

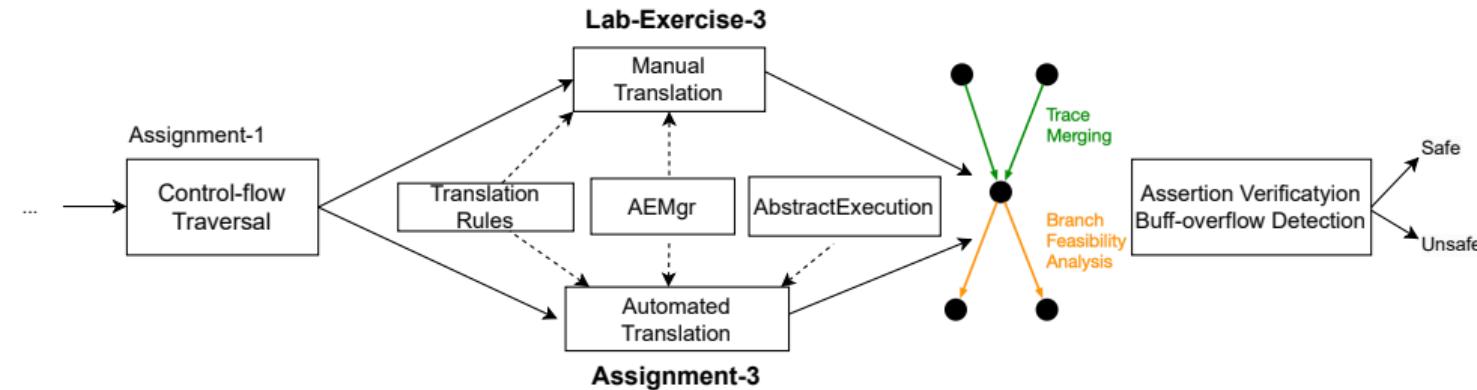
Foundations of Abstract Interpretation

(Week 8)

Yulei Sui

School of Computer Science and Engineering
University of New South Wales, Australia

Classes in the Next Three Weeks



Outline of Today's lecture

- An Introduction to Abstract Interpretation: What and Why
- Abstract Interpretation vs Symbolic Execution
- Definitions: Abstract domains, Abstract State and Abstract Trace.
- Step-by-Step Motivating Examples.
- Widening and Narrowing to Improve Analysis Speed and Precision

Abstract Interpretation

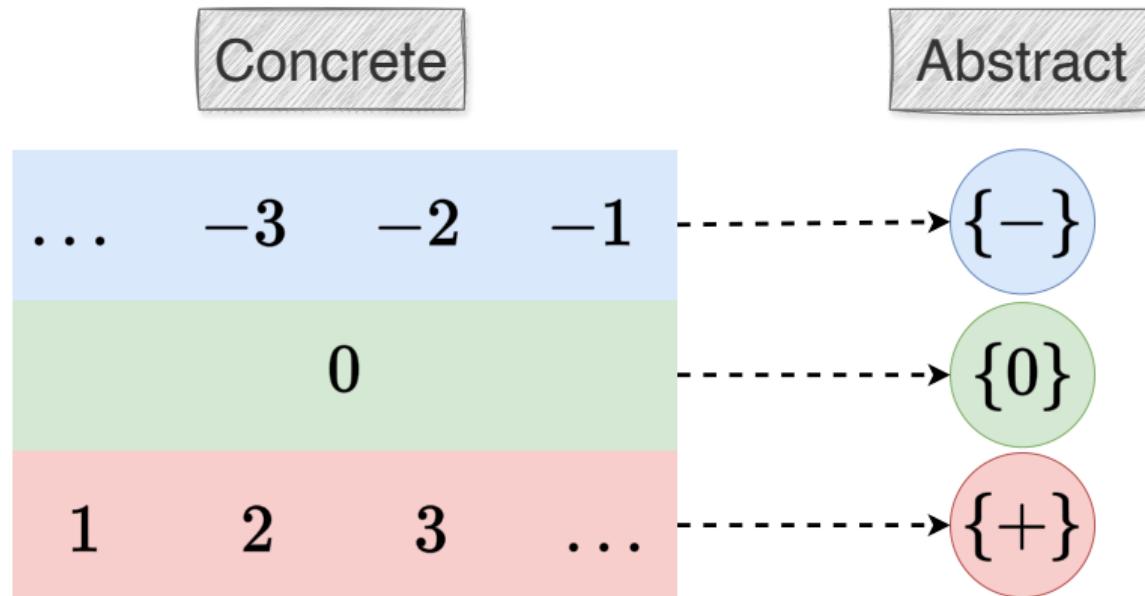
Abstract interpretation or Abstract Execution [Cousot & Cousot, POPL'77]¹, a general framework for static analysis, aims to **soundly approximate** the potential concrete values program variables may take during runtime, **based on monotonic functions over ordered sets, particularly lattices.**

Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.

Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.



Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.

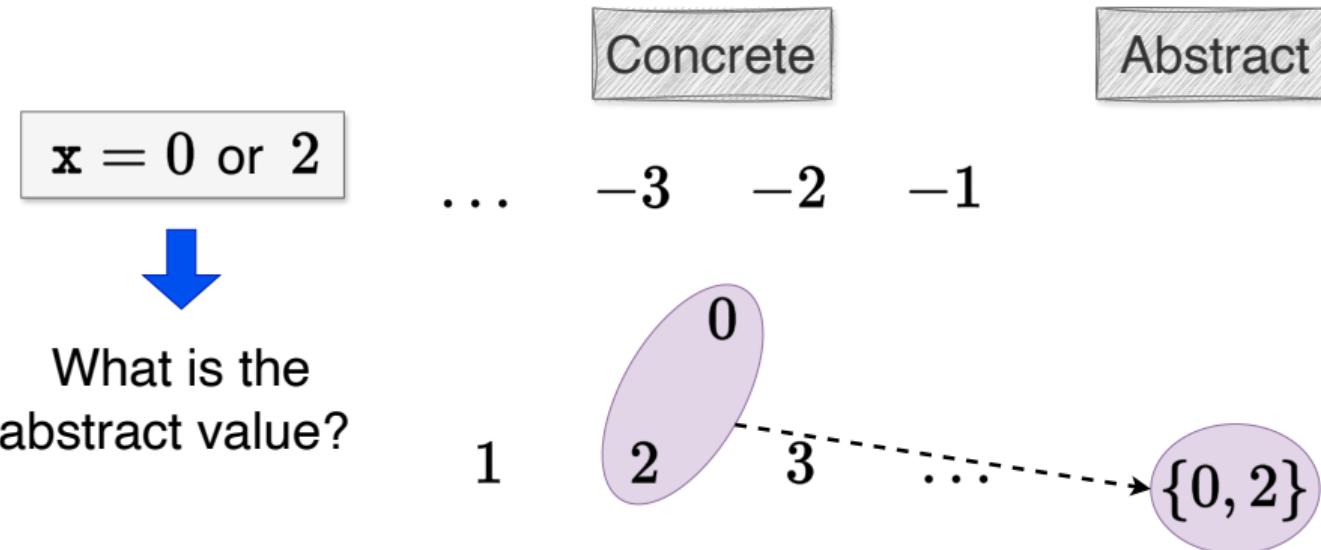
x = 0 or 2



What is the
abstract value?

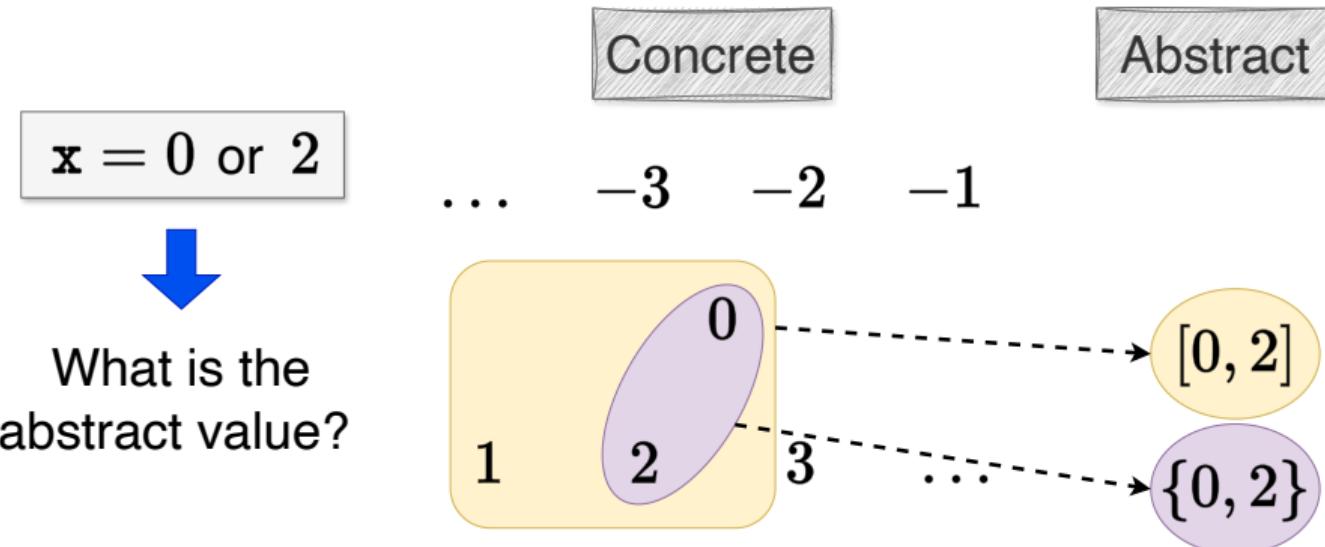
Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.



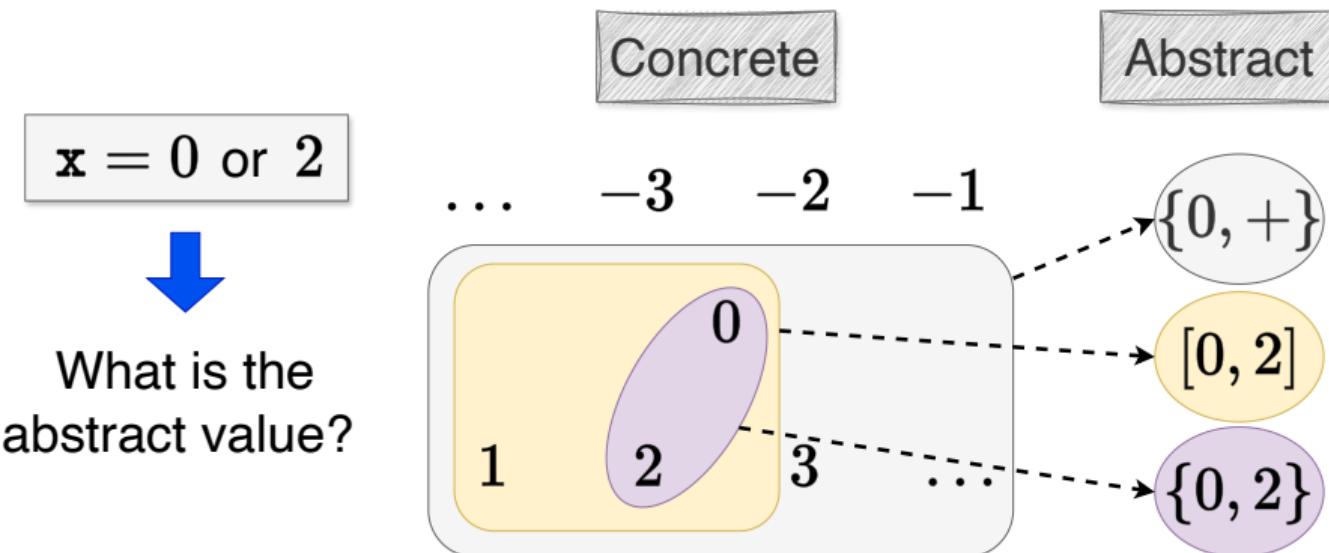
Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.



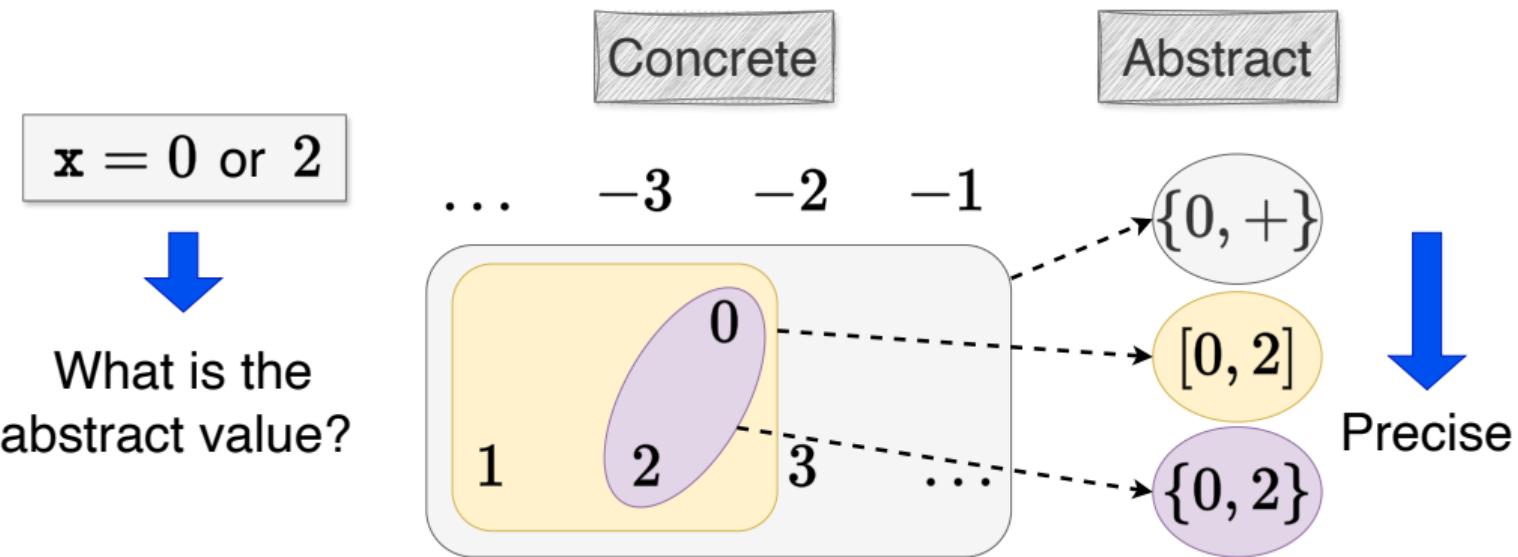
Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.



Abstract Interpretation: Levels of Abstractions

The key lies in abstracting a potentially infinite number of concrete values into a finite number of abstract values.



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 Analyzer**, **BAP**, **Soot** and many more ...

Abstract Interpretation vs. Symbolic Execution

Soundness

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

Abstract Interpretation vs. Symbolic Execution

Soundness

- **Abstract interpretation** aims for **sound results**. It can conservatively approximate **all possible execution paths and runtime behaviors**.
- **Symbolic execution** can be **unsound**. It precisely explores individual yet **feasible paths**, facing a “path explosion” problem in large programs, and may result in **under-approximation** of program behaviors.

