,10]$

12 iterations while analyzing the loop

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter			11st loop iter		12nd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	...	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,1]$	$[0,1]$	...	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,2]$	...	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	...	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



$[0,0] \Rightarrow [0,1] \Rightarrow [0,2] \Rightarrow \ldots \Rightarrow [0,10] \Rightarrow [0,10]$

12 iterations while analyzing the loop

What if $a < 10000$?
More iterations!

Widening: Accelerating Fixed-Point Computation

Widening technique can accelerate the fixpoint computation of $\sigma_{\underline{\ell_2}}(a)$.

Naive fixpoint computation: value changes of $\sigma_{\underline{\ell_2}}(a)$

$$[0,0] \Rightarrow [0,1] \Rightarrow \ldots \Rightarrow [0,10] \Rightarrow [0,10]$$

Widening: Accelerating Fixed-Point Computation

Widening technique can accelerate the fixpoint computation of $\sigma_{\underline{\ell_2}}(a)$.

Naive fixpoint computation: value changes of $\sigma_{\underline{\ell_2}}(a)$

$$[0,0] \Rightarrow [0,1] \Rightarrow \ldots \Rightarrow [0,10] \Rightarrow [0,10]$$

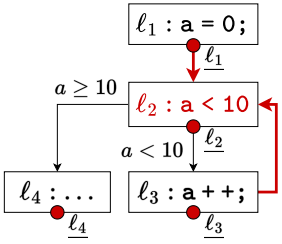
Widening

aggressively update $\sigma_{\underline{\ell_2}}(a)$

$$[0,+\infty]$$

Widening: Accelerating Fixed-Point Computation

Widening at the k^{th} iteration in the loop for analyzing ℓ_2 to update $\sigma_{\underline{\ell_2}}$.



Control Flow Graph

$$\sigma_{\underline{\ell_2}}(a) := \sigma_{\underline{\ell_1}}(a) \sqcup \sigma_{\underline{\ell_3}}(a)$$

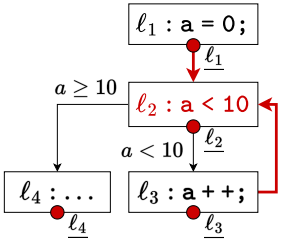
Apply widening operator ∇

$$\sigma^k_{\underline{\ell_2}}(a) := \sigma^{k-1}_{\underline{\ell_2}}(a) \nabla (\sigma_{\underline{\ell_1}}(a) \sqcup \sigma^{k-1}_{\underline{\ell_3}}(a))$$

$\sigma^k_{\underline{\ell_2}}$ denotes the value of $\sigma_{\underline{\ell_2}}$ after the k^{th} analysis of ℓ_2, and $\sigma_{\underline{\ell_1}}$ does not have a superscription as it is updated only once and is not involved in the loop

Widening: Accelerating Fixed-Point Computation

Widening at the k^{th} iteration in the loop for analyzing ℓ_2 to update $\sigma_{\underline{\ell_2}}$.



Control Flow Graph

$$\sigma_{\underline{\ell_2}}(a) := \sigma_{\underline{\ell_1}}(a) \sqcup \sigma_{\underline{\ell_3}}(a)$$

Apply widening operator ∇

$$\sigma^k_{\underline{\ell_2}}(a) := \sigma^{k-1}_{\underline{\ell_2}}(a) \nabla (\sigma_{\underline{\ell_1}}(a) \sqcup \sigma^{k-1}_{\underline{\ell_3}}(a))$$

$\sigma^k_{\underline{\ell_2}}$ denotes the value of $\sigma_{\underline{\ell_2}}$ after the k^{th} analysis of ℓ_2, and $\sigma_{\underline{\ell_1}}$ does not have a superscription as it is updated only once and is not involved in the loop

What is a **Widening Operator**?