Assignment-2 vs. Assignment-3

Assignment-2

- **Delegate** the constraint solving to the **z3 SMT solver**.
- Each time, it returns **one solution with concrete values for all variables** in the search space when the solver is satisfiable.
- Per-path verification **without handling the inner parts of a loop**.

Assignment-3

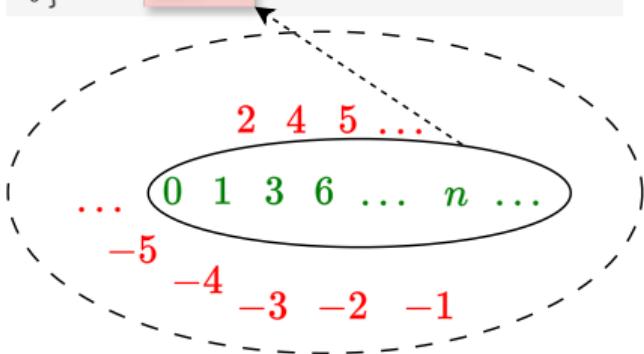
- Use **Abstract State** (AEState) and **Abstract Trace** (a set of AEStates for all ICFGNodes) to **compute and maintain abstract values** of variables.
- **Abstract all possible values** of a variable into a value **interval** (for scalars) or an **address set** (for memory addresses).
- **Approximate loop behaviors** based on widening and narrowing.

Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

```
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. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

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

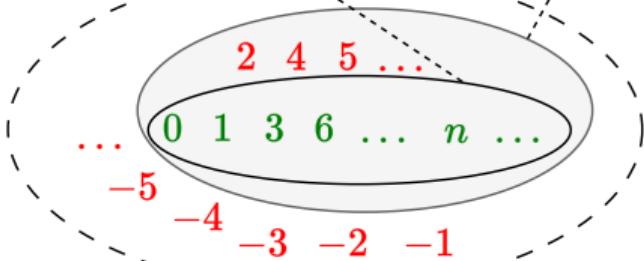
sum = ?

Abstract Interpretation

{0, +}

Sound (include all non-negative numbers)

imprecise (may include infeasible numbers: 2, 4, 5, ...)



Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

```
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

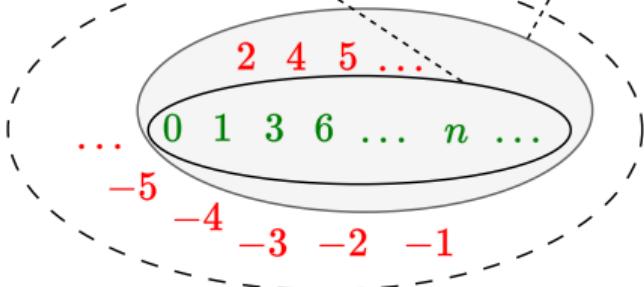
{0, +}

Sound (include all non-negative numbers)

imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

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

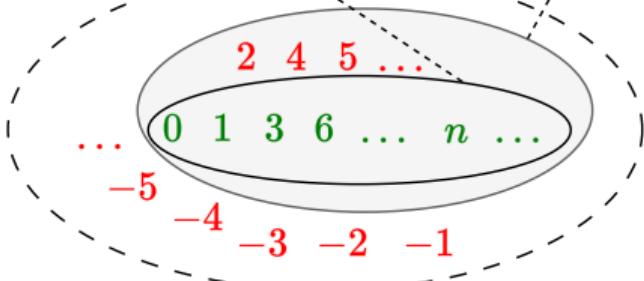


Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

```
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
$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_6$	0
$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_5 \rightarrow \ell_6$	1

Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

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

sum = ?

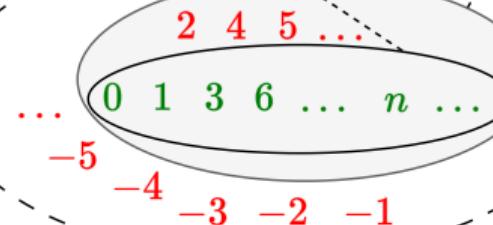
Abstract Interpretation

{0, +}

Sound (include all non-negative numbers)

imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution



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

Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

```
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

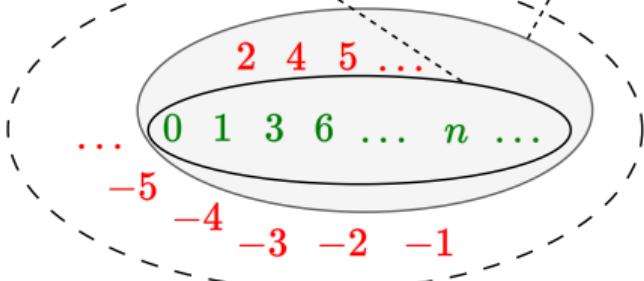
{0, +}

Sound (include all non-negative numbers)

imprecise (may include infeasible numbers: 2, 4, 5, ...)

Symbolic Execution

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



Abstract Interpretation vs. Symbolic Execution

Over-Approximation (soundness) vs. Under-Approximation (unsoundness)

```
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

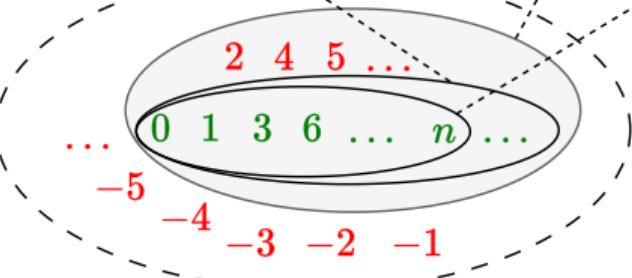
{0, +}

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
$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_6$	0
$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_5 \rightarrow \ell_6$	1
$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_5 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_5 \rightarrow \ell_6$	3
.....	infinite paths!
	...

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. Symbolic Execution

Termination

- **Abstract interpretation** is typically guaranteed to terminate within a finite step. Uses an abstracted, and hence more manageable, version of the state space to represent the infinite number of runtime states and paths.

Abstract Interpretation vs. Symbolic Execution

Termination

- **Abstract interpretation** is typically guaranteed to terminate within a finite step. Uses an abstracted, and hence more manageable, version of the state space to represent the infinite number of runtime states and paths.
- **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:** ensure that all parts of the code are analyzed, avoiding missed sections and ensuring thorough coverage for detecting issues.

Abstract Interpretation: A Code Example

Approximation!

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

x	{+}
---	-----

x	[1, 3]
---	--------

x	{1, 3}
---	--------

Abstract Interpretation: A Code Example

Approximation!

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

Coarse-grained
but faster

x	{+}
---	-----

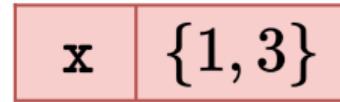
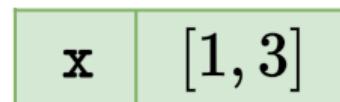
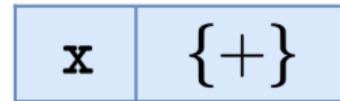
x	[1, 3]
---	--------

x	{1, 3}
---	--------

Abstract Interpretation: A Code Example

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

Approximation!



Fine-grained
but slower



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.

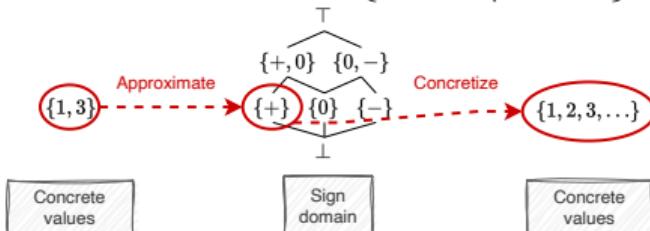
Abstract Domain

- An **abstract domain** \mathbb{A} contains *abstract values* approximating a set of concrete values.
- An abstract domain is typically implemented using a **lattice** $\mathbb{L} = \langle \mathbb{A}, \sqsubseteq, \sqcap, \sqcup, \perp, \top \rangle$ 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 \perp and \top are unique least and greatest elements of \mathbb{A} .

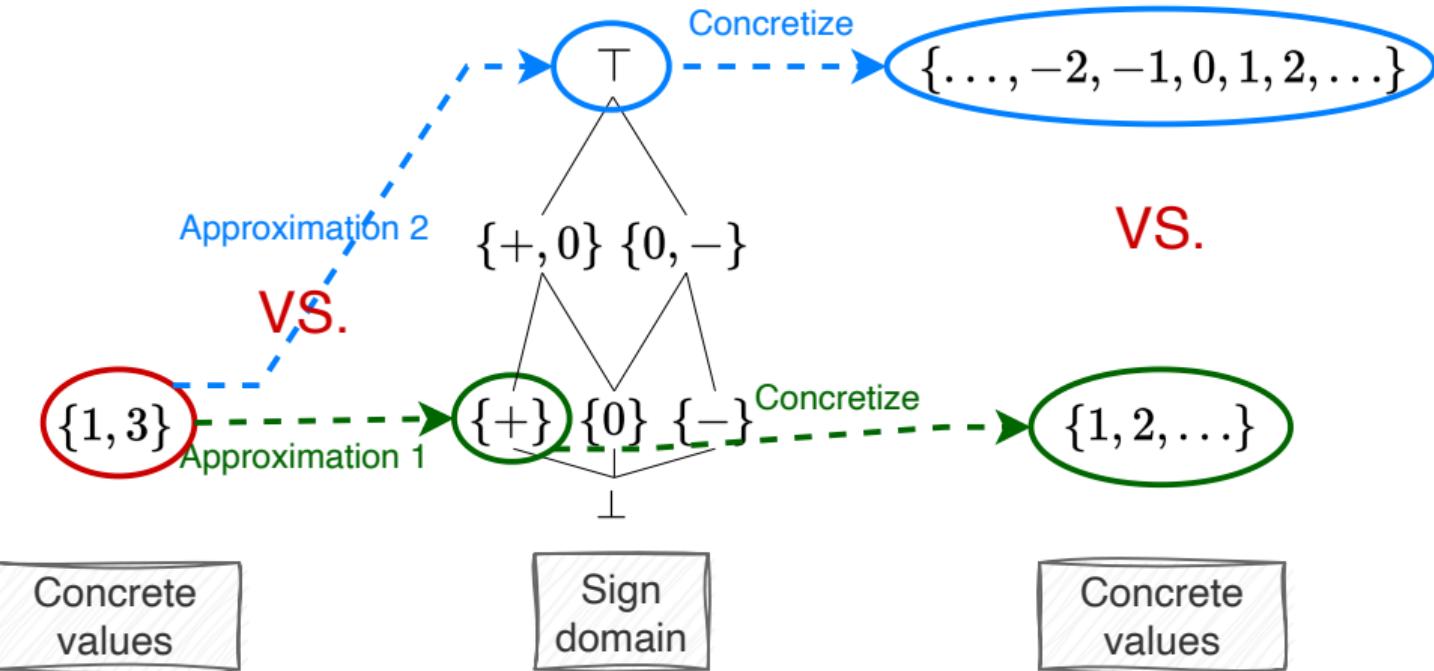
An Example: Abstract Sign Domain

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

- Lattice is defined as $\mathbb{L} = \langle \mathcal{P}(\{-, 0, +\}), \sqsubseteq, \sqcap, \sqcup, \perp, \top \rangle$.
- **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**)
 - $\{+\} \sqcap \{0\} = \perp$
- **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

When each concrete value has a unique best abstraction, the correspondence is a **Galois connection**, which is a two-way connections between abstract domain and concrete domain using abstraction function and concretization function.

- Abstraction function $\alpha : \mathbb{C} \rightarrow \mathbb{A}$ maps a set of concrete values to its abstract ones;
- Concretization function $\gamma : \mathbb{A} \rightarrow \mathbb{C}$ maps a set of abstract values to concrete ones.

Galois Connection

When each concrete value has a unique best abstraction, the correspondence is a **Galois connection**, which is a two-way connections between abstract domain and concrete domain using abstraction function and concretization function.

- Abstraction function $\alpha : \mathbb{C} \rightarrow \mathbb{A}$ maps a set of concrete values to its abstract ones;
- Concretization function $\gamma : \mathbb{A} \rightarrow \mathbb{C}$ maps a set of abstract values to concrete ones.

Example: Abstraction/concretization functions on sign domain

$$\gamma_{\text{Sign}}(\top) = \mathbb{Z}$$

$$\gamma_{\text{Sign}}(\{-\}) = \{x \mid x < 0\}$$

$$\gamma_{\text{Sign}}(\{+\}) = \{x \mid x > 0\}$$

...

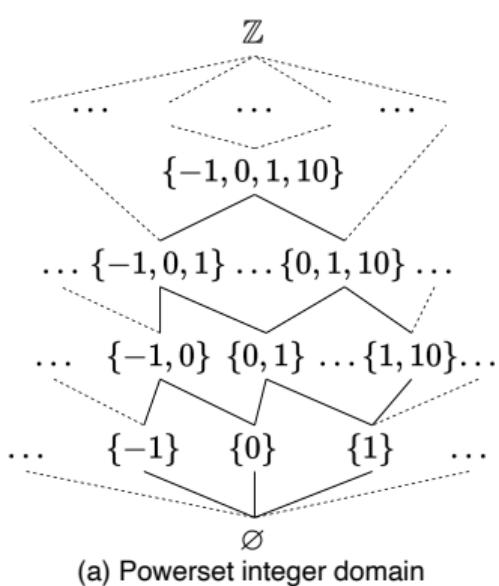
$$\alpha_{\text{Sign}}(c) = \{+\} \text{ if } c \in \mathbb{Z}_{>0}$$

$$\alpha_{\text{Sign}}(c) = \{-\} \text{ if } c \in \mathbb{Z}_{<0}$$