Widening Operator

The Widening Operator ($\nabla : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$) is formally defined on a poset $(\mathbb{A}, \sqsubseteq)$. ∇ on interval domain could be defined as:

$$[\ell_1, h_1] \nabla [\ell_2, h_2] = [\ell_3, h_3]$$

Widening Operator

The Widening Operator ($\triangledown : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$) is formally defined on a poset $(\mathbb{A}, \sqsubseteq)$. ∇ on interval domain could be defined as:

$$[\ell_1, h_1] \nabla [\ell_2, h_2] = [\ell_3, h_3]$$

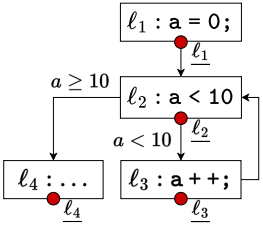
where

$$l_3 = \begin{cases} -\infty & l_2 < l_1 \\ l_1 & l_2 \geq l_1 \end{cases}, h_3 = \begin{cases} +\infty & h_2 > h_1 \\ h_1 & h_2 \leq h_1 \end{cases}$$

As a concrete example, $[0, 0] \nabla [0, 1] = [0, +\infty]$.

Widening: The Loop Example

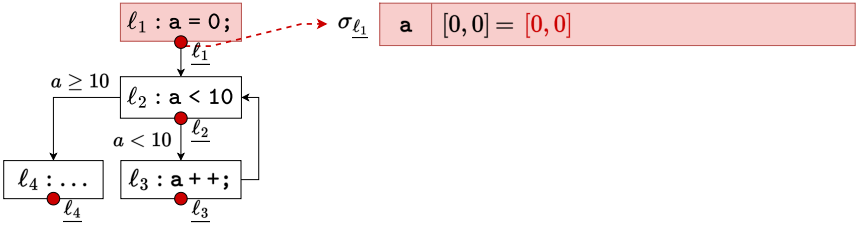
Abstract trace	Init								
$\sigma_{\ell_1}(a)$	$\bot$								
$\sigma_{\ell_2}(a)$	$\bot$								
$\sigma_{\ell_3}(a)$	$\bot$								
$\sigma_{\ell_4}(a)$	$\bot$								



Control Flow Graph

Widening: The Loop Example

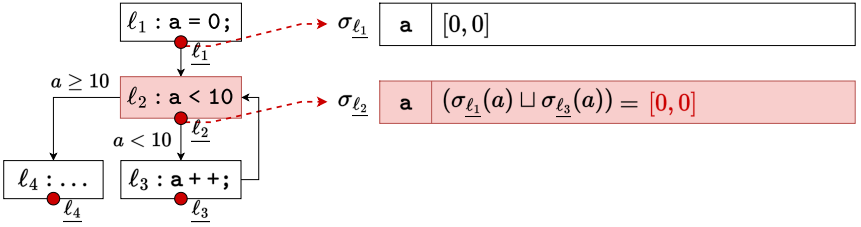
Abstract trace	Init	After analyzing ℓ_1						
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$						
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$						
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$						
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$						



Control Flow Graph

Widening: The Loop Example

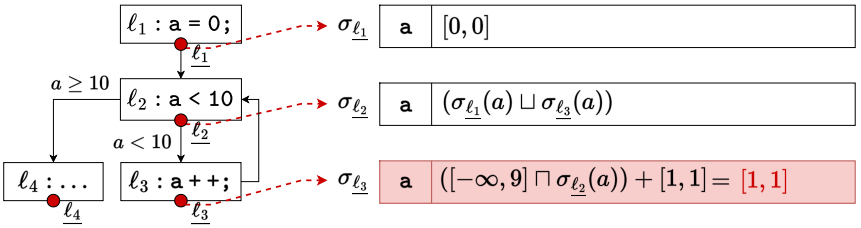
Abstract trace	Init	After analyzing ℓ_1	1st loop iter: After ℓ_2						
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$						
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$						
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$						
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$						



Control Flow Graph

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1^{st} loop iter						
			After ℓ_2	After ℓ_3					
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$					
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$					
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$					
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$					

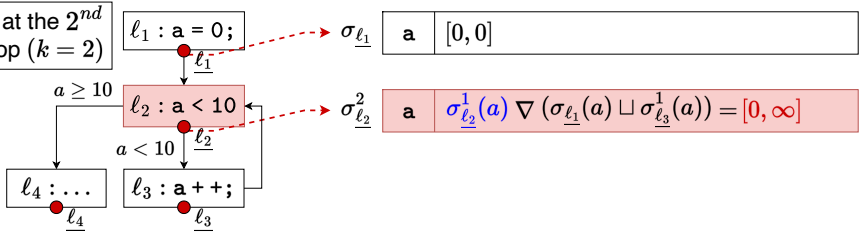


Control Flow Graph

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2				
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$				
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$				
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$				
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$				

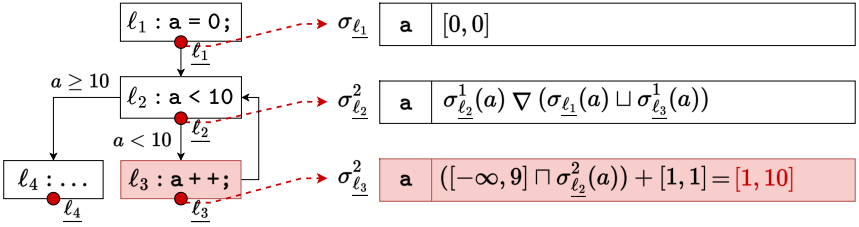
Start widening at the 2^{nd} iteration of loop ($k = 2$)



Control Flow Graph

Widening: The Loop Example

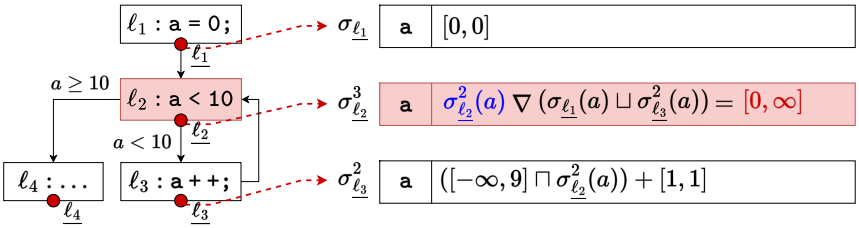
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3			
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$			
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$			
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$			
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$			



Control Flow Graph

Widening: The Loop Example

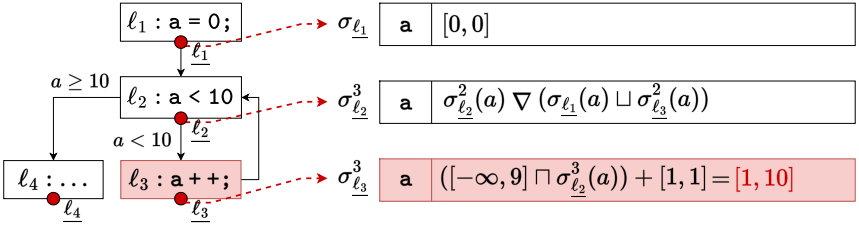
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2		
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$		
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$		
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$		
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$		



Control Flow Graph

Widening: The Loop Example

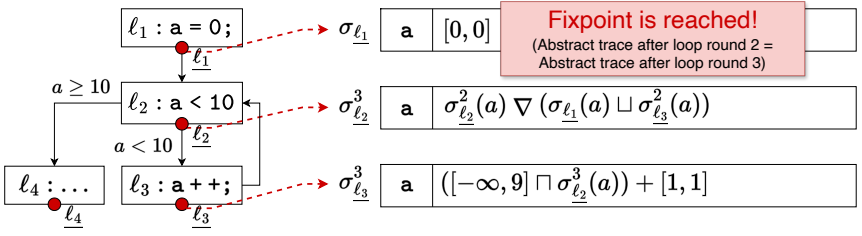
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	



Control Flow Graph

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter	
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

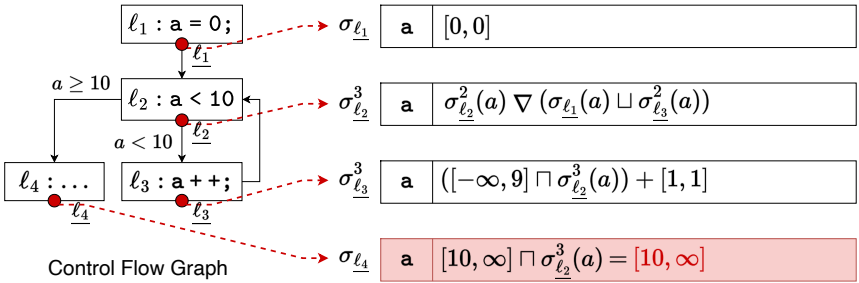


Fixpoint is reached!
(Abstract trace after loop round 2 = Abstract trace after loop round 3)