$$\alpha_{\text{Sign}}(c) = \{+, 0\} \text{ if } c \in \mathbb{Z}_{\geq 0}$$

...

Galois Connection of Sign Domain

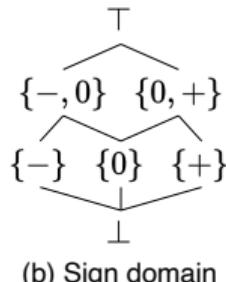


$$\gamma_{Sign}(\top) = \mathbb{Z}$$
$$\gamma_{Sign}(\{-\}) = \{x \mid x < 0\}$$
$$\gamma_{Sign}(\{+\}) = \{x \mid x > 0\}$$
$$\gamma_{Sign}(\{-, 0\}) = \{x \mid x \leq 0\}$$
$$\gamma_{Sign}(\{+, 0\}) = \{x \mid x \geq 0\}$$
$$\gamma_{Sign}(\{\perp\}) = \emptyset$$

(blue dashed box)

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

(red dashed box)



Interval Domain

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

- Lattice is defined as $\mathbb{L}_{interval} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \perp, \top \rangle$, where $\mathbb{I} = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}\} \cup \{\perp\}$.
- 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

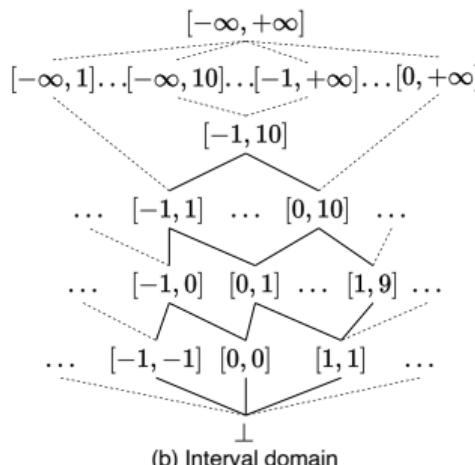
The interval domain is an abstract domain that represents a set of integers that fall between two given endpoints.

- Lattice is defined as

$$\mathbb{L}_{\text{interval}} = \langle \mathbb{I}, \sqsubseteq, \sqcap, \sqcup, \perp, \top \rangle, \text{ where } \mathbb{I} = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}\} \cup \{\perp\}.$$

- 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., $[0, 0], [0, 1] \in \mathbb{A}_{\text{interval}}$, satisfying $[0, 0] \sqsubseteq [0, 1]$.



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

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

- GLB = $[7, 8]$, the largest range that is shared by both a_1 and a_2 .

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

- LUB = $[3, 12]$, the smallest range that includes both a_1 and a_2 .

LUB and GLB of lattice $\mathbb{L}_{\text{interval}}$ are $[-\infty, +\infty]$ and \perp respectively.

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

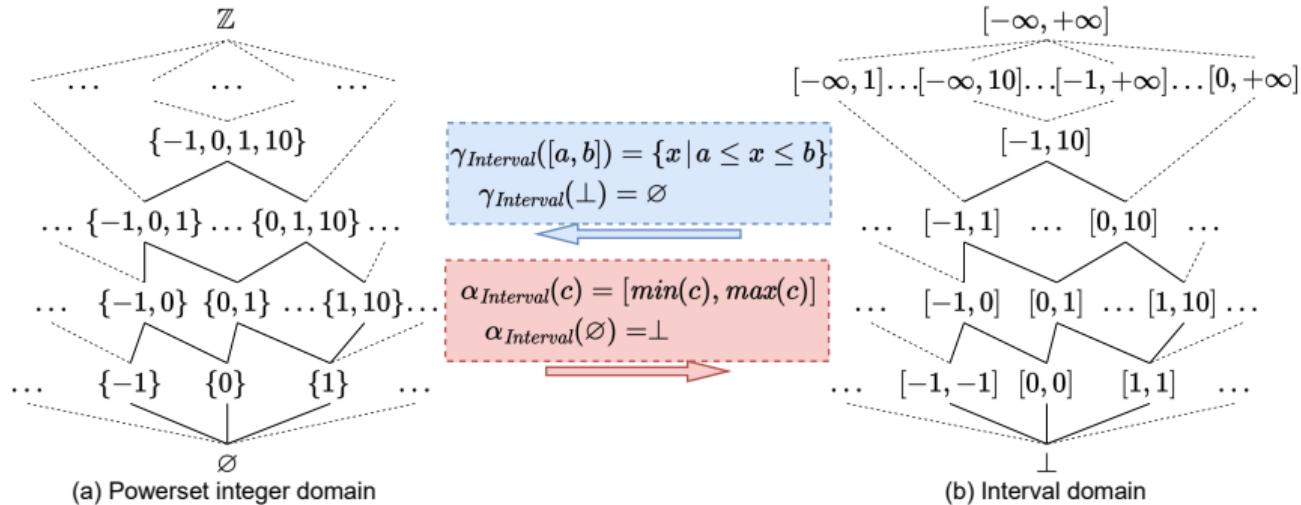


Figure: Powerset integer domain \mathbb{C} and its abstraction as the interval domain $\mathbb{A}_{interval}$.

Abstract State and Abstract Trace

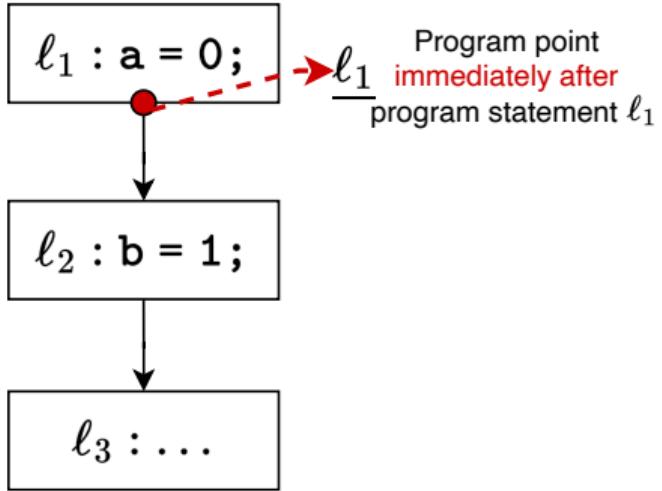
- An **abstract state** (AEState in Lab-3 and Assignment-3) is defined as a map $AS : \mathcal{V} \rightarrow \mathbb{A}$ associating program variables \mathcal{V} with an abstract value in \mathbb{A} , approximating the runtime states of program variables.

Abstract State and Abstract Trace

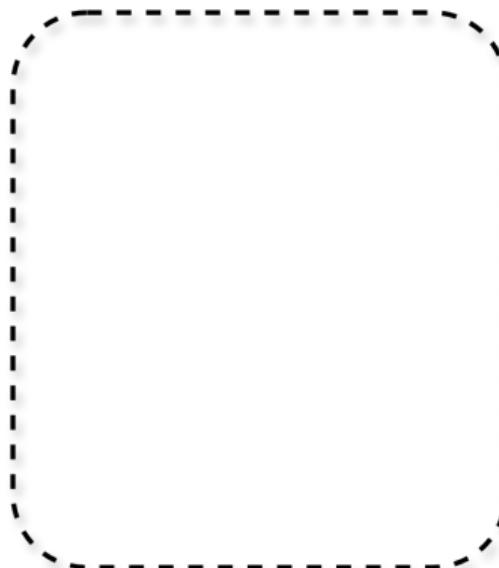
- An **abstract state** (`AESState` in Lab-3 and Assignment-3) is defined as a map $AS : \mathcal{V} \rightarrow \mathbb{A}$ associating program variables \mathcal{V} with an abstract value in \mathbb{A} , approximating the runtime states of program variables.
- An **abstract trace** $\sigma \in \mathbb{L} \times \mathcal{V} \rightarrow \mathbb{A}$ represents a list of abstract states before ($\bar{\ell}$) and after ($\underline{\ell}$) each program statement ℓ (`preAbsTrace` and `postAbsTrace` in Assignment-3).

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

Abstract Trace : A Simple Example

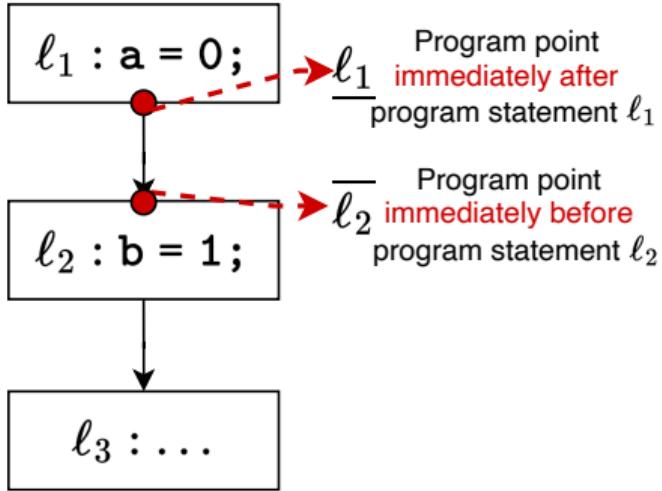


Control Flow Graph

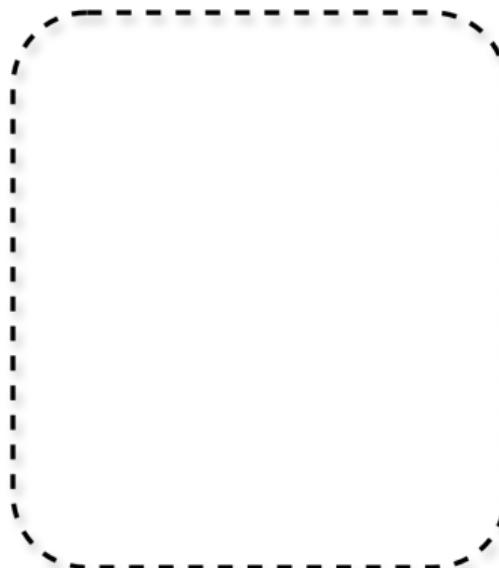


Abstract Trace σ

Abstract Trace : A Simple Example

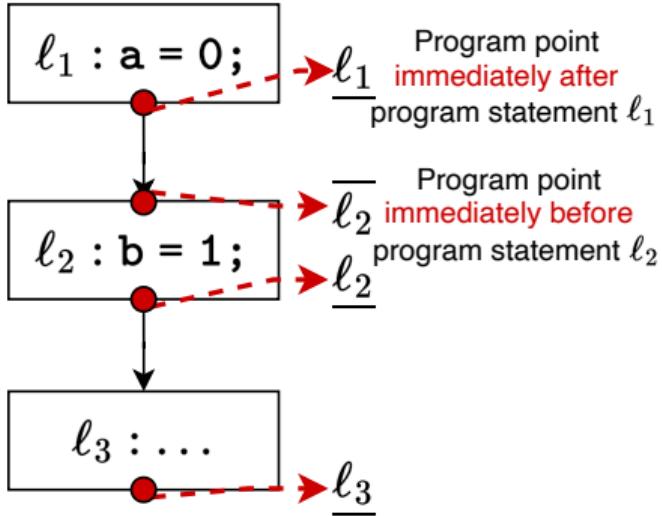


Control Flow Graph

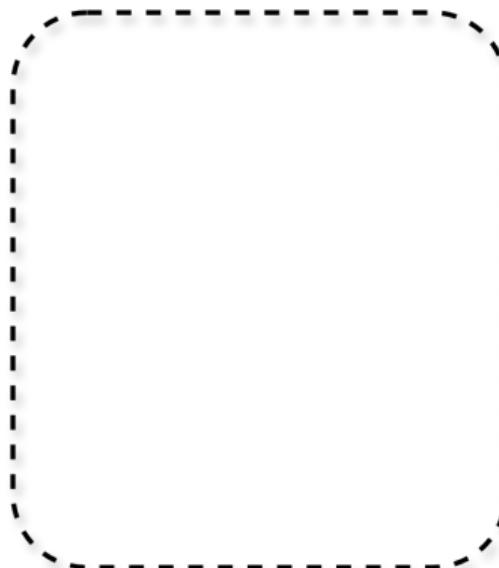


Abstract Trace σ

Abstract Trace : A Simple Example

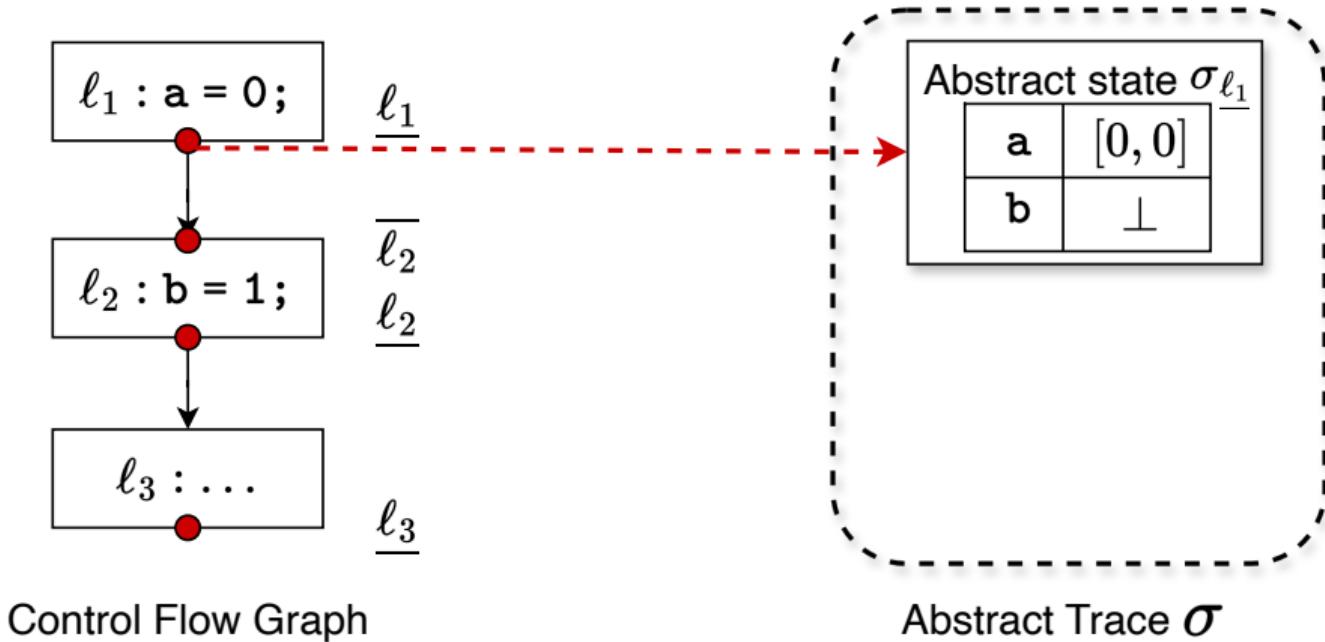


Control Flow Graph

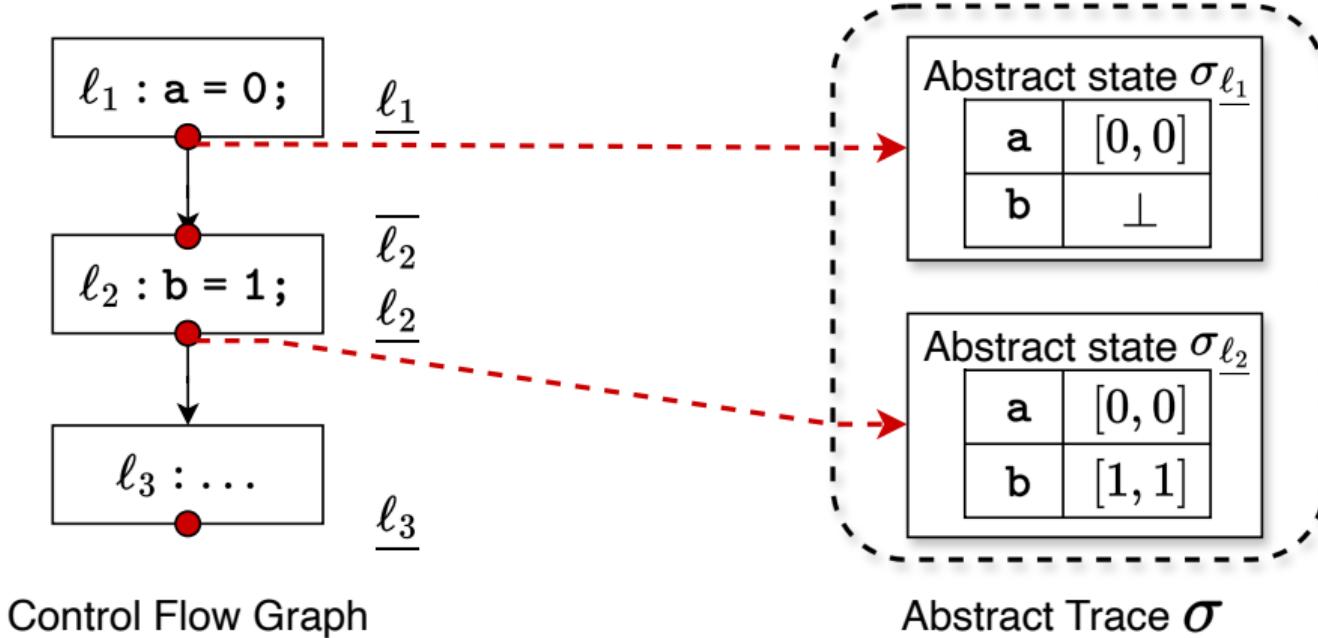


Abstract Trace σ

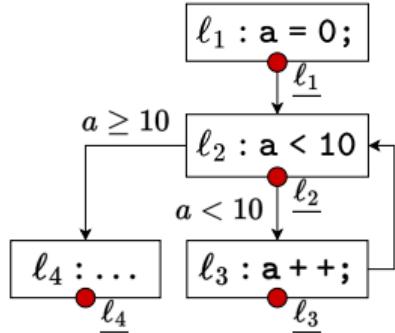
Abstract Trace : A Simple Example



Abstract Trace : A Simple Example

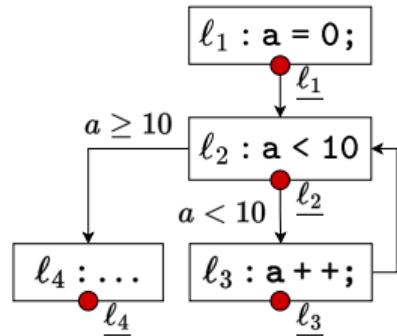


Abstract Trace: Naive Fixed-Point Computation for Loops



Control Flow Graph

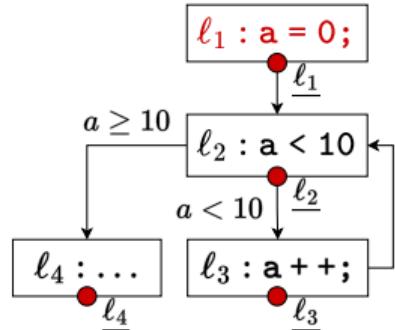
Abstract Trace: Naive Fixed-Point Computation for Loops



What is the abstract state after analyzing each statement?

Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops



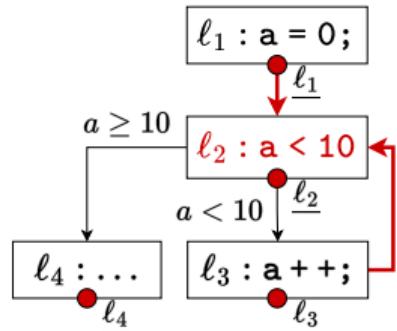
What is the abstract state after analyzing each statement?

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

Control Flow Graph

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

Abstract Trace: Naive Fixed-Point Computation for Loops



Control Flow Graph

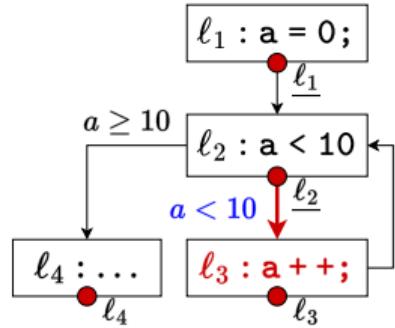