Control Flow Graph

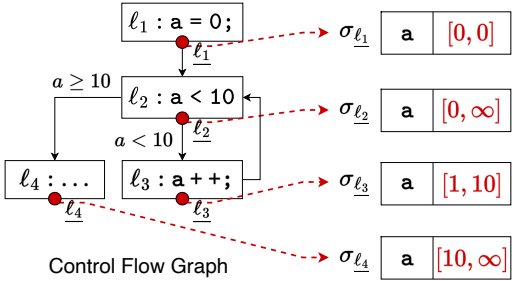
Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,\infty]$



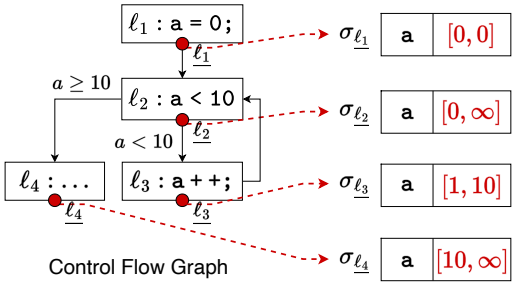
Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,\infty]$



Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,\infty]$

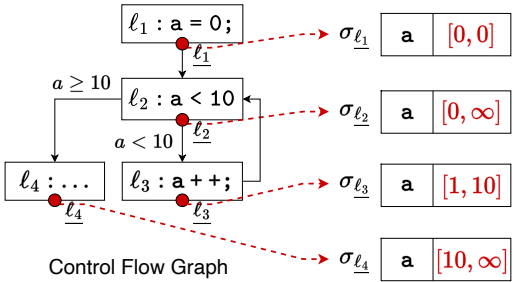


$[0,0] \Rightarrow [0,\infty] \Rightarrow [0,\infty]$

3 iterations while analyzing the loop

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,\infty]$



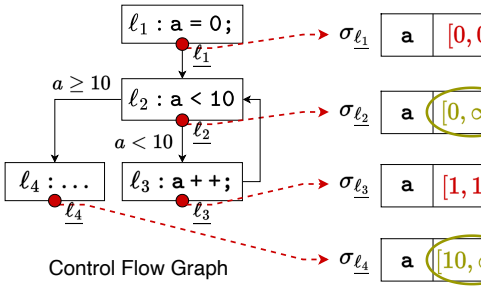
$[0,0] \Rightarrow [0,\infty] \Rightarrow [0,\infty]$

3 iterations while analyzing the loop

Faster than naive fixpoint computation (12 iterations)!

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,\infty]$



σ_{ℓ_1} | a | $[0,0]$

σ_{ℓ_2} | a | $[0,\infty]$ $\sqsupseteq$ $[0,10]$

σ_{ℓ_3} | a | $[1,10]$

σ_{ℓ_4} | a | $[10,\infty]$ $\sqsupseteq$ $[10,10]$

Less precise than without widening

$[0,0] \Rightarrow [0,\infty] \Rightarrow [0,\infty]$

3 iterations while analyzing the loop

Faster than without widening (12 iterations)!

Narrowing: Precision Refinement

Narrowing can recover some of the precision lost during widening while preserving convergence to a sound post-fixpoint (e.g., by improving imprecise $\sigma_{\underline{\ell_2}}$ and $\sigma_{\underline{\ell_4}}$).

Naive fixpoint computation: value changes of $\sigma_{\underline{\ell_2}}(a)$

$$[0,0] \Rightarrow [0,1] \Rightarrow \ldots \Rightarrow [0,10] \Rightarrow [0,10]$$