What is the abstract state after analyzing each statement?

$$\sigma_{\underline{\ell}_1}(a) := F_1() = [0, 0]$$

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

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

Abstract Trace: Naive Fixed-Point Computation for Loops



Control Flow Graph

What is the abstract state after analyzing each statement?

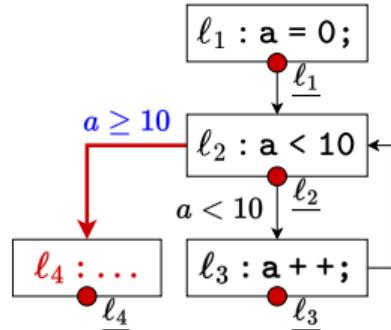
$$\sigma_{\underline{\ell}_1}(a) := F_1() = [0, 0]$$

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

$$\sigma_{\ell_3}(a) := F_3(\sigma_{\ell_2}) = \textcolor{blue}{([-∞, 9] ∩ σ_{ℓ_2}(a))} + [1, 1]$$

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

Abstract Trace: Naive Fixed-Point Computation for Loops



Control Flow Graph

What is the abstract state after analyzing each statement?

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

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

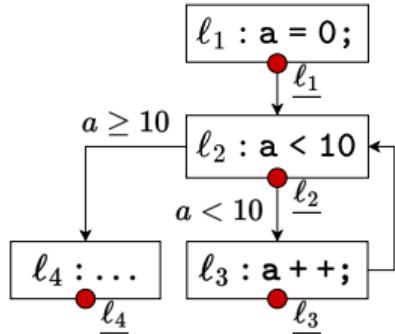
$$\sigma_{\ell_3}(a) := F_3(\sigma_{\ell_2}) = ([-\infty, 9] \sqcap \sigma_{\ell_2}(a)) + [1, 1]$$

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

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

Abstract Trace: Naive Fixed-Point Computation for Loops

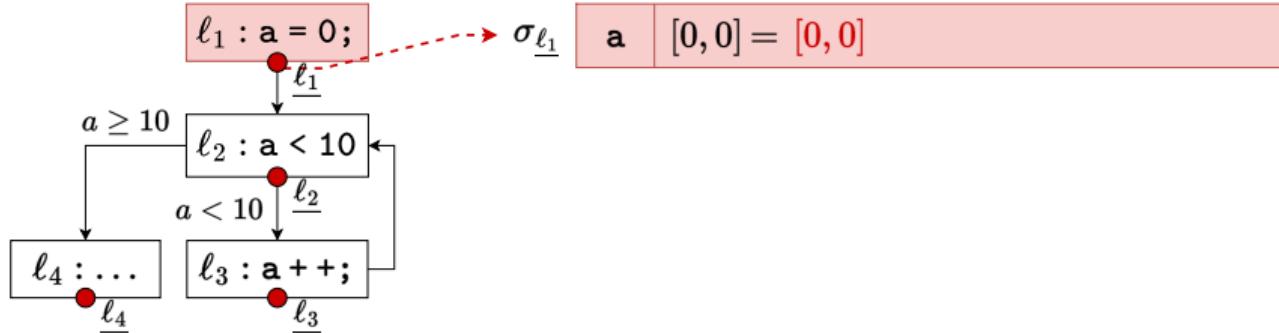
Abstract trace	Init										
$\sigma_{\ell_1}(a)$	\perp										
$\sigma_{\ell_2}(a)$	\perp										
$\sigma_{\ell_3}(a)$	\perp										
$\sigma_{\ell_4}(a)$	\perp										



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

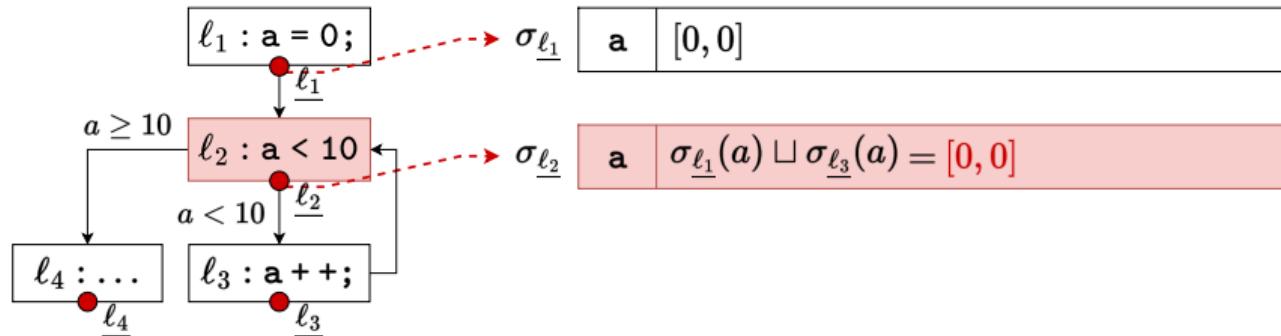
Abstract trace	Init	After analyzing ℓ_1										
$\sigma_{\ell_1}(a)$	\perp	[0, 0]										
$\sigma_{\ell_2}(a)$	\perp	\perp										
$\sigma_{\ell_3}(a)$	\perp	\perp										
$\sigma_{\ell_4}(a)$	\perp	\perp										



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

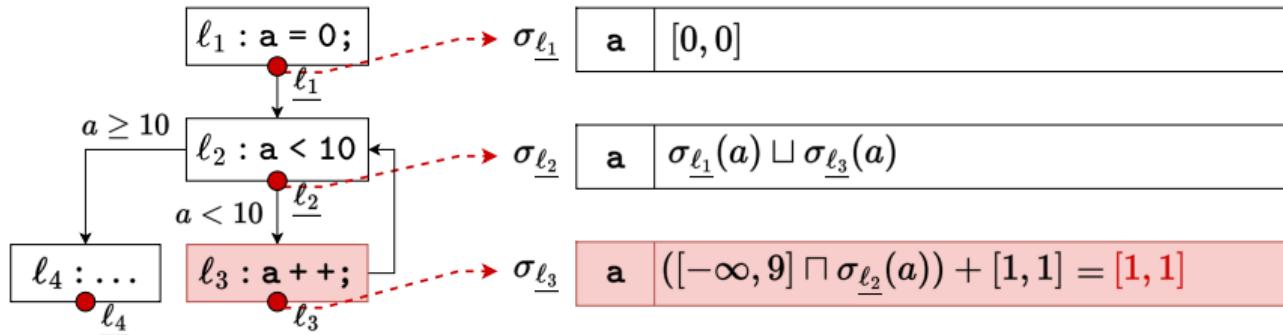
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter							
			After ℓ_2							
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]							
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]							
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp							
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp							



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

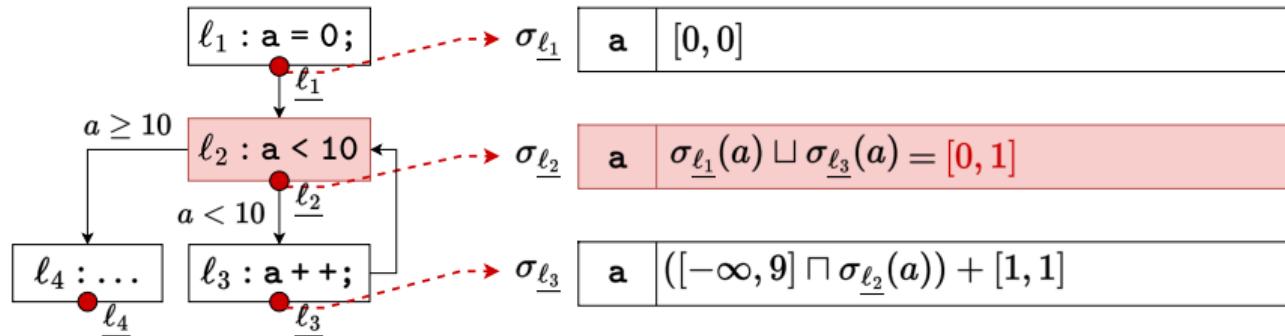
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter					
			After ℓ_2	After ℓ_3				
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]				
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]				
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]				
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp				



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

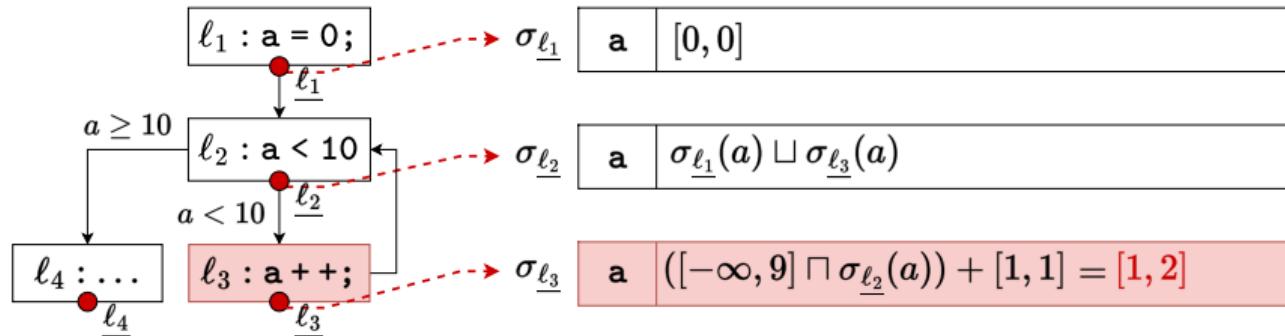
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter					
			After ℓ_2	After ℓ_3	After ℓ_2					
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]					
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]					
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]					
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp					



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

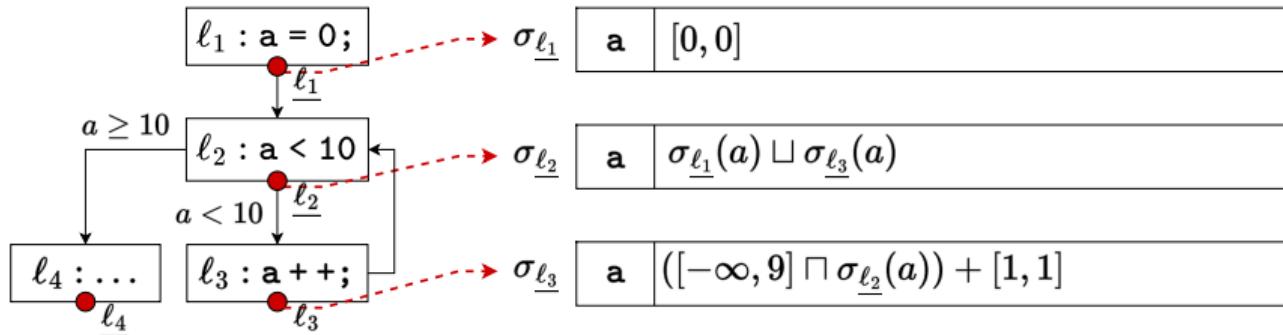
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter				
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3			
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]			
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]			
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]			
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp			



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

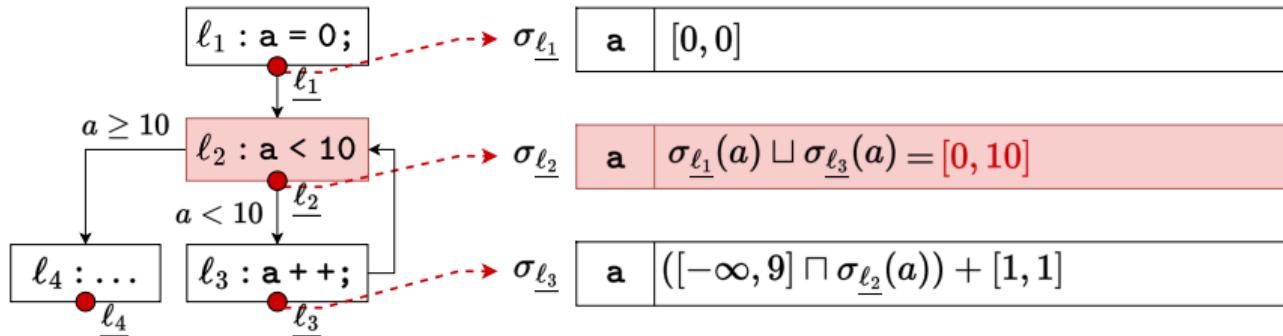
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		...	11 th loop iter			
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3		
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	...	[0, 0]	[0, 0]		
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]	...	[0, 10]	[0, 10]		
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]	...	[1, 10]	[1, 10]		
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	...	\perp	\perp		



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

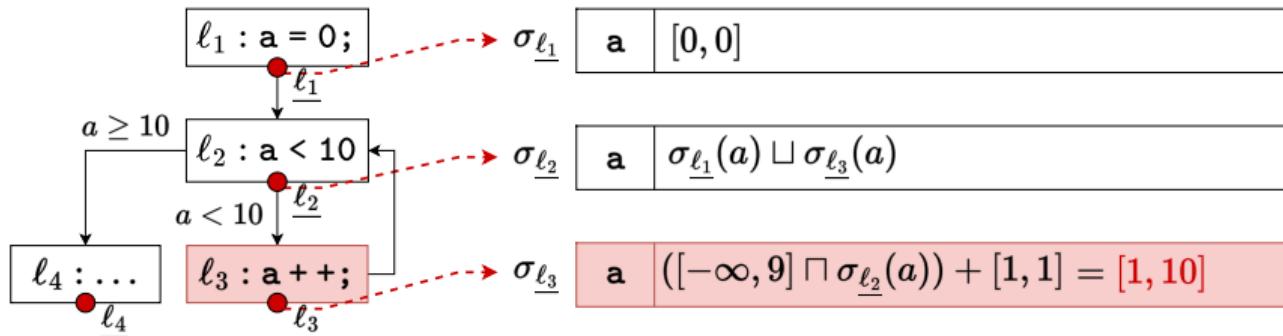
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		...	11 th loop iter		12 nd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	...	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]	...	[0, 10]	[0, 10]	[0, 10]	[0, 10]	