Widening

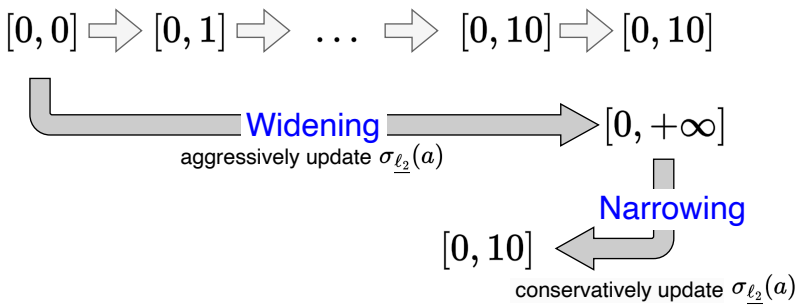
aggressively update $\sigma_{\underline{\ell_2}}(a)$

$$\Rightarrow [0,+\infty]$$

Narrowing: Precision Refinement

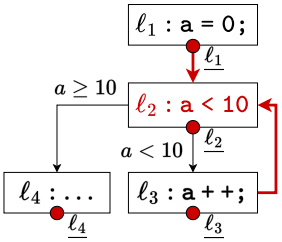
Narrowing can recover some of the precision lost during widening while preserving convergence to a sound post-fixpoint (e.g., by improving imprecise $\sigma_{\underline{\ell_2}}$ and $\sigma_{\underline{\ell_4}}$).

Naive fixpoint computation: value changes of $\sigma_{\underline{\ell_2}}(a)$



Narrowing: Precision Refinement

After the widening reaches a fixpoint at the k^{th} iteration when analyzing the loop, we start performing narrowing at the $(k+1)^{th}$ to update $\sigma_{\underline{\ell_2}}$.



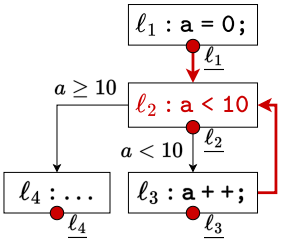
Control Flow Graph

Widening reaches a fixpoint

$$\sigma^k_{\underline{\ell_2}}(a) := \sigma^{k-1}_{\underline{\ell_2}}(a) \nabla (\sigma_{\underline{\ell_1}}(a) \sqcup \sigma^{k-1}_{\underline{\ell_3}}(a))$$

Narrowing: Precision Refinement

After the widening reaches a fixpoint at the k^{th} iteration when analyzing the loop, we start performing narrowing at the $(k+1)^{th}$ to update $\sigma_{\underline{\ell_2}}$.



Control Flow Graph

Widening reaches a fixpoint

$$\sigma^k_{\underline{\ell_2}}(a) := \sigma^{k-1}_{\underline{\ell_2}}(a) \nabla (\sigma_{\underline{\ell_1}}(a) \sqcup \sigma^{k-1}_{\underline{\ell_3}}(a))$$

Apply narrowing operator Δ instead

Start performing narrowing

$$\sigma^{k+1}_{\underline{\ell_2}}(a) := \sigma^{k}_{\underline{\ell_2}}(a) \Delta (\sigma_{\underline{\ell_1}}(a) \sqcup \sigma^{k}_{\underline{\ell_3}}(a))$$

What is a **Narrowing Operator**?

Narrowing Operator

The Narrowing Operator ($\Delta : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$) is formally defined on a poset $(\mathbb{A}, \sqsubseteq)$. Δ on interval domain could be defined as:

$$[l_1, h_1]\Delta[l_2, h_2] = [l_3, h_3]$$

Narrowing Operator

The Narrowing Operator ($\Delta : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$) is formally defined on a poset $(\mathbb{A}, \sqsubseteq)$. Δ on interval domain could be defined as:

$$[l_1, h_1]\Delta[l_2, h_2] = [l_3, h_3]$$

where

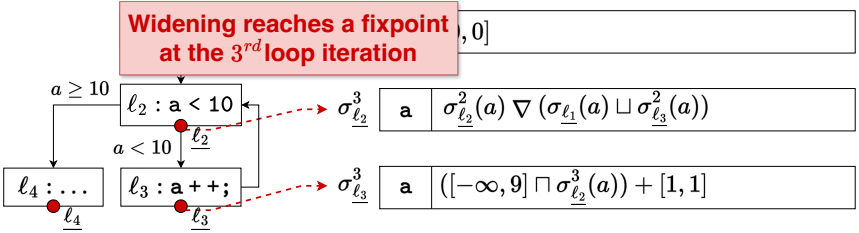
$$l_3 = \begin{cases} l_2 & l_1 \equiv -\infty \\ l_1 & l_1 \neq -\infty \end{cases}, h_3 = \begin{cases} h_2 & h_1 \equiv \infty \\ h_1 & h_1 \neq \infty \end{cases}$$

As a concrete example, $[0, \infty]\Delta[0, 10] = [0, 10]$.

Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter						
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3					
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$					
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$					
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$					
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$					

Widening reaches a fixpoint at the 3^{rd} loop iteration

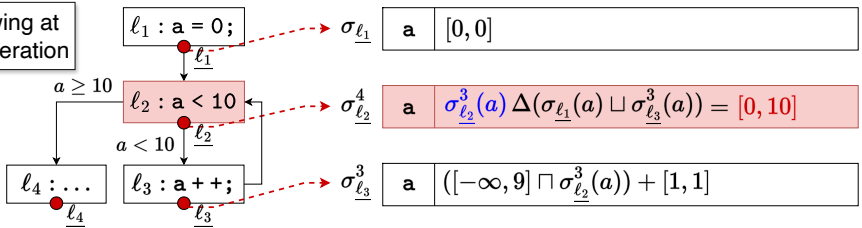


Control Flow Graph

Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2				
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$				
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$				
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$				
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$				

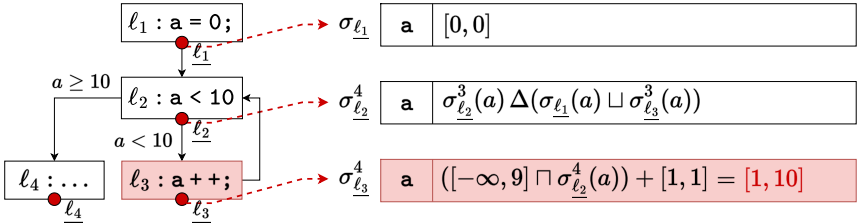
Start narrowing at the 4^{th} loop iteration



Control Flow Graph

Narrowing: The Loop Example

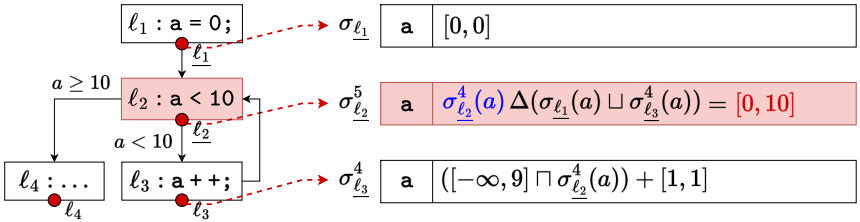
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3			
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$			
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$			
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$			
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$			



Control Flow Graph

Narrowing: The Loop Example

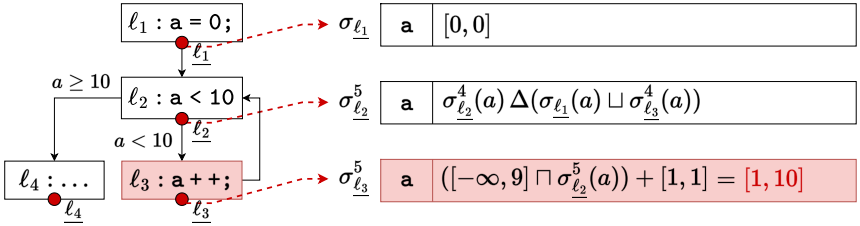
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter	
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	
$\sigma_{\underline{\ell_1}}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	
$\sigma_{\underline{\ell_2}}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	
$\sigma_{\underline{\ell_3}}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	
$\sigma_{\underline{\ell_4}}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	



Control Flow Graph

Narrowing: The Loop Example

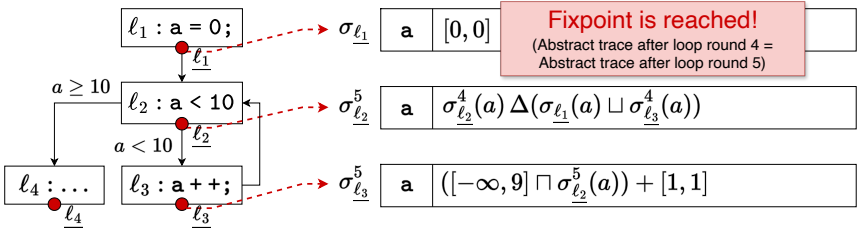
Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	