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]	...	[1, 10]	[1, 10]	[1, 10]	[1, 10]	
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	...	\perp	\perp	\perp	\perp	



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

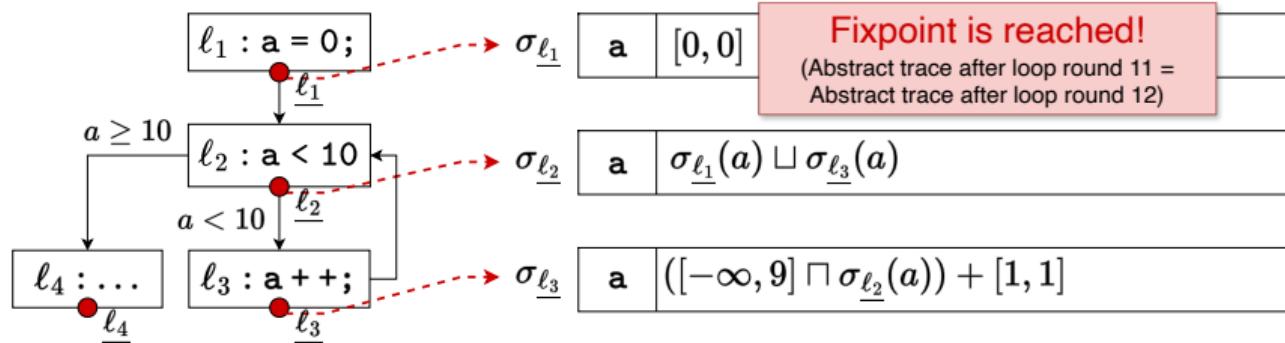
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		...	11 th loop iter		12 nd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	...	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]	...	[0, 10]	[0, 10]	[0, 10]	[0, 10]	
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]	...	[1, 10]	[1, 10]	[1, 10]	[1, 10]	
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	...	\perp	\perp	\perp	\perp	



Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

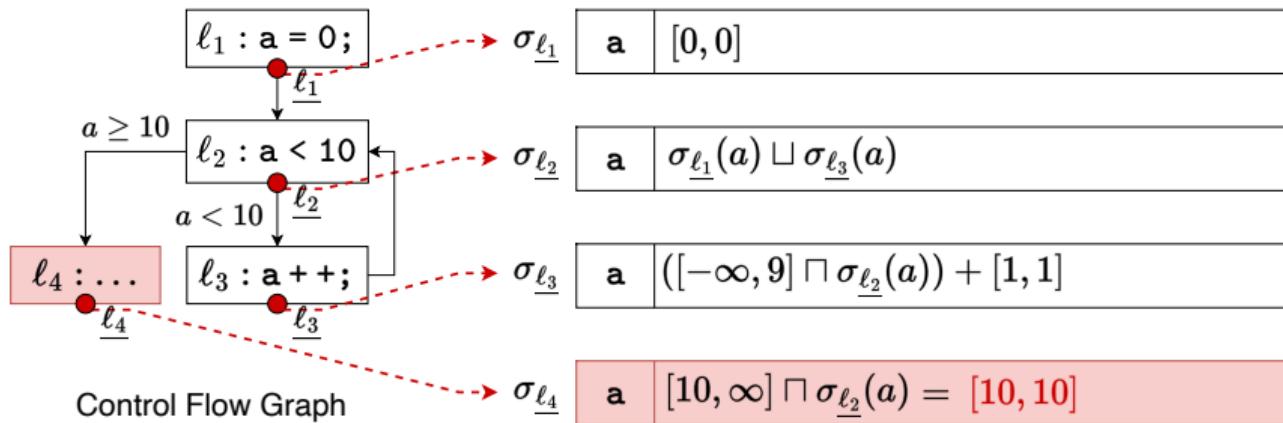
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		...	11 th loop iter		12 nd loop iter	
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3		After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3
$\sigma_{\underline{\ell_1}}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	...	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell_2}}(a)$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



Control Flow Graph

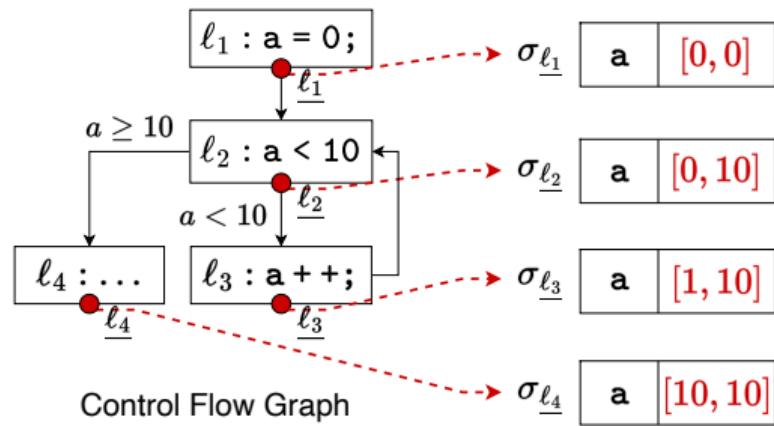
Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		...	11 th loop iter		12 nd 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	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	...	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]	...	[0, 10]	[0, 10]	[0, 10]	[0, 10]	[0, 10]
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]	...	[1, 10]	[1, 10]	[1, 10]	[1, 10]	[1, 10]
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	...	\perp	\perp	\perp	\perp	[10, 10]



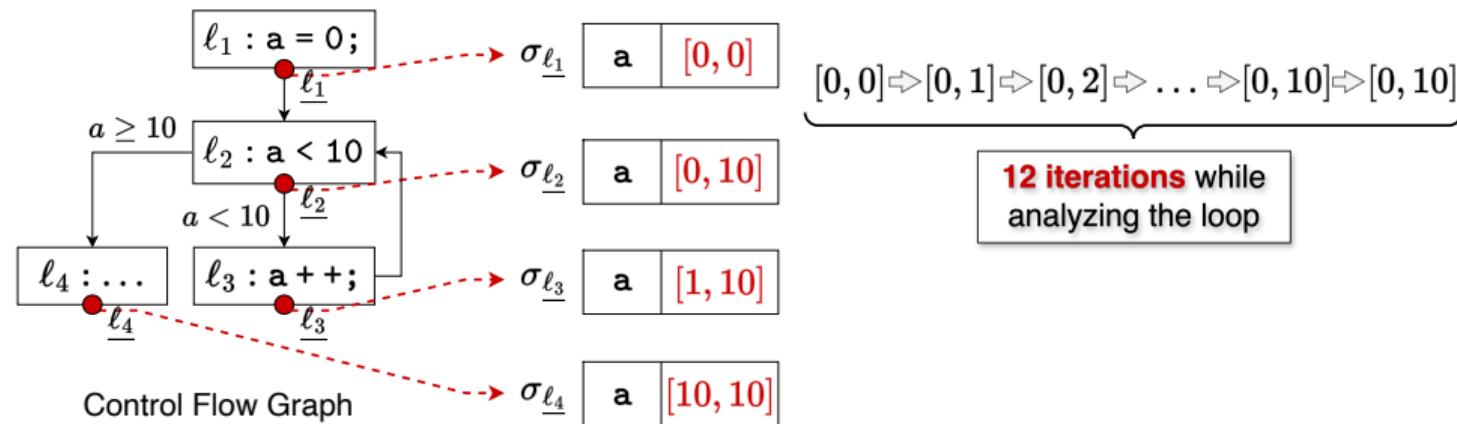
Abstract Trace: Naive Fixed-Point Computation for Loops

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



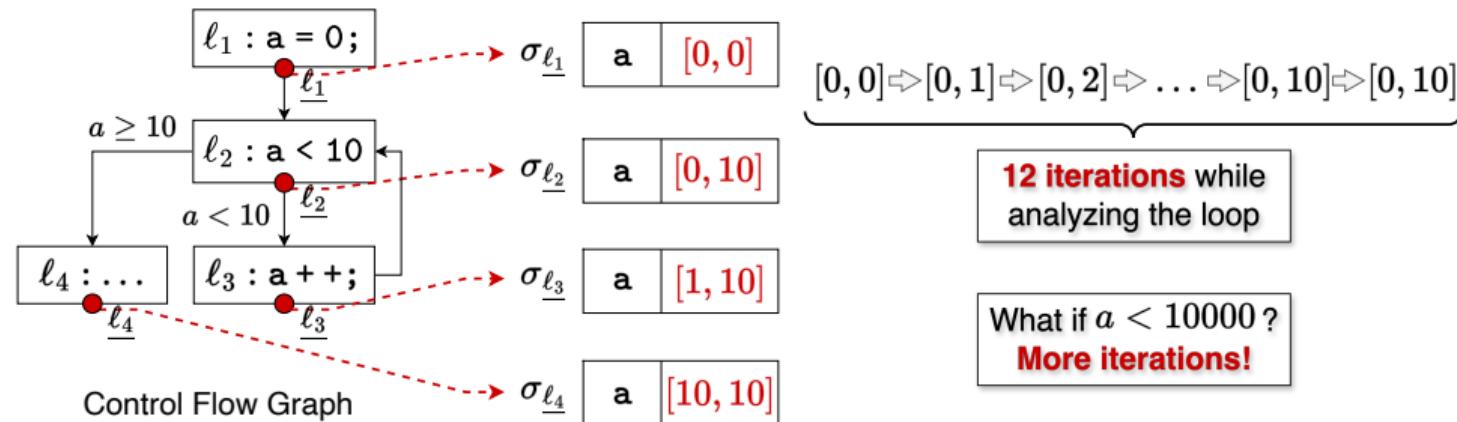
Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter				11 th loop iter		12 nd 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	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, 1]	[0, 1]	[0, 10]	[0, 10]	[0, 10]	[0, 10]	[0, 10]	[0, 10]	[0, 10]
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 2]	[1, 10]	[1, 10]	[1, 10]	[1, 10]	[1, 10]	[1, 10]	[1, 10]
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	[10, 10]



Abstract Trace: Naive Fixed-Point Computation for Loops

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

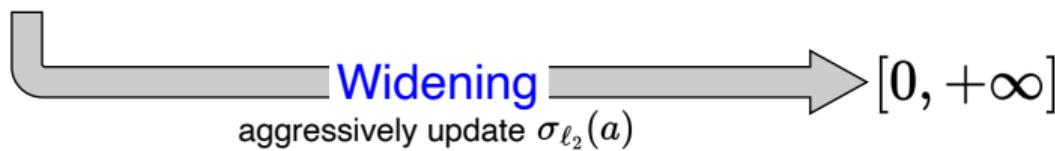
[0, 0] \Rightarrow [0, 1] \Rightarrow ... \Rightarrow [0, 10] \Rightarrow [0, 10]

Widening: Accelerating Fixed-Point Computation

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

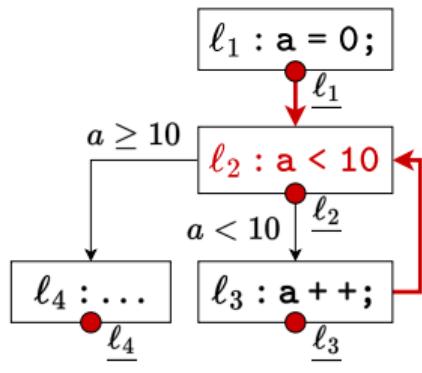
Naive fixpoint computation: value changes of $\sigma_{\ell_2}(a)$

$$[0, 0] \rightarrow [0, 1] \rightarrow \dots \rightarrow [0, 10] \rightarrow [0, 10]$$



Widening: Accelerating Fixed-Point Computation

Widening at the k^{th} iteration in the loop for analyzing ℓ_2 to update $\sigma_{\underline{\ell}_2}$.



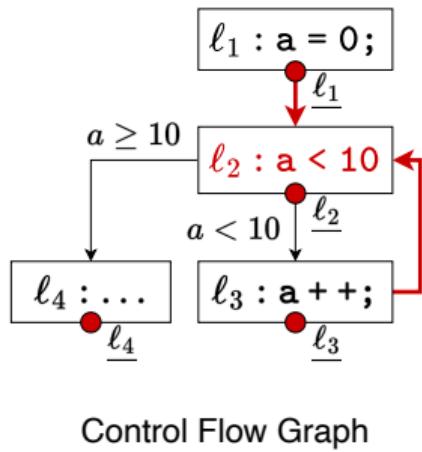
Control Flow Graph

$$\begin{aligned}\sigma_{\underline{\ell}_2}(a) := & \quad \sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}(a) \\ & \text{Apply widening operator } \nabla \\ \sigma_{\underline{\ell}_2}^k(a) := & \quad \sigma_{\underline{\ell}_2}^{k-1}(a) \nabla (\sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}^{k-1}(a))\end{aligned}$$

$\sigma_{\underline{\ell}_2}^k$ 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 $\sigma_{\underline{\ell}_2}$.



$$\begin{aligned}\sigma_{\underline{\ell}_2}(a) := & \quad \sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}(a) \\ & \text{Apply widening operator } \nabla \\ \sigma_{\underline{\ell}_2}^k(a) := & \quad \sigma_{\underline{\ell}_2}^{k-1}(a) \nabla (\sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}^{k-1}(a))\end{aligned}$$

$\sigma_{\underline{\ell}_2}^k$ 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 ($\nabla : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{A}$) 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 ($\nabla : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{A}$) is formally defined on a poset $(\mathbb{A}, \sqsubseteq)$. ∇ on interval domain could be defined as:

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

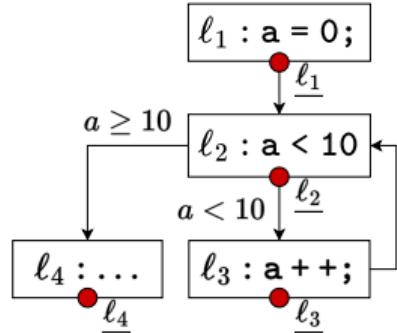
where

$$\ell_3 = \begin{cases} -\infty & \ell_2 < \ell_1 \\ \ell_1 & \ell_2 \geq \ell_1 \end{cases}, h_3 = \begin{cases} +\infty & h_2 > h_1 \\ h_1 & h_2 \leq h_1 \end{cases}$$

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

Widening: The Loop Example

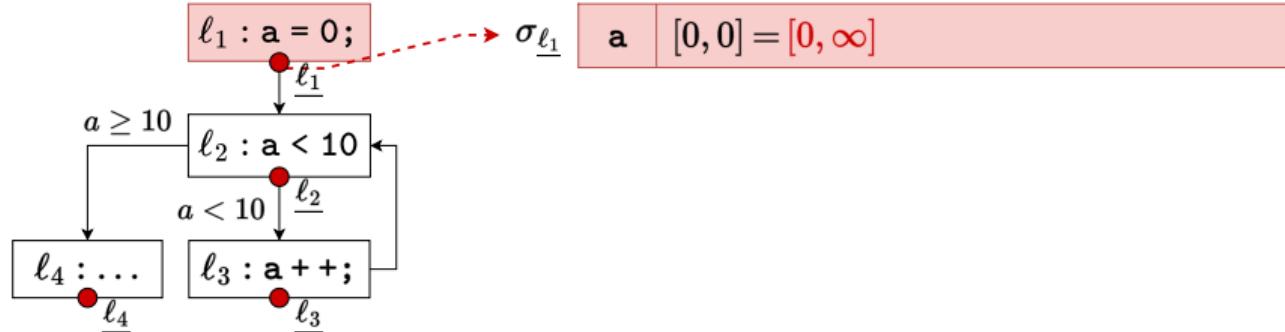
Abstract trace	Init								
$\sigma_{\ell_1}(a)$	\perp								
$\sigma_{\ell_2}(a)$	\perp								
$\sigma_{\ell_3}(a)$	\perp								
$\sigma_{\ell_4}(a)$	\perp								



Control Flow Graph

Widening: The Loop Example

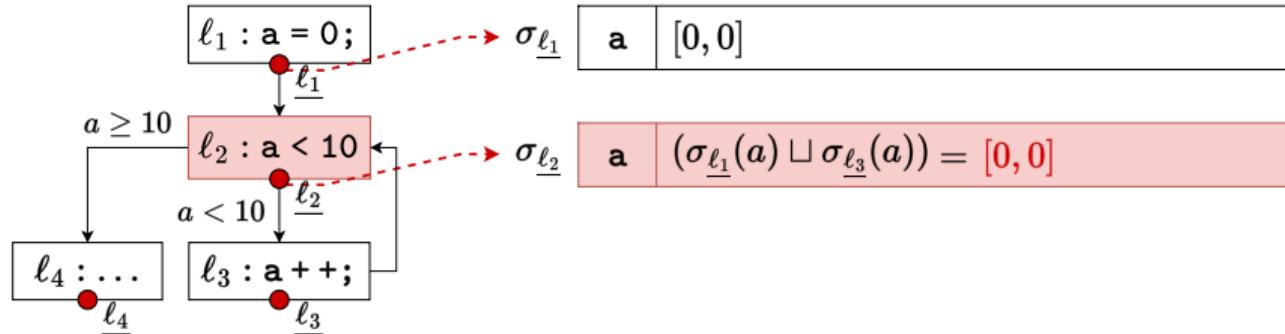
Abstract trace	Init	After analyzing ℓ_1						
$\sigma_{\underline{\ell}_1}(a)$	\perp	$[0, 0]$						
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp						
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp						
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp						



Control Flow Graph

Widening: The Loop Example

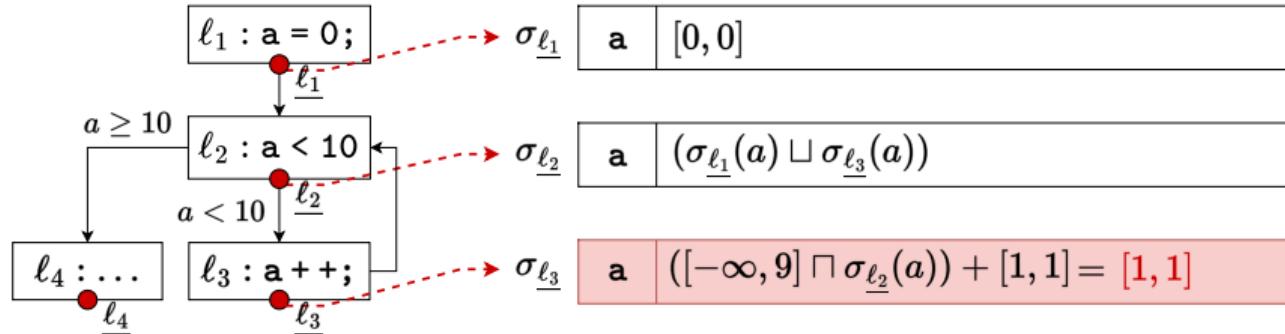
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter					
			After ℓ_2					
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]					
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]					
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp					
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp					



Control Flow Graph

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1 th loop iter						
			After ℓ_2	After ℓ_3					
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]					
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]					
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]					
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp					

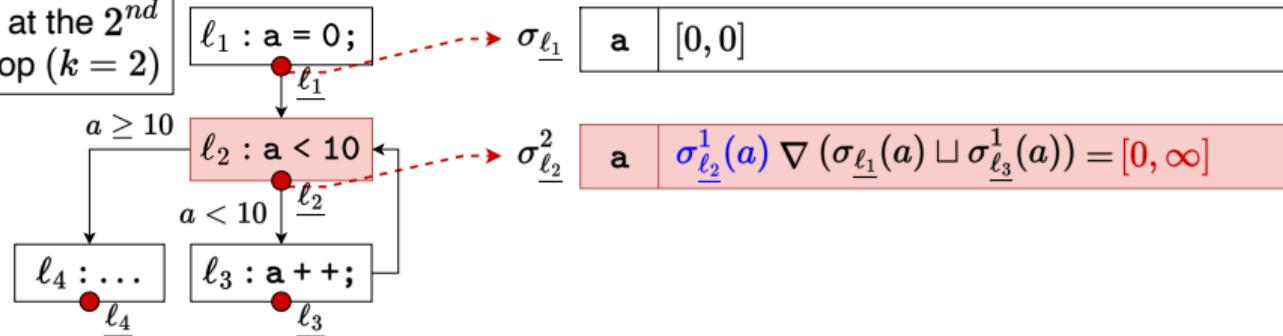


Control Flow Graph

Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1 st loop iter		2 nd loop iter			
			After ℓ_2	After ℓ_3	After ℓ_2			
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]			
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]			
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]			
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp			

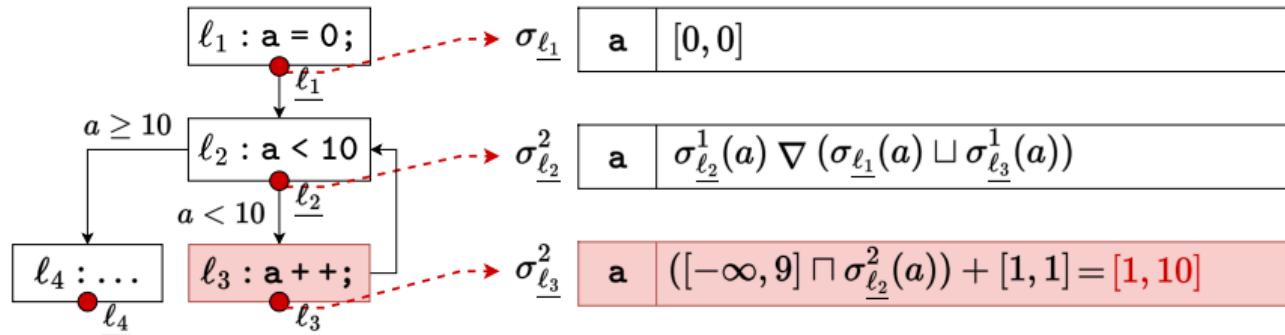
Start widening at the 2nd iteration of loop ($k = 2$)



Control Flow Graph

Widening: The Loop Example

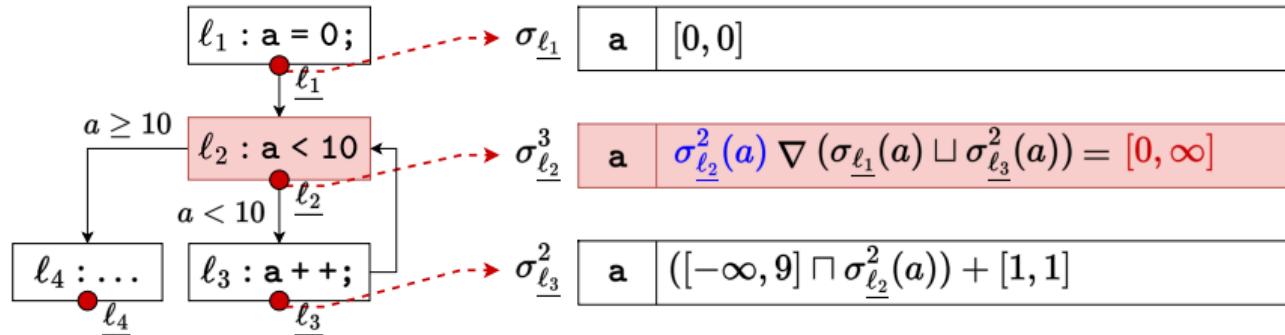
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞)	[0, ∞)	
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	



Control Flow Graph

Widening: The Loop Example

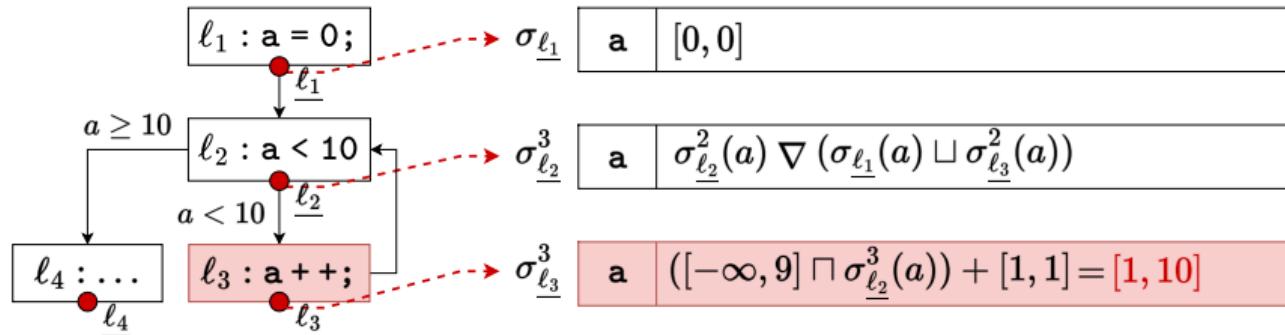
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2		
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]		
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞)	[0, ∞)	[0, ∞)		
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]		
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp		



Control Flow Graph

Widening: The Loop Example

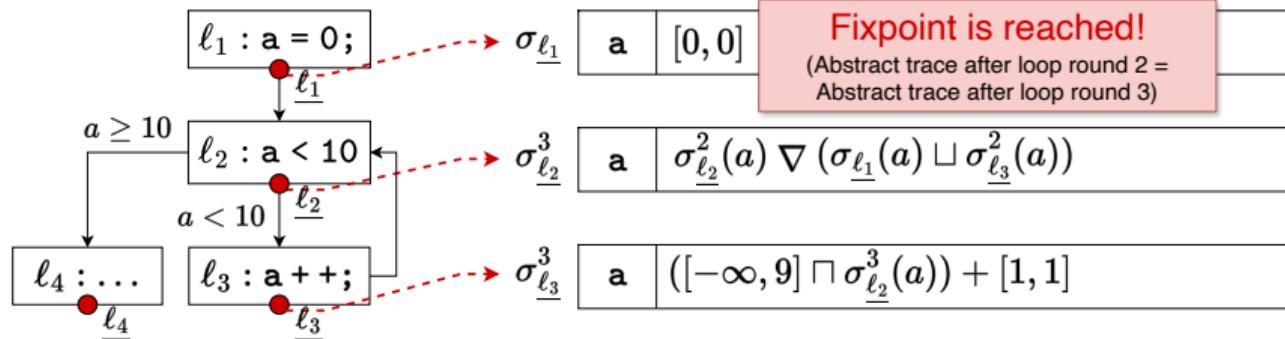
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]	
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]	[1, 10]	
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	



Control Flow Graph

Widening: The Loop Example

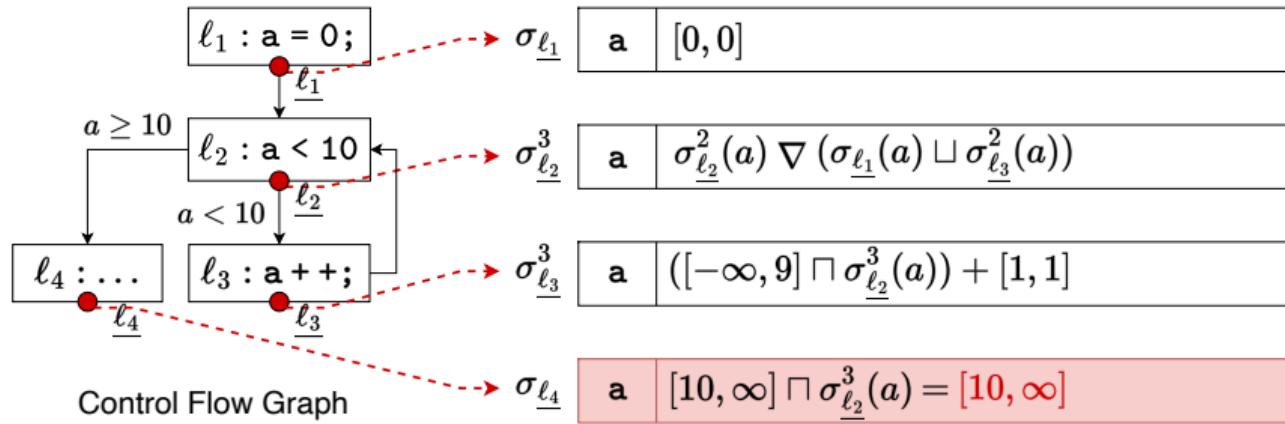
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter	
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3
$\sigma_{\underline{\ell_1}}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell_2}}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0,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



Control Flow Graph

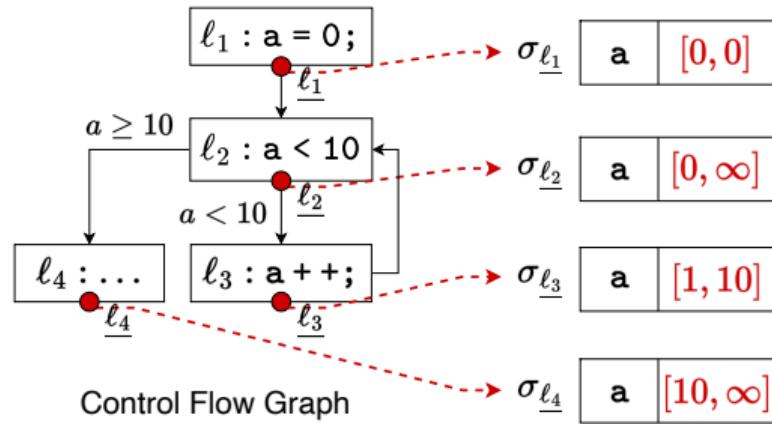
Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]	[1, 10]	[1, 10]
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	[10, ∞]



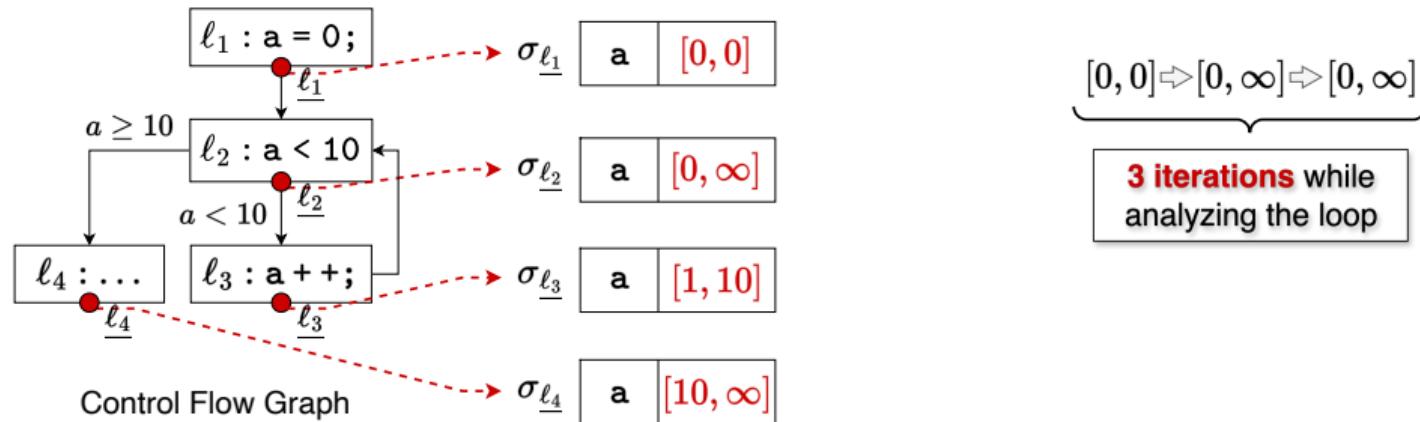
Widening: The Loop Example

Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]	[1, 10]	[1, 10]
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	[10, ∞]



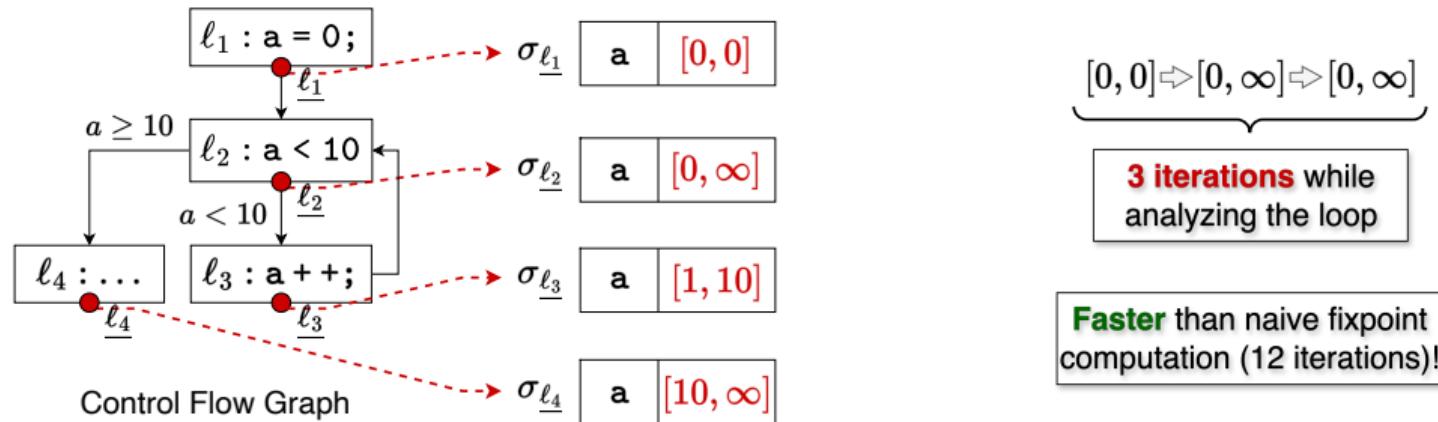
Widening: The Loop Example

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



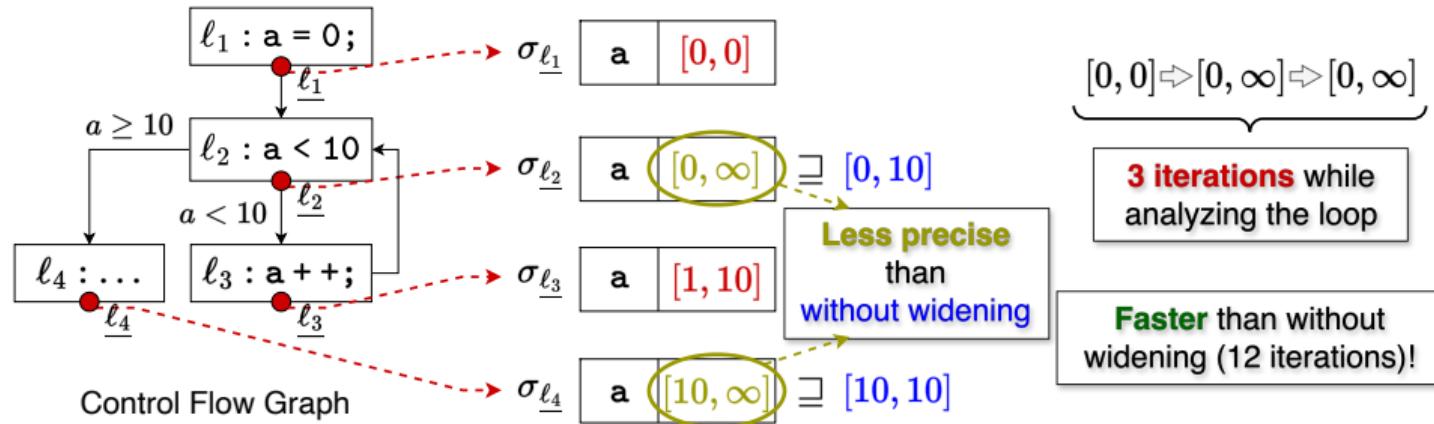
Widening: The Loop Example

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



Widening: The Loop Example

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

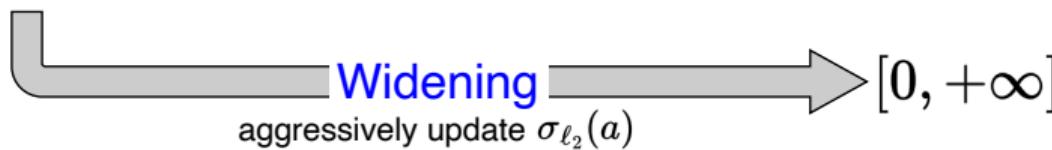


Narrowing: Precision Refinement

Narrowing technique can eliminate the precision loss after a widening operation (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 ... \rightarrow [0, 10] \rightarrow [0, 10]

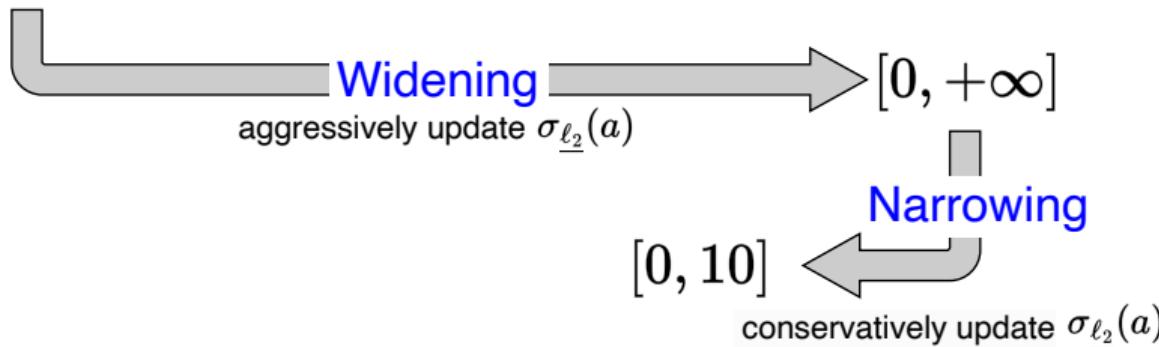


Narrowing: Precision Refinement

Narrowing technique can eliminate the precision loss after a widening operation (e.g., by improving imprecise $\sigma_{\underline{\ell_2}}$ and $\sigma_{\underline{\ell_4}}$).

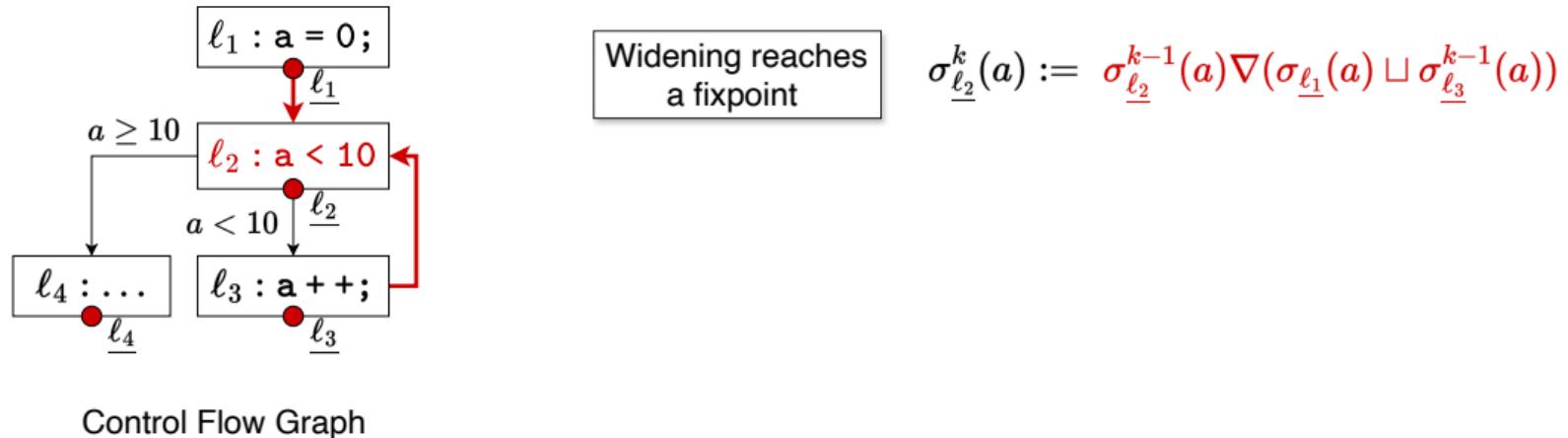
Naive fixpoint computation: value changes of $\sigma_{\underline{\ell_2}}(a)$

[0, 0] \rightarrow [0, 1] \rightarrow ... \rightarrow [0, 10] \rightarrow [0, 10]



Narrowing: Precision Refinement

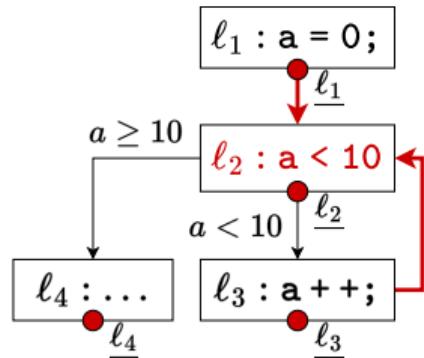
After the widening reaches a fixpoint at the k^{th} iteration when analyzing the loop, we start performing narrowing at the $(k + 1)^{th}$ to update $\sigma_{\underline{\ell}_2}$.



Control Flow Graph

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 $\sigma_{\underline{\ell}_2}$.



Widening reaches
a fixpoint

$$\sigma_{\underline{\ell}_2}^k(a) := \sigma_{\underline{\ell}_2}^{k-1}(a) \nabla (\sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}^{k-1}(a))$$

Apply narrowing operator Δ instead

$$\sigma_{\underline{\ell}_2}^{k+1}(a) := \sigma_{\underline{\ell}_2}^k(a) \Delta (\sigma_{\underline{\ell}_1}(a) \sqcup \sigma_{\underline{\ell}_3}^k(a))$$

Control Flow Graph

What is a **Narrowing Operator**?

Narrowing Operator

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

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

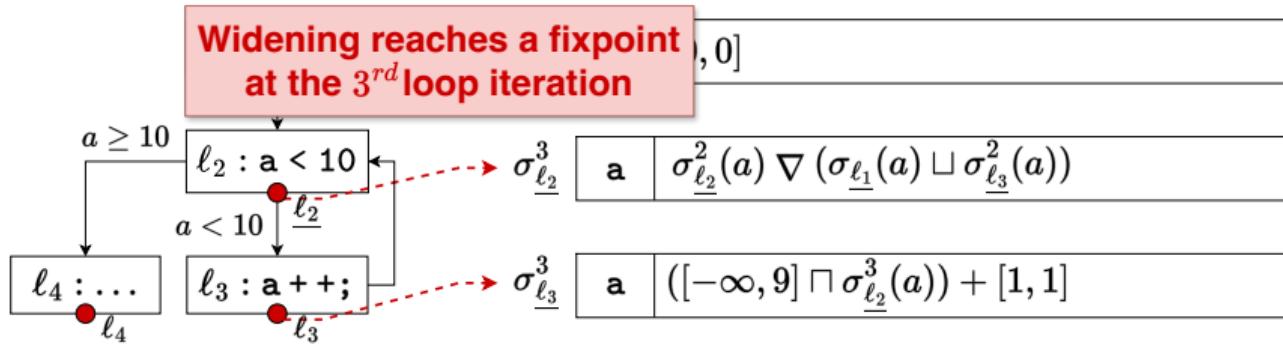
where

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

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

Narrowing: The Loop Example

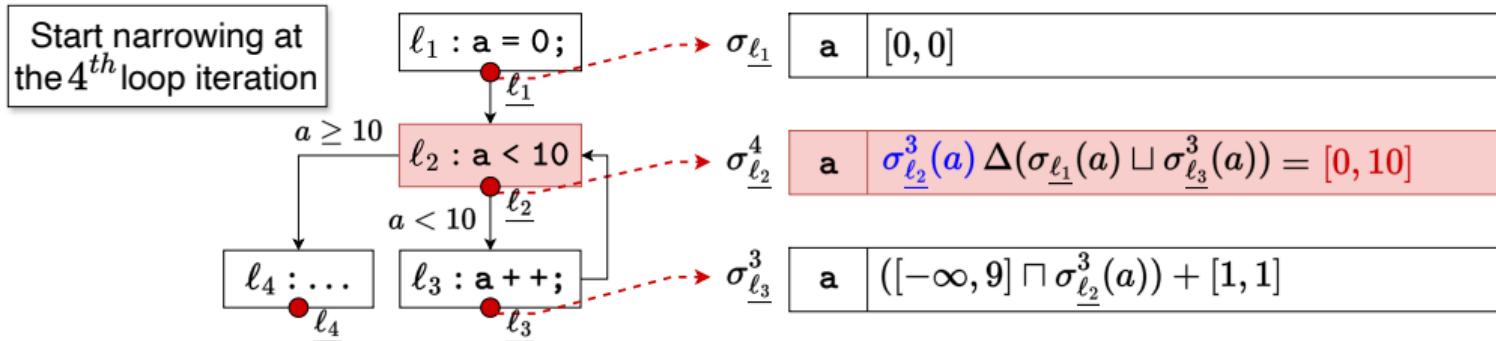
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter					
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3				
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]				
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]				
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]	[1, 10]				
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp				



Control Flow Graph

Narrowing: The Loop Example

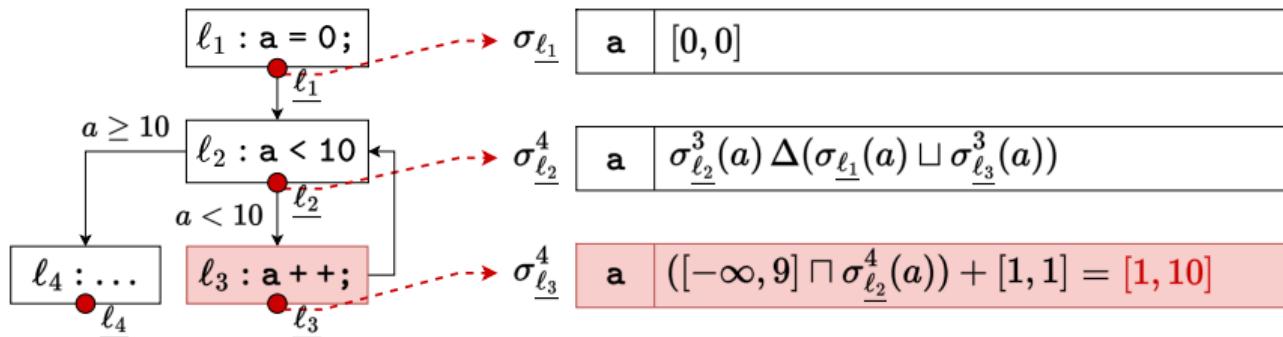
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		4 th loop iter		
			After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2	After ℓ_3	After ℓ_2		
$\sigma_{\ell_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]		
$\sigma_{\ell_2}(a)$	\perp	\perp	[0,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



Control Flow Graph

Narrowing: The Loop Example

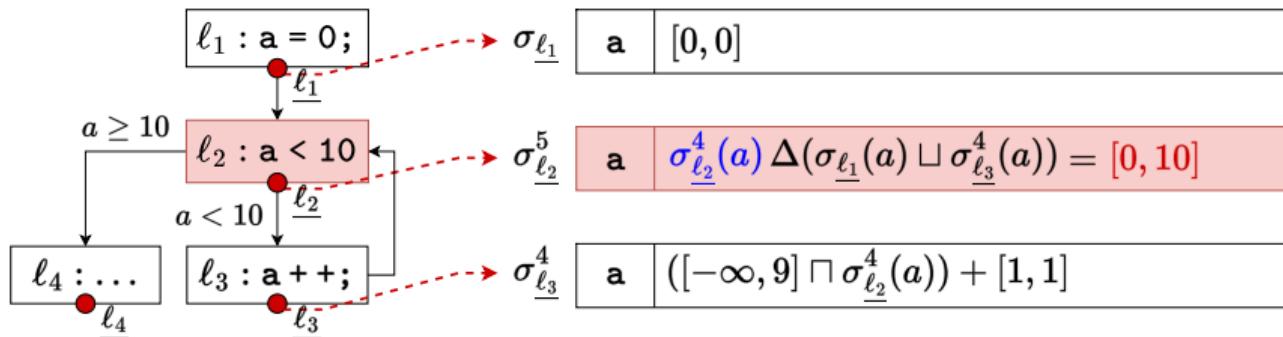
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		4 th loop iter		
			After ℓ_2	After ℓ_3							
$\sigma_{\underline{\ell}_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\underline{\ell}_2}(a)$	\perp	\perp	[0, 0]	[0, 0]	[0, ∞]	[0, ∞]	[0, ∞]	[0, ∞]	[0, 10]	[0, 10]	
$\sigma_{\underline{\ell}_3}(a)$	\perp	\perp	\perp	[1, 1]	[1, 1]	[1, 10]	[1, 10]	[1, 10]	[1, 10]	[1, 10]	
$\sigma_{\underline{\ell}_4}(a)$	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	



Control Flow Graph

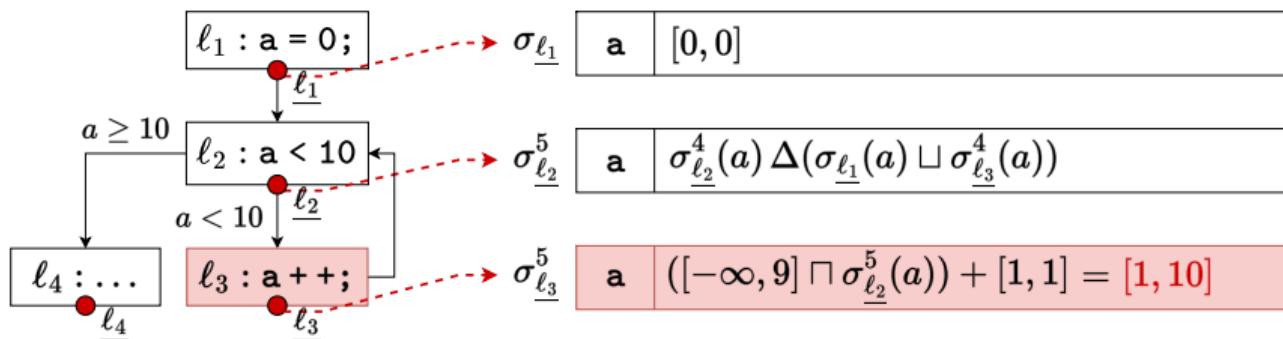
Narrowing: The Loop Example

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



Narrowing: The Loop Example

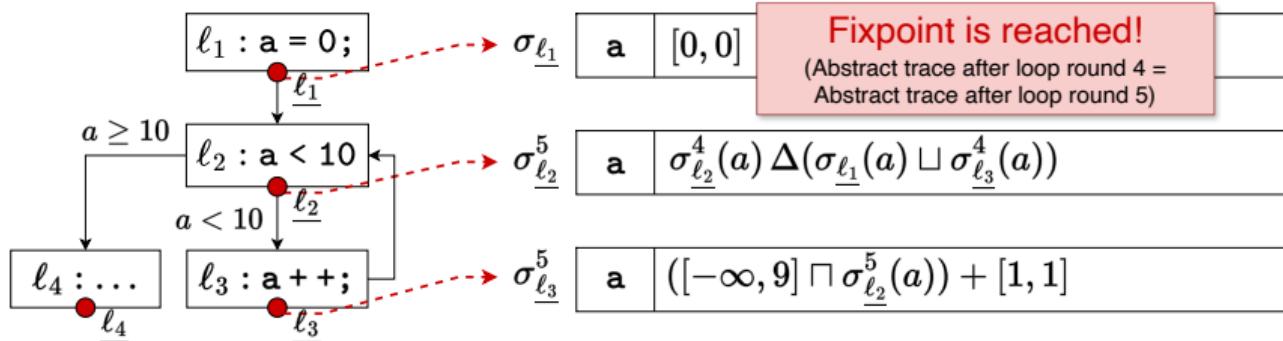
Abstract trace	Init	After analyzing ℓ_1	1 th loop iter		2 nd loop iter		3 rd loop iter		4 th loop iter		5 th loop iter		After analyzing ℓ_4
			After ℓ_2	After ℓ_3									
$\sigma_{\ell_1}(a)$	\perp	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	
$\sigma_{\ell_2}(a)$	\perp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



Control Flow Graph

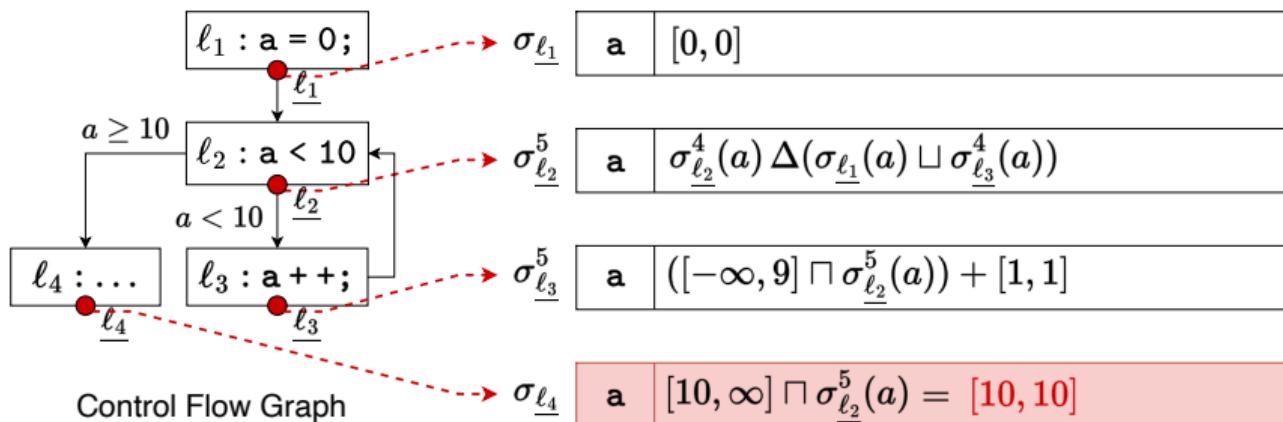
Narrowing: The Loop Example

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



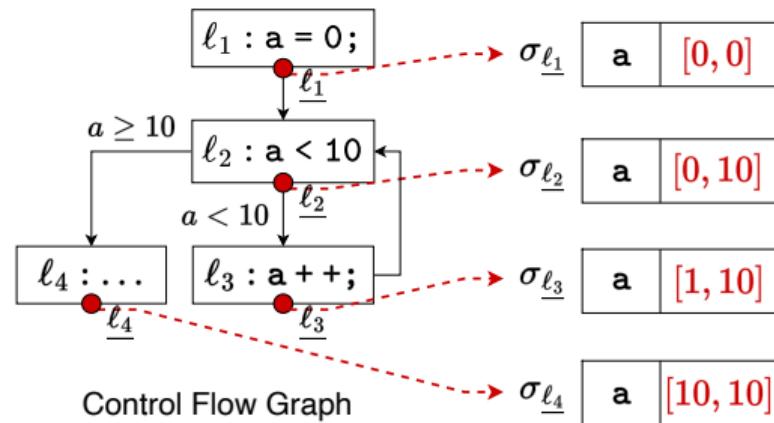
Narrowing: The Loop Example

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



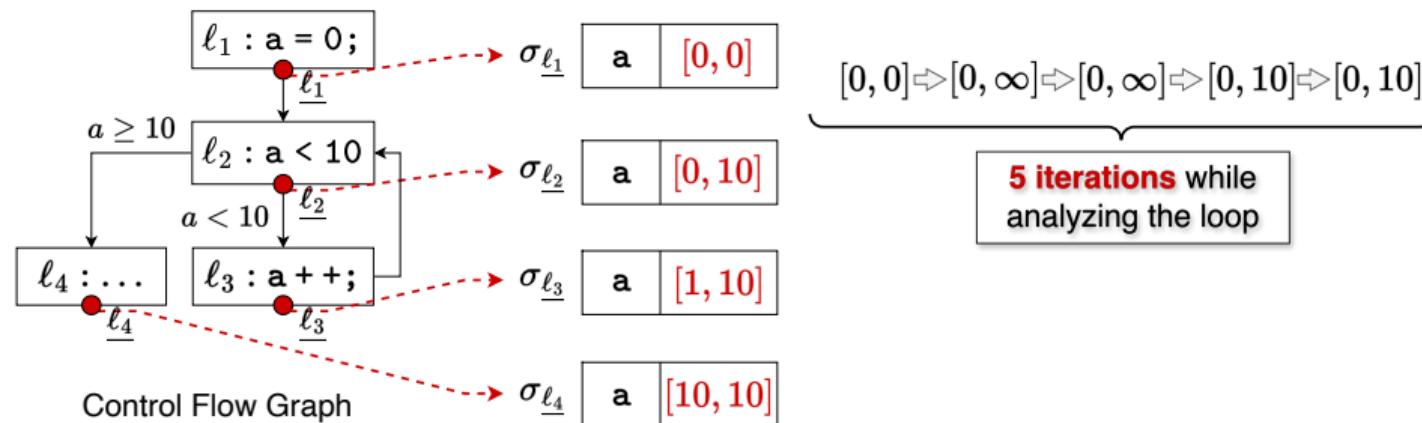
Narrowing: The Loop Example

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



Narrowing: The Loop Example

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



Narrowing: The Loop Example

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