Control Flow Graph

Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	

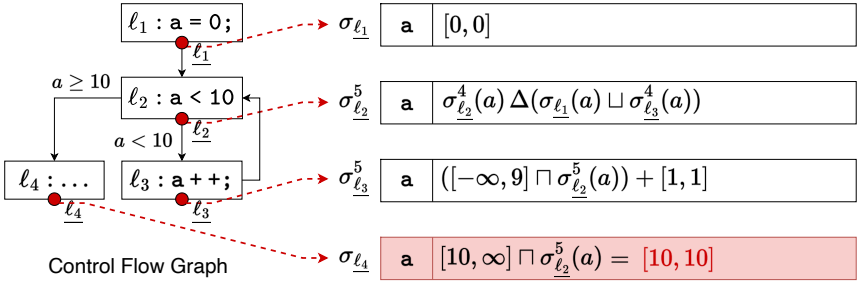


Fixpoint is reached!
(Abstract trace after loop round 4 = Abstract trace after loop round 5)

Control Flow Graph

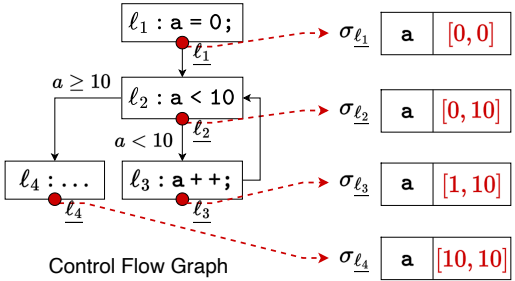
Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



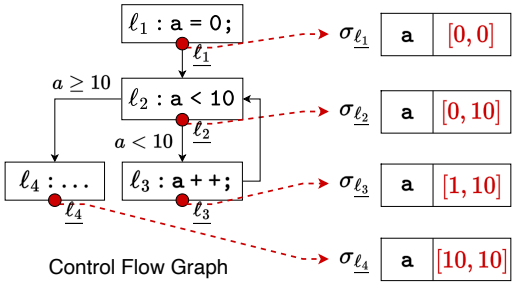
Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0, 0]$	$[0, 0]$	$[0, \infty]$	$[0, \infty]$	$[0, \infty]$	$[0, \infty]$	$[0, 10]$	$[0, 10]$	$[0, 10]$	$[0, 10]$	$[0, 10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1, 1]$	$[1, 1]$	$[1, 10]$	$[1, 10]$	$[1, 10]$	$[1, 10]$	$[1, 10]$	$[1, 10]$	$[1, 10]$	$[1, 10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10, 10]$

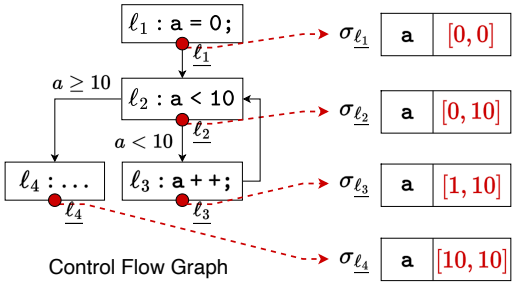


$[0, 0] \Rightarrow [0, \infty] \Rightarrow [0, \infty] \Rightarrow [0, 10] \Rightarrow [0, 10]$

5 iterations while analyzing the loop

Narrowing: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1st loop iter		2nd loop iter		3rd loop iter		4th loop iter		5th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\ell_1}(a)$	$\bot$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\sigma_{\ell_2}(a)$	$\bot$	$\bot$	$[0,0]$	$[0,0]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,\infty]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$
$\sigma_{\ell_3}(a)$	$\bot$	$\bot$	$\bot$	$[1,1]$	$[1,1]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$	$[1,10]$
$\sigma_{\ell_4}(a)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$[10,10]$



$[0,0] \Rightarrow [0,\infty] \Rightarrow [0,\infty] \Rightarrow [0,10] \Rightarrow [0,10]$

5 iterations while analyzing the loop

Faster!

Precise!